UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Corso di laurea magistrale in
INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE
Dipartimento di ingegneria dell'informazione

ANALISI DI CONTESTI INDUSTRIALI MEDIANTE LINGUAGGIO AADL

Analysis of industrial contexts using AADL language

Relatore:
Prof. David **SCARADOZZI**

Laureando
Dott. Enrico Maria **GABBANINI**

Correlatori:
*Dott. Nicolò **CIUCCOLI***
*Sig. Giuseppe **CINA'***

Anno accademico 2020-2021

PARTE 1

RINGRAZIAMENTI

Ringrazio mio padre e mia madre per tutto il sostegno che mi hanno dato, sempre.

Questa tesi nasce dalla volontà di conoscere e imparare a sfruttare un nuovo strumento per la modellazione di sistemi e la loro analisi, sistemi semplificati oppure complessi e realistici.

Civitanavi System aveva come obiettivo quello di capire se AADL potesse essere un valido strumento da introdurre nel loro sistema di progettazione, ha avviato così una collaborazione col Professor Scaradozzi che ci ha proposto AADL come argomento di tirocinio e tesi.

Un grande ringraziamento va a David Scaradozzi per averci proposto questo lavoro che oltre ad averci aperto le porte ad un argomento incredibilmente utile e potente, ci ha dato l'opportunità di affacciarci su una realtà aziendale e lavorativa.

Un grande ringraziamento va a Civitanavi System per averci fornito il materiale relativo ad una loro IMU sulla quale applicare gli studi fatti su AADL, dandoci così la possibilità di modellarla e analizzarla.

Civitanavi deve essere ringraziata anche per averci dato la possibilità di visitare l'azienda e conoscere di persona una parte del loro organico, che ci hanno affiancato durante gli 8 mesi in cui si è svolto il lavoro.

I membri dell'organico di Civitanavi che ci hanno seguito e guidato sono Giuseppe Cinà e Dario Donati.

Un imprescindibile ringraziamento va proprio a Giuseppe Cinà che ci ha seguito fin dal primo giorno, restando sempre disponibile per qualsiasi informazione, fornendoci il materiale necessario rimanendo sempre comprensivo nei momenti in cui il lavoro veniva sospeso per portare avanti gli esami universitari rimanenti. Una persona incredibile dalle grandi conoscenze che ci ha fornito un supporto essenziale.

Ringraziamo Dario Donati per il supporto fornitoci in fase di analisi, procurandoci i dati utili a modellare il sistema in AADL.

Infine, ma non per importanza, un ringraziamento va a Veronica Bartolucci e Niccolò Ciuccoli per il supporto fornito durante tutto il percorso e in particolar modo durante gli esami.

Grazie.

P.S.

Ora ci saranno dei ringraziamenti più personali, sono informali e direttamente dal Cuore.

Grazie a tutti, Stronzi.

INDICE

Introduzione

In fase di sviluppo, è chiaro che si venga a creare un gap tra il linguaggio di programmazione e il linguaggio macchina. Occorre dunque, attraverso differenti tecniche, costruire un ponte che renda possibile la comunicazione tra questi due differenti tipi di linguaggi.

Negli anni, gli ingegneri hanno sviluppato nuovi linguaggi che avvicinano il linguaggio macchina al linguaggio di programmazione, in modo da rendere più chiaro il funzionamento. Nella maggior parte dei casi, più si astrae il linguaggio, più facile risulta analizzare, capire e sviluppare programmi senza errori. Un alto livello di astrazione è utile per focalizzarsi su ciò che è realmente importante, il modello è un alto livello di astrazione del programma sulla quale è possibile eseguire studi preliminari dettagliati. Questo approccio è noto come Model Based Engineering (MBE), infatti il modello è al centro delle attività ingegneristiche, offrendo supporto in particolar modo ad analisi e verifica.

Un appropriato linguaggio di modellazione fornisce le tecniche necessarie di sintassi e semantica per descrivere correttamente e progettare gli aspetti chiave del sistema (concorrenza, organizzazione, modularità, scheduling…). Il livello di astrazione, così come la semantica del linguaggio sono fattori chiave di quali analisi possono essere eseguite tramite il linguaggio.

Il modello, va intese come astrazione del sistema che deve essere usato nella fase di progettazione per analizzare la correttezza del lavoro svolto e impostato. Utilizzando il modello e gli strumenti di analisi, è possibile scoprire in anticipo problemi che potrebbero, altrimenti, venire fuori in fasi successive. Ciò comporta una significativa riduzione di tempi e costi di sviluppo. Infatti, in questo modo, si evita di dover tornare alle fasi iniziali di sviluppo ogni volta che sorge o viene scoperto un nuovo problema. Ovviamente, condizione necessaria per far si che tutto funzioni, è quello di sviluppare un modello corretto e coerente con il lavoro da svolgere, ossia un modello che possa poi essere validato.

Al fine di avere un buon sviluppo MBE occorre:

- Una semantica ricca e precisa;
- Notazione grafica: una rappresentazione grafica risulta facile da capire e può essere riutilizzata per creare della documentazione;
- User_friendly tool: un buon tool può offrire esperienze di utilizzo migliori all'operatore e può cambiare radicalmente il modo in cui viene visto il modello.
- Un formato aperto e interoperabile: in questo modo cambiando tool è possibile riutilizzare e riadattare lo stesso modello.

Esistono già differenti tipi di linguaggi di modellazione e tools che si distinguono principalmente per il tipo di approccio utilizzato: quello funzionale e quello architetturale. Il modello funzionale cerca di rispondere alla domanda "cosa" (cosa fa il sistema?), mentre un linguaggio architetturale cerca di rispondere alla domanda "come" (come il sistema svolge e gestisce una determinata operazione). AADL rientra nella seconda casistica, ossia quella dei linguaggi architetturali.

Con "software architecture" ci si riferisce alla struttura fondamentale di un sistema software, ossia la disciplina di creare sia la struttura, che la documentazione della struttura stessa. Ogni struttura è composta da elementi software, relazionati tra loro, e proprietà sia degli elementi sia delle relazioni. Dunque, l'architettura software è in grado di mostrare come il sistema è organizzato, quali elementi sono connessi e quali parti sono state usate per realizzarlo.

Tra i linguaggi di tipo architetturale rientra AADL (Architecture Analysis and Design Language) con l'obiettivo del real time, per i sistemi embedded. Questo linguaggio si focalizza sul design di sistema usando un linguaggio ricco, semanticamente formale che può essere utilizzato per analizzare e generare il sistema. Tramite AADL, una volta modellato il sistema, è possibile analizzarlo, generare l'implementazione e derivare i vari test. Inoltre, attraverso l'analisi, la simulazione e una prototipazione rapida è possibile affinare il modello ed eventualmente procedere con una nuova implementazione.

Dunque, l'obiettivo che ci si è posti è quello di riuscire a creare un modello del sistema che possa essere analizzato ed ampliato garantendo il flow dei requisiti di sistema a Hardware e Software. Si procede con l'utilizzo di AADL in quanto:

- I sistemi che si progettano sono sempre più complessi e difficili da analizzare. Infatti, nonostante le normative applicabili agli sviluppi Hardware e Software siano rigorose, la fase di integrazione del sistema resta sempre complicata.
- Scoprire le anomalie o marginalità a valle dell'implementazione porta ritardi ed extracosti spesso non sostenibili.
- Validare l'architettura di un sistema safety critical richiede lo svolgimento di analisi dedicate che necessitano di una visione chiara del sistema.
- La disponibilità di un modello virtuale del sistema permette di effettuare analisi comparative, modificando in tempo reale il modello e verificarne il comportamento.
- La possibilità di procedere per livelli di dettaglio fa si che la verifica possa essere incrementale.
- Garantire un approccio uniforme, strutturato e coordinato al design di Sistema, Hardware e Software

Al di là dei vantaggi immediati derivanti dalla disponibilità di un gemello virtuale (Digital Twin) del sistema sul quale effettuare analisi e sperimentazioni, a costi contenuti sin dalle fasi iniziali del progetto, vi sono altri vantaggi derivanti dall'utilizzo di AADL nella progettazione del sistema:

- La possibilità di supportare le analisi di Safety – FHA (Functional Hazard Analysis), FMEA (Failure Mode and Effects Analysis), FTA (Fautl Tree Analysis), MTBF (Mean Time Between Failures) - documentando opportunamente il modello.
- Nel caso di modifiche limitate ad un sistema già modellato la possibilità di verificarne anticipatamente l'impatto a livello di sistema.
- Fornire input a tool di simulazione e di generazione automatica del codice.
- Svolge uno sviluppo 'Architecture driven', creando un percorso lineare ed omogeneo dalla fase di progettazione di sistema sino all'integrazione.
- Uso di una sintassi standardizzata e precisa per descrive il Sistema.
- Tracciare le evoluzioni del modello del Sistema lungo l'arco del Progetto.

L'utilizzo di AADL per Civitanavi rappresenta un passo avanti nel miglioramento e nell'efficientamento dei processi aziendali. La richiesta sempre più frequente di sistemi safety critical impone l'aderenza dei processi di sviluppo – Sistema, Hardware e Software- a standard stringenti, onerosi da soddisfare e da mantenere. Dunque, AADL costituisce un aspetto di innovazione:

- AADL è già utilizzato in ambito industriale, spazio ed avionico e quindi è un tool già noto.
- Promuove un concetto di Virtual Integration, dove il sistema viene modellizzato e verificato virtualmente prima di essere realizzato fisicamente.
- Facilitare l'applicazione di modifiche a sistemi già realizzati analizzandone gli impatti prima sul modello virtuale.

- AADL offre un percorso verso tool di modellazione di sistema e progettazione del software che possono rappresentare un'ulteriore evoluzione dei processi aziendali nel raccordare la progettazione di sistema con il design software.
- Favorire la creazione di un processo lineare ed omogeneo che parte dalla progettazione di sistema, arriva al design hardware/software passando attraverso la safety, garantendo sempre la massima visibilità sull'evoluzione del design, e consentendo di confrontarlo continuamente con requisiti applicabili.

Dunque, l'obiettivo che ci si prefigge è:

- quello di effettuare la modellazione di una IMU già sviluppata da Civitanavi arrivando sino al livello di dettaglio necessario per effettuare le analisi di timing e safety.
- Svolgere sul modello le analisi funzionali e di safety valutando i risultati con quelli già attenuti con metodologia 'classica'.
- Identificare un processo che consenta ai vari enti aziendali coinvolti nella progettazione di avere un ruolo attivo nello sviluppo del sistema, definendo la metodologia da adottarsi ed il livello di coinvolgimento di ogni singolo ente.

Una volta completata la fase di modellazione in AADL dell'IMU, il passo successivo è stato quello di andare a svolgere delle analisi specifiche di Latency e di Safety.

L'analisi della latenza viene eseguita su modelli AADL che includono flussi end-to-end e calcola la latenza minima e massima tenendo conto di un'ampia gamma di contributori di latenza.
Il calcolo della latenza si basa sui contributori assegnati ai diversi elementi architettonici e alle informazioni di progettazione, man mano che i progetti si evolvono. Il modello AADL può variare da un'architettura funzionale con budget di latenza a diversi livelli di scomposizione, a un'architettura di attività e comunicazione con velocità di esecuzione mappate su una piattaforma hardware che supporta il partizionamento. La fedeltà dell'analisi è determinata dai dettagli nel modello AADL.
I risultati vengono riportati in un formato di risultati comune e in un formato di foglio di calcolo insieme ai dettagli di ciascun contributore alla latenza. L'analisi della latenza può essere parametrizzata da alcune impostazioni delle preferenze che consentono agli utenti di esplorare le variazioni architettoniche senza dover modificare i dettagli nel modello (ad esempio se il sistema si comporta come un sistema asincrono o sincrono).

Nell'analisi di safety invece andiamo a vedere come si può eseguire un'analisi FHA (functional hazard assessment), FMEA (ex. fault impact analysis), FTA (fault tree analysis) di un sistema, tramite l'uso di AADL, Error Model V2 (EMV2) e OSATE.

Vedremo come AADL e EMV2 possono essere utilizzati in questo processo e dimostreremo l'uso di EMV2 a tre livelli di astrazione:

1. Propagazioni e flussi di errori: tramite un'analisi dell'impatto (FMEA) tipicamente da un'unica fonte, procedendo in avanti, si traccia l'impatto degli errori; un'analisi di propagazione all'indietro per identificare tutti i potenziali contributi a un evento catastrofico o importante.

2. Specifiche del comportamento degli errori dei componenti: identificazione delle modalità di guasto, tipi di guasti dei componenti (eventi di errore), logica del comportamento degli errori per riflettere la ridondanza dell'input esterno e la ridondanza dei sottocomponenti.

3. Generazione di un albero degli errori FTA con probabilità del guasto utile per determinare l'affidabilità iniziale del sistema. Può essere utilizzato per generare un albero dei guasti delle parti COMPOSITE.

Nei capitoli successivi verrà sintetizzato il lavoro svolto nella creazione del modello AADL del sistema fornito da Civitanavi. Nel primo capitolo verrà introdotto AADL per fornire delle nozioni base, utili alla comprensione dei capitoli successivi.

Buona lettura.

CAPITOLO 1: Analisi di latenza

1.0 Introduzione al capitolo

Come detto nell'introduzione, AADL fornisce gli strumenti per effettuare analisi preliminari su una grande varietà di caratteristiche del sistema (FTA, BUS LOAD, FHA, LATENCY, fault Impact, …).
In questo capitolo verrà trattata l'Analisi di Latenza applicata al sistema, per studiare le tempistiche del Software dal momento dell'acquisizione dati, tutte le elaborazioni previste, fino al momento della restituzione dei dati in uscita.

1.1 Concetto di latenza

La latenza è il tempo che trascorre dalla creazione del dato all'utilizzo del dato, quello che si desidera è modellare e realizzare un sistema real-time, cioè che garantisca una latenza fissa, meglio ancora, nulla. Essere un sistema Real-Time significa DETERMINISMO: un'azione o un dato viene generato (pensata) e consegnato (eseguita) in momento deterministico.
La latenza esiste poiché è fisicamente impossibile, per ora, riprodurre il real-time perfetto, solo il corpo umano è real-time ma quando si bevono alcolici, per esempio, questi ultimi sono contributori di latenza al sistema.

1.2 Caso di studio

Nel Sistema Inerziale analizzato, è stato modellato tutto il data flow del segnale proveniente dall'Accelerometro X, dal momento della sua nascita nel *device Accel_X* fino all'uscita dal Firmware. È stato analizzato il percorso del dato che, dopo essere stato letto dal Firmware, viene fornito al Software che lo calibra e compensa, lo elabora e poi lo restituisce al Firmware pronto per essere usato dall'algoritmo di controllo esterno.
Si vuole analizzare la latenza durante le operazioni di lettura ed elaborazione, che vengono definite come le principali e più importanti operazioni del sistema. E quindi stato analizzato il dataflow attraverso questo percorso, prendendo in considerazioni tutti gli elementi attraversati e valutando il loro contributo di latenza e specificandolo tramite delle proprietà apposite.
Bisogna quindi considerare le definizioni dei flussi fornite da AADL e stabilire tramite un **flow source(**l'inizio**)**, i relativi **flow path** attraverso gli elementi, e un **flow sink**(fine) dove muore il dato.

```
-------Dalla Sensing Elements che contiene il device Accel X----------

F1: flow source Accel_Value_x;

-------Presente nel PS_Module----------

IND_Flow: flow path RAW_Data -> Inertial_Data_Message;

--------Presente nel system [redacted] come fine del dato---------

F1: flow sink Acc_Value_x -> C1 -> PL_MODULE.F1;

-----------------
```

Figura 1 Flow source, path e sink descrittivi dell'end-to-end flow

Va strutturato questo percorso tra i flow della Type ma soprattutto vanno ben definiti quelli nelle varie Implementation. Una volta modellato questo percorso attraverso il sistema si può definire un **end-to-end flow** riferito a quel percorso sulla quale viene effettuata l'analisi da parte di OSATE2.

```
flows
------------------End to End flow of Software Data elaboration--------------------
ETEF1: end to end flow PL_MODULE.F10 -> C14 -> PS_MODULE.IND_Flow -> C16 ->
PL_MODULE.FF {latency => 500us .. 2ms;};

properties
      actual_connection_binding => (reference (AXI_BUS)) applies to C14, C16;
```

Figura 2 end-to-end definite nella Component Implementation del SoC

PL_MODULE.F10: flow che descrive la lettura del dato da parte del Firmware e il suo tragitto attraverso la PL.

C14 e **C16**: sono le connessioni che legano la PL e la PS attraverso le quali i due moduli si scambiano i dati e sono rappresentative dell'AXI BUS

PS_MODULE.IND_Flow: flow che descrive il percorso attraverso la PS e quindi attraverso il Software e i sugli algoritmi di C&S.

PL_MODULE.FF: flow sink che descrive la destinazione finale del dato

Molti fattori possono influire sulla latenza end-to-end della porzione di sistema:

- Velocità processore
- Uso di dati condivisi
- Contesa bus
- Jitter
- Tempo di esecuzione dei thread
- Periodo
- Scadenze
- Protocolli di programmazione e comunicazione

Per eseguire l'analisi della latenza, il modello deve specificare:

- Il flusso di dati dal produttore al consumatore
- Tutti i collaboratori degli elementi di modellazione che potrebbero influire sulla latenza del sistema

1.3 Contributori di latenza

Esistono numerosi fattori che contribuiscono alla latenza di un sistema. La tabella sottostante riassume ogni contributore di latenza e i suoi componenti AADL rilevanti con le relative proprietà.
Esistono due categorie principali di contributori di latenza:

- **Categoria di pianificazione**: acquisisce la velocità di esecuzione dei componenti e produce/utilizza i dati. Importante è il modo in cui i componenti vengono programmati (periodici, sporadici).
  All'interno della categoria sono inclusi:
  - Il **tempo** di esecuzione rappresenta il tempo necessario per eseguire il componente stesso e viene acquisito utilizzando la proprietà *Compute_Execution_Time* in *subcomponent*, *thread* o *device*.
  - Il **tipo di attività** o, più specificamente, il modo in cui viene attivata o inviata. Un'attività o un dispositivo è periodico o sporadico. I componenti del periodo vengono eseguiti a un tasso fisso (il periodo). I componenti sporadici vengono eseguiti solo quando ricevono dati o segnali sulle porte di ingresso. Questa caratteristica viene acquisita utilizzando la *Dispatch_Protocol* associata a un *device* o *thread* e può assumere il valore *periodic o sporadic.*
  - Il **periodo** un'attività o di un dispositivo specifica il tasso di esecuzione dei componenti periodici. In AADL, il periodo viene acquisito utilizzando la proprietà *Period* su un *thread* o un *device*, assegnandole un valore di tempo.
  - **Scadenza di un'attività** o di un dispositivo: rappresenta quando si suppone che il componente abbia completato un processo e inviato risultati. La scadenza viene acquisita utilizzando la proprietà *Deadline* in un *thread* o in un *device*.
  - Criteri di pianificazione delle **execution platform**. Quando le attività vengono eseguite contemporaneamente su un processore condiviso, il sistema operativo deve pianificare le attività in base a criteri specifici. In AADL, il protocollo di pianificazione viene specificato utilizzando i criteri di pianificazione nei *processor* e *virtual processor.*

```
properties
    Dispatch_Protocol => Periodic;
    Period => 250us;
    compute_execution_time => 150us .. 300us;
```

Figura 3 Esempio di proprietà della categoria di pianificazione

- La **categoria di trasporto** si riferisce alla velocità di trasferimento dei dati tra nodi distribuiti. Quando i dati scorrono tra componenti situati su processori diversi, usano bus per scambiare i dati. Il bus hardware e i protocolli di trasporto influenzano la latenza end-to-end. Questi fattori sono legati alla categoria dei trasporti:
    - **Tempo di acquisizione e trasmissione del bus**. Quando i dati vengono trasportati utilizzando un bus, due fattori influiranno sulla latenza end-to-end: il tempo di acquisizione del bus e il tempo di trasmissione. Il tempo di acquisizione del bus è il tempo necessario affinché il bus condiviso sia pronto per la trasmissione dei dati.
      Il tempo totale di acquisizione del bus viene acquisito utilizzando un intervallo di tempo. Il valore inferiore dell'intervallo rappresenta il momento migliore per acquisire il bus, mentre il valore più alto rappresenta il momento peggiore. Il tempo di trasmissione dipende dalle dimensioni dei dati trasportati e dalla velocità del bus (quanti byte possono essere trasportati al secondo). Queste caratteristiche vengono acquisite utilizzando la *Transmission_Time* di un componente bus. La proprietà è un record con due membri: un membro *Fixed* che caratterizza il tempo necessario per acquisire il collegamento e il membro *Per_Byte* che specifica il tempo necessario per trasportare un byte di dati utilizzando questo bus.
    - Un **protocollo di comunicazione** impiega del tempo per organizzare e sincronizzare i dati. Un protocollo viene acquisito utilizzando un *virtual bus* e le connessioni che utilizzano questo protocollo sono associate ad esso. Ogni *virtual bus* (protocollo) può introdurre la propria latenza utilizzando la proprietà *latency*.
    - La **dimensione dei dati trasportati** su un bus influisce sulla latenza (più dati devi trasportare, più tempo è necessari per completare la trasmissione). La *Transmission_Time* di un bus caratterizza il tempo necessario per il trasporto di un byte. Queste informazioni devono essere utilizzate con le dimensioni dei dati per calcolare il tempo necessario per trasportare i dati dall'origine alla destinazione. Questo viene fatto con la *Data_Sizer* del set di proprietà standard.

```
properties
    Latency => 1 ms .. 2 ms;
    SEI::BandWidthCapacity => 2000.0 bitsps;
    Transmission_Time => [ Fixed => 500us .. 2ms;];
                      --   PerByte => 1 us .. 1 us; ];
```

Figura 4 Esempio di proprietà della categoria di trasporto

latenza collaboratore	**Modello AADL**
Esecuzione dei componenti Ore	**Compute_Execution_Time** in un *device*, *thread* o *subprogram* AADL.
spedizione protocollo	**Dispatch_Protocol** proprietà su un *device* o *thread* AADL. Il valore è periodico o sporadico.
componente periodo	Proprietà **Period** in un *device* o *thread* AADL. Il valore è un valore di tempo con un'unità (ad esempio 100ms).
componente scadenza	Proprietà **Deadline** in un *device* o *thread* AADL. Il valore è un valore di tempo con un'unità.
Politica di scheduling del processore	Proprietà su un componente *processor* AADL che sta pianificando le attività. Questa proprietà è specificata utilizzando il **scheduling_protocol** per i sistemi operativi non partizionanti e **module_schedule** per i sistemi ARINC653
Acquisizione bus e Tempo di trasmissione	**Transmission_time** proprietà su un componente bus AADL. Questa proprietà viene utilizzata per ogni connessione associata al bus correlato. Transmission_time => [fixed => 2 ms . . 4 ms; Perbyte => 10 us . . 20 us;] ;
protocollo	Proprietà **Latency** su un bus virtuale AADL
Dimensioni dei dati	**Data_Size** sugli elementi di dati associati alle porte dati AADL e alle porte dati degli eventi.

Figura 5 Tabella dei contributori di latenza

1.4 Calcolo della latenza

Gli strumenti elaborano ogni dichiarazione di flusso end-to-end contenuta in una specifica istanza di sistema. Per ogni dichiarazione di flusso end-to-end, lo strumento calcola due valori:

- La **latenza prevista** è il valore della proprietà di latenza AADL su ogni elemento di flusso
- La **latenza effettiva** è quella calcolata in base ai fattori di latenza

Lo strumento di analisi calcola le latenze effettive e previste per ogni elemento di flusso del flusso end-to-end in esame. Lo strumento recupera le latenze previste ed effettive per ogni elemento di flusso del flusso end-to-end e le somma.

$$latency_{end\ to\ end\ flow} = \sum_{0}^{flows\ elements\ (end\ to\ flow)} latency_{flow\ element}$$

Figura 6 Calcolo della latenza di un end-to-end flow

1.5 Analisi della latenza del caso di studio

L'analisi della latenza viene eseguita su modelli AADL che includono flussi end-to-end e calcola la latenza minima e massima tenendo conto di un'ampia gamma di collaboratori di latenza. Lo fa in base ai budget di latenza assegnati a diversi elementi architettonici e in base alle informazioni di progettazione man mano che i progetti architettonici si evolvono. Il modello AADL può variare da un'architettura funzionale con budget di latenza a diversi livelli di decomposizione, a un'architettura di attività e comunicazione con tassi di esecuzione mappati a una piattaforma hardware che supporta il partizionamento. La fedeltà dell'analisi è determinata dai dettagli nel modello AADL.
I risultati vengono riportati in un formato di risultato comune e in un formato foglio di calcolo insieme ai dettagli di ogni collaboratore di latenza. L'analisi della latenza può essere parametrizzata da alcune impostazioni delle preferenze che consentono agli utenti di esplorare le variazioni architettoniche senza dover modificare i dettagli nel modello, ad esempio se il sistema si comporta come un sistema asincrono o sincrono.
L'analisi della latenza del flusso supporta le impostazioni seguenti:

- **Tipo di sistema**. Utilizzato per valutare la latenza di campionamento tra componenti periodici non intrinsecamente sincroni (thread o partizioni sullo stesso processore) o specificati in modo esplicito come sincroni facendo riferimento allo stesso valore della proprietà *Reference_Time*.
    - **Sistema asincrono (AS)**: i componenti non sono sincronizzati nel tempo, cioè le spedizioni possono avere un cambio di tempo.
    - **Sistema sincrono (SS)**: i componenti sono sincronizzati nel tempo, cioè le spedizioni periodiche sono allineate tra i sistemi.
- **Criteri di output delle partizioni**. Utilizzato per riflettere il contributo di latenza tramite comunicazione tra partizioni a causa di diversi criteri di intercondizione nei sistemi partizionati.
    - **Fine partizione (PE):** si supponga che le connessioni tra partizioni siano disponibili alla fine della partizione la cui attività invia dati. Se un'attività nella partizione A invia dati a un'attività nella partizione B, il successivo riceverà i dati nello stesso frame principale se la partizione B viene eseguita dopo la partizione A.

    - **Frame principale ritardato (MF):** si supponga che le connessioni tra partizioni siano svuotate/realizzate alla fine del frame principale. Se un'attività nella partizione A invia dati a un'attività nella partizione B, i nuovi dati saranno disponibili solo dopo il completamento di tutte le partizioni rimanenti, indipendentemente dall'ordine di esecuzione della partizione A o della partizione B.

- **Per l'utilizzo del tempo di elaborazione peggiore**: gli utenti possono scegliere tra la scadenza e il tempo di esecuzione del calcolo peggiore come tempo di elaborazione del caso peggiore. Nel migliore dei casi usiamo sempre il tempo di esecuzione del calcolo. Questa impostazione è rilevante solo quando non è disponibile alcun tempo di risposta.

    - **Scadenza (DL):** la scadenza rappresenta il tempo di completamento peggiore supponendo che le attività siano pianificabili.

    - **Tempo massimo di esecuzione del calcolo (ET):** il tempo massimo di esecuzione del calcolo è utile quando si considera il tempo di elaborazione senza programmazione delle risorse.

- **Per la latenza di accodamento best-case sulle porte in ingresso**: influisce sul modo in cui viene determinato il ritardo di accodamento dei casi migliori. Nel peggiore dei casi usiamo sempre la coda completa.

    - **Supponi coda vuota (EQ):** nessun ritardo poiché si presuppone che la coda sia vuota.

    - **Supponi coda piena (Full Queue):** utilizzare il tempo minimo di esecuzione del calcolo per le dimensioni della coda per determinare il tempo di accodamento dei casi migliore.

- **Disabilita la latenza di accodamento nei risultati:** determina se il ritardo di accodamento nel peggiore dei casi viene segnalato per i bus asincroni.

    - **Disabilita:** il ritardo di accodamento nel peggiore dei casi viene sempre segnalato come 0.

    - **Abilita:** il ritardo di accodamento nel peggiore dei casi viene riportato come calcolato di seguito.

Il report dei risultati dell'analisi della latenza è costituito da report dettagliati separati per ogni flusso end-to-end e per ogni modalità operativa, se il modello AADL include specifiche di modalità.
L'intestazione di ogni report identifica il nome del flusso end-to-end analizzato, l'implementazione del componente che contiene la dichiarazione di flusso end-to-end, la modalità operativa analizzata (se le modalità vengono dichiarate) e le impostazioni delle preferenze utilizzate.
Ogni report include una voce separata per ogni collaboratore di latenza:

- I componenti contribuiscono all'elaborazione della latenza e al campionamento o alla latenza di accodamento. La latenza di campionamento è influenzata dal fatto che le connessioni sono dichiarate immediate (mid-frame) o (frame) ritardate.

- Le connessioni contribuiscono alla latenza della comunicazione. È determinata dai componenti (bus virtuale, componenti hardware) a cui è associata la connessione.

- Le partizioni, se presenti, contribuiscono alla latenza totale e alla latenza dell'output della partizione in termini di offset del frame principale.

Ogni colonna del report si riferisce ad una caratteristica del collaboratore di latenza:

1. Tipo e nome del **collaboratore**.

2. La **latenza minima specificata** come indicato dal valore della proprietà di latenza associato al collaboratore. La proprietà *latency* accetta un valore di intervallo e il limite inferiore viene utilizzato come minimo.

3. La **latenza minima attuale** minimo utilizzato dall'analisi. Questo valore è determinato dalle proprietà nella progettazione dell'architettura, ad esempio il periodo e il tempo di esecuzione del calcolo di un *thread* o il tempo di trasmissione da parte di un *bus*. Se non è possibile determinare alcun valore effettivo, viene utilizzato il valore specificato.

4. Il **minimum method** utilizzato per determinare il valore effettivo minimo. Le etichette sono:

    - **No latency**: utilizzare il valore zero in quanto nessun contributo di latenza è stato specificato o calcolato dal modello.

    - **Specified:** utilizzare il valore della proprietà di latenza specificato.

    - **Response time**: il tempo di risposta del componente, se specificato.

    - **Processing time**: il contributo di latenza relativo all'elaborazione di un componente in base al tempo di esecuzione del calcolo e alla scadenza.

    - **Transmission time**: contributo di latenza per una connessione. Utilizza l'associazione a un bus o bus virtuale utilizzando i valori delle proprietà di latenza di tali componenti o il calcolo del tempo di trasmissione basato sui valori delle proprietà delle dimensioni dei dati trasmessi se sono stati specificati *Transmission_Time* e *Data_Size*

    - **queued**: contributo di latenza dovuto all'accodamento. Utilizza le dimensioni della coda, il tempo di esecuzione del calcolo e i valori delle proprietà del periodo.

    - **sampling:** contributo di latenza dovuto periodicamente al campionamento di un componente. Utilizza i valori delle proprietà *period*. Questo valore può differire nelle impostazioni di sistema sincrone e asincrone.

    - **Delayed sampling**: contributo di latenza dovuto a una connessione ritardata. Utilizza i valori delle proprietà *period*.

    - **Sampling protocol/bus** dovuto al protocollo bus periodico. Utilizza i valori delle proprietà *period*.

    - **partition major frame**: Contributo di latenza dovuto all'esecuzione periodica di una partizione. Utilizza il frame principale della pianificazione O della proprietà *period* di ARINC653 del processore virtuale che rappresenta la partizione.

- **Partition output**: contributo di latenza dovuto al ritardo nell'output fino alla fine della partizione (PE) o al frame principale (MF). Utilizza i valori delle proprietà della pianificazione delle partizioni ARINC653.

- **Partition offset**: contributo di latenza dovuto all'offset della partizione nella pianificazione delle partizioni, dall'inizio del frame principale o della partizione precedente nella stessa pianificazione. Utilizza i valori delle proprietà della pianificazione delle partizioni ARINC653.

5. Le colonne **maximum specified**, **maximum actual** e **maximum method** descrivono il contributo massimo di latenza

6. Nella colonna **Commenti** vengono forniti ulteriori dettagli sui calcoli e possono essere visualizzati avvisi o messaggi di errore per segnalare incoerenze

Ogni voce del report di latenza end-to-end mostra i totali calcolati in una riga con etichetta **Latency Total**. Mostra il totale per i valori specificati e il totale per i valori effettivi. Nella riga con etichetta **Specified End To End Latency** il report mostra la latenza end-to-end prevista come specificato dal valore della proprietà di latenza associato al flusso end-to-end.
Segue un riepilogo dei risultati etichettati **End to end Latency Summary**. Nel riepilogo i totali calcolati dei contributi di latenza specificati ed effettivi per la latenza minima e massima vengono confrontati con la latenza end-to-end prevista ed è incluso un messaggio appropriato (*success*, *warming* o *error message*). Il riepilogo confronta anche la variazione di latenza effettiva tra il minimo e il massimo che supera la variazione nella variazione di latenza end-to-end prevista specificata dal valore dell'intervallo di latenza associato al flusso end-to-end.

```
bus UART
properties
      Latency => 1 Ms .. 1 Ms;
      SEI::BandWidthCapacity => 921600.0 bitsps;
      Transmission_Time => [ Fixed => 488us .. 490us;
                             PerByte => 10 us .. 11 us; ];

end UART;



thread Coning_Sculling_S1

      properties
            Dispatch_Protocol => Periodic;
            Period => 250us;
            compute_execution_time => 150us .. 300us;
--          reference_processor => classifier (::platform::ecu);
--          sei::instructionsperdispatch => 1.24 kipd .. 1.25 mipd;

end Coning_Sculling_S1;
```

Figura 7 Esempi di contributori con le relative proprietà che definiscono il contributo fornito

CAPITOLO 2: Analisi di sicurezza

2.0 Introduzione al capitolo

Quando vengono realizzati dei sistemi critici dal punto di vista della sicurezza, i progettisti devono dimostrare che il sistema è affidabile e resiliente ai diversi casi d'uso e ai guasti. A tal fine, sono necessari diversi tipi di analisi per dimostrare che il sistema è sicuro e dispone di misure appropriate per gestire i guasti critici.
AADL non supporta i costrutti semantici necessari per eseguire analisi di sicurezza. Per questo motivo, il linguaggio di base viene esteso con un altro dedicato alla sicurezza (chiamato Error Model Annex v2, EMV2) che fornisce la semantica necessaria per acquisire informazioni di sicurezza e mostrarne la propagazione attraverso l'architettura.

2.1 Introduzione all'EMV2

Questa estensione di AADL è fondamentale per acquisire le informazioni su cosa può andare storto all'interno dell'architettura. EMV2 estende il linguaggio per specificare quali errori possono derivare dai componenti, come tali errori si propagano attraverso l'architettura e in che modo potrebbero influire sul comportamento dei componenti.
EMV2 è in grado di sopportare la modellazione e l'analisi dei guasti dell'architettura sfruttando i modelli implementati in AADL con:

- Librerie di modelli di errore riutilizzabili.
- Annotazioni del comportamento degli specifici errori nelle componenti tramite clausole imposte sul sistema.

Dunque, il ruolo della libreria EMV2 è quello di fornire delle raccolte riutilizzabili sul tipo di errori e fornire dei set sui principali tipi di errore che si possono verificare. In più, una libreria sui modelli di errori può contenere una macchina che raffigura gli eventi, gli stati e le transizioni.

Esistono 4 livelli di astrazione:

- **Livello 0**: ontologia dell'errore. Questo elenco di tipi di errore distingue gli errori all'interno dell'architettura e il modo in cui ogni tipo, se propagato, può influire sul comportamento. I tipi di errore possono essere estesi.
- **Livello1**: *error flow* all'interno dell'architettura. Questo livello indica come gli errori attraversano l'architettura, definendo dove gli errori hanno origine (*error source*), dove si propagano (*error path*) e dove impattano (*error sink*) l'architettura. Le informazioni a questo livello forniscono informazioni sufficienti per generare un report sulla sicurezza, ad esempio il report **Fault Impact**, che mostrano come ogni errore originato da un componente (*error source*) alla fine influisce sull'architettura.
- **Livello 2**: comportamento in caso di errore del componente. Questo livello indica in che modo gli errori hanno un impatto sul comportamento di un componente.
- **Livello 3**: comportamento di errore composito. Lo stato corrente potrebbe dipendere non solo da eventi esterni, ma anche dai sottocomponenti. In un approccio di componente gerarchico come AADL, lo stato del componente può dipendere dallo stato del sottocomponente.

Livello 0: Tipi di errore e Ontologia degli errori

L'EMV2 include il concetto di tipo di errore. Un tipo di errore è identificato da un nome univoco che definisce l'errore che rappresenta. Ad esempio, il tipo di errore *ValueError* rappresenta tutti gli errori correlati al valore dei dati. Un tipo di errore può essere esteso da altri tipi, formando una gerarchia di tipi di errore. Ad esempio, il tipo *ValueErro*r può essere esteso a *OutOfRange* che rappresenta il caso in cui un valore non rientra nell'intervallo previsto. A sua volta, il tipo di errore *OutOfRange* può essere esteso a *BelowRange* o *AboveRange*, che specifica come il valore non rientra nell'intervallo. Allo stesso modo, un errore di concorrenza può essere esteso ad un errore mutex o ad una race condition. L'EMV2 definisce un insieme di tipi di errore riutilizzabili. I tipi di errore predefiniti rappresentano gli errori comuni durante la progettazione di un sistema. Tuttavia, se i nomi esistenti e l'ontologia degli errori non si adattano al progetto, è possibile definirne uno proprio. A tale fine, è necessario creare un pacchetto AADL con una clausola EMV2 contenente i tipi di errore e le estensioni (Appendice 2).

Livello 1: Flussi di errore all'interno dell'architettura

Una volta che abbiamo i tipi di errore per il nostro sistema, possiamo definire come gli errori si propagano attraverso l'architettura (figura 54, figura 55). L'EMV2 richiede di descrivere quanto segue:

- **Propagazioni degli errori**: definire quali errori provengono dall'interno e dall'esterno del componente. Le propagazioni degli errori specificano i tipi di errori previsti sulle interfacce dei componenti e associazioni (*processor*, *memory* o *bus*). Le propagazioni non definiscono il modo in cui si relazionano tra loro le componenti, questo viene definito nei flussi di errore.
  La propagazione dell'errore si identifica con:

  - Nome della funzionalità o dell'associazione in cui viene propagato l'errore.
  - una direzione (in uscita o in uscita).
  - Un tipo di errore impostato.

- **Flussi di errore**: definire il modo in cui le propagazioni degli errori in entrata e in uscita si relaziona con gli altri. Le propagazioni degli errori specificano i tipi di errore che i componenti devono inviare o ricevere. Ma non specifica come errori in entrata e in uscita si correlano tra loro: infatti, nessuno può sapere come un tipo di errore da una porta in ingresso possa generare un tipo di errore su una porta in uscita o come un di errore possa influire su un processo e causare una propagazione in uscita sulle sue porte dati. La relazione tra i tipi di errore è definita con i flussi di errore, che sono delle estensioni della propagazione di errori.
  Ci sono tre tipi di flussi di errore, che imitano i flussi standard fondamentali:

  - **error source**: l'errore ha origine da o all'interno del componente. Il componente ha un errore interno che alla fine genererà l'errore propagato.
  - **error path**: l'errore viene propagato dal componente, passandolo da un errore in ingresso ad un errore in uscita
  - **error sink**: l'errore viene gestito e/o mitigato all'interno di un componente.

  Una specifica del flusso di errore è composta dai seguenti elementi:

- Un identificatore univoco
- Tipo di flusso di errore (ad esempio *error source*, *error path* e *error sink*)
- Nome della propagazione dell'errore a cui si fa riferimento e dei tipi di errore associati. Per l'*error source* e l'*error sink*, è necessaria una sola propagazione degli errori. Per *l'error path* sono necessari due percorsi di propagazione. Uno per la propagazione dell'errore in ingresso e un altro per la propagazione dell'errore in uscita.

Livello 2: Comportamento in caso di errore del componente

Il livello successivo di astrazione è la **Error Behavior State Machine (EBSM)**. Consente di associare una macchina a stati specifica per la sicurezza che definisce con precisione lo stato del componente in base alle propagazioni di errori in ingresso e agli *error events* interni (Component Error Behavior Appendice 5). In EMV2, la specifica dell'EBSM è fatta in due fasi:

1. Innanzitutto, viene specificata una macchina a stati generici completamente indipendente dal componente che può essere associata a qualsiasi altro componente.
2. La finestra di progettazione perfeziona quindi la macchina a stati generica in modo che corrisponda alle specifiche del componente.

Una macchina generica a stati è definita da:

- Un identificatore univoco
- Zero per i diversi eventi di errore che rappresentano errori interni al sistema. Ogni evento viene identificato con un nome univoco all'interno del componente.
- Uno o più stati di errore con esattamente uno stato iniziale. Uno stato di errore è definito da un nome univoco all'interno della macchina a stati.
- Una o più transizioni di errore. Una transizione di errore è definita da un nome univoco, uno stato iniziale, una condizione e uno stato finale.

Livello 3: Comportamento di un errore composito

L'EBSM consente di specificare il comportamento di errore per un componente con i relativi input e output. Nonostante ciò, in questa macchina a stati, non possiamo rappresentare lo stato di un componente in base al suo sottocomponente. In alcuni casi, lo stato non dipende dall' input o output, ma piuttosto dallo stato dei sottocomponenti (Composite Error Behavior Appendice 5).
Si indica lo stato corrente in base al sottocomponente tramite un comportamento di errore composito, che può essere specificato solo nell'implementazione del componente in cui vengono specificati i sottocomponenti.

Un comportamento di errore composito richiede l'inclusione di una macchina a stati generica che definisce gli stati di errore del componente. È quindi necessario definire logicamente lo stato di errore del componente in base agli stati di errore dei relativi sottocomponenti.

Il comportamento di un errore composito è utile anche quando si definiscono gli stati di errore del componente radice che integra poi tutto il componente sistema.

2.2 Proprietà dell'errore

Per quanto riguarda il linguaggio di base, EMV2 definisce anche il proprio meccanismo di proprietà. L'obiettivo è annotare gli elementi EMV2 (propagazione degli errori, flusso, stati) con informazioni che possono essere utilizzate dagli strumenti di analisi del modello. Le proprietà EMV2 vengono dichiarate in una sezione delle proprietà nella sotto clausola EMV2.
Per l'analisi di sicurezza disponibile in OSATE vengono utilizzate le seguenti proprietà:

- **emv2: : occorrencedistribution**: indica il verificarsi di un evento di errore o di una propagazione di errori. Definisce la funzione di distribuzione dell'errore e il relativo valore associato.
- **emv2:: severity**: gravità di un evento di errore o propagazione di errori. È un numero da 1 (alto) a 5 (basso).
- **emv2:: likelihood**: la probabilità che si verifichi un evento di errore o una propagazione dell'errore. Il valore è un valore letterale che va da A (alto) a E (basso).
- **emv2::hazards**: specifica le informazioni aggiuntive associate a un evento di errore, stato di errore o propagazione di errore. La proprietà acquisisce tutti i dati utilizzati dall'analisi di sicurezza, che richiede l'estrazione di informazioni da un qualsiasi errore che si verifichi. La proprietà è un record che richiede dei campi quali titolo, descrizione dell'errore, effetto dell'errore, fasi di missione, metodo di verifica.

```
annex EMV2{**
    use types ErrorLibrary;
    use behavior ErrorLibrary::FailStop;

    error propagations
        Temp_Data: out propagation{OutOfRange,ItemOmission};
    flows
        ef0 : error source Temp_Data{OutOfRange,ItemOmission};
    end propagations;

    properties
        emv2::OccurrenceDistribution =>[ProbabilityValue => 0.01e-4;    Distribution => Poisson;] applies to Temp_Data.itemomission,Temp_Data.OutOfRange;
        emv2::Severity => ARP4761::Major applies to Temp_Data.itemomission,Temp_Data.OutOfRange;
        emv2::likelihood => ARP4761::Probable applies to Temp_Data.itemomission,Temp_Data.OutOfRange;
        emv2::hazards => ([crossreference => "T_AccY_1.00";
                           failure        => "ItemOmission";
                           phases         => ("all");
                           description    => "No data from the Temp_Sensor_Y";
                           comment        => "Reset the sistem if the accelerometer is not working as well";]) applies to Temp_Data.itemomission,Temp_Data.OutOfRange;
        emv2::hazards => ([crossreference => "T_AccY_1.00";
                           failure        => "OutOfRange";
                           phases         => ("all");
                           description    => "Data from the Y-Accelerometer Temp Sensor is Out of Range";
                           comment        => "Look at alimentation or restart the system and recalibrate sensor";]) applies to Temp_Data.OutOfRange;

    **};
```

Figura 8 Proprietà dell'errore

2.3 Strumenti per l'analisi

Di seguito vengono elencati alcuni Tool di analisi forniti da Osate2 e da AADL. I Tool usati in questa trattazione sono solo alcuni di quelli disponibili, ogni tool richiede che sul sistema vengano modellate le proprietà e le caratteristiche richieste per l'analisi che si vuole effettuare.
Viene creata un'apposita sezione (figura 8), in ogni elemento considerato, relativa, esclusivamente, agli errori e a tutti gli elementi che descrivono il comportamento del sistema in presenza di Fault.

Questa sezione viene inserita all'interno della *declaration type* e dell'*implementation* dei componenti presi in considerazione nell'analisi. Nei capitoli precedenti sono stati mostrati alcuni esempi dell'introduzione della libreria EMV2 nella modellazione del sistema IMU. Grazie alla grammatica fornita da questa libreria sono stati modellati i vari livelli di astrazione dell'errore per tutti i componenti presi in considerazione (vedere Capitoli 2 e 3 e Appendici 2 e 5). Sulle istanze del modello realizzato con l'ausilio di EMV2 si possono eseguire i tre Tool di analisi spiegati di seguito:

- **Functional Hazard Analysis**
- **Fault Tree Analysis**
- **Fault Impact Analysis**

2.4 Functional Hazard Assessment

La *Functional Hazard Assessment* (FHA) è un'analisi di sicurezza basata sullo standard ARP4761. La FHA è un'analisi completa di tutte le funzioni per identificare le condizioni eccezionali che possono innescare un errore. Lo standard ARP4761 fornisce un elenco di possibili gravità per classificare tali condizioni. Una condizione eccezionale, pericolo, viene rappresentata come un artefatto EMV2, ad esempio un *error event* o una **error propagation** in uscita specificata come *error source*.

Tale strumento analizza ogni componente e segnala le informazioni su tutti gli *error event* e *error source* su di esso.

Per eseguire questa analisi devono essere specificate le seguenti proprietà EMV2 per ogni elemento rilevante definito nella sezione sopra mostrata:

- emv2::occurrencedistribution
- emv2::severity
- emv2::likelihood
- emv2::hazards

```
emv2::OccurrenceDistribution => [ProbabilityValue => 0.2e-6; Distribution => Poisson;] applies to Temp_Data.itemomission;
emv2::OccurrenceDistribution => [ProbabilityValue => 0.01e-4; Distribution => Poisson;] applies to Temp_Data.OutOfRange;
emv2::Severity => ARP4761::Major applies to Temp_Data.itemomission, Temp_Data.OutOfRange;
emv2::likelihood => ARP4761::Probable applies to Temp_Data.itemomission, Temp_Data.OutOfRange;
emv2::hazards => ([crossreference => "T_AccX_1.00 v2.34";
        failure => "ItemOmission";
        phases => ("all");
        description => "No data from the Temp_Sensor_X";
        comment => "Reset the sistem if the Temp_Sensor is not working as well";]) applies to Temp_Data.itemomission;
emv2::hazards => ([crossreference => "T_AccX_1.00 v3.34";
                failure         => "OutOfRange";
                phases          => ("all");
                description     => "Data from the X-Accelerometer Temp Sensor is Out of Range";
                comment         => "Look at alimentation or restart the system and recalibrate sensor";]) applies to Temp_Data.OutOfRange;
```

Figura 9 Esempio di proprietà per la FHA

Una volta eseguita l'analisi, gli strumenti di analisi generano un foglio di calcolo che contiene l'elenco di tutti i pericoli con le informazioni raccolte dai modelli.

2.5 Fault Tree Analysis

L'analisi dell'albero dei guasti (FTA, Fault Tree Analysis), mostra la gerarchia degli errori e l'origine dell'errore correlata a un particolare stato. È più complessa perché si basa su più elementi di modellazione rispetto all'FHA. L'FHA elenca tutti i potenziali errori per il sistema, mentre l'FTA mostra

le dipendenze tra eventi di errore e propagazione degli errori correlati a una condizione speciale. Per questo motivo, FTA è un metodo migliore per analizzare un particolare evento di errore.

L'FTA è costituito dai seguenti elementi:

- **error events**: evidenzia uno stato del sistema o l'avvenimento di una particolare condizione. C'è almeno un evento nell'FTA: l'evento radice.
- **gate**: quando l'evento di errore non è una foglia, il gate definisce la condizione che genera un evento. La condizione è una condizione binaria come OR, AND o XOR.

Quando si generano alberi di guasto da modelli AADL, sono rilevanti i seguenti elementi EMV2:

- **error propagation** e **error flow**: lo strumento di analisi FTA cercherà le fonti di errore e gli eventi di errore che hanno influenzato il componente in analisi.
- **error behavior state machine**: l'EBSM di tutti i componenti collegati al componente in fase di analisi. Lo strumento rileva gli *error events* o le *error propagation* che possono influire sul componente in analisi.
- **composite error behavior**: la macchina a stati di un errore composito definisce lo stato errore di un componente in base agli stati di errore dei relativi sottocomponenti. Questa definizione è un elemento essenziale dell'FTA e l'istanza di sistema deve disporre di una macchina a stati di errore composito che avvii l'FTA. L'analisi inizia con lo stato finale dell'istanza di sistema ed elabora lo stato dei sottocomponenti a cui viene fatto riferimento nel comportamento di errore composito.
- **emv2::occurrencedistribution** : la proprietà *occurrencedistribution* viene utilizzata per completare il diagramma FTA con le probabilità appropriate. Lo strumento FTA può quindi utilizzare questi valori per calcolare la probabilità di un guasto o mostrare la probabilità di ogni cutset.

La *component error behavior* viene definita nella sua Type, è relativa solo a quello specifico componente e identifica la causa (*error event*) per la quale il componente passa da uno stato all'altro.

```
component error behavior
    events
        Reset: recover event;
    transitions
        t0: Operational-[accel_failure]-> FailStop;
        t01: Operational-[Power_5V{LOW_Power}]-> NCritF_5V;
        t02: Operational-[Power_5V{NoPower}]-> CritF_5V;
        t03: NCritF_5V-[Reset]-> Operational;
        t04: CritF_5V-[Broken]-> FailStop;
        t05: CritF_5V-[Reset]-> Operational;
        t1: FailStop -[Reset]-> Operational;
        t2: Operational-[Critical_Failure]-> CriticalModeFailure;
        t3: CriticalModeFailure-[Stop]-> FailStop;
end component;
```

Figura 10 Esempio della component error behavior dei sensore di temperatura

La *composite error behavior* è relativa ai sistemi che contengono dei sottocomponenti e va inserita nella sezione errori all'interno della *implementation* del sistema. Nella Appendice 5 sono mostrate le component e le composite error behavior dei vari elementi modellati.

2.6 Fault impact

La **Fault impact analysis** mostra l'impatto di ogni errore all'interno del sistema. In AADL l'analisi segnala ogni flusso di errore e mostra ogni elemento del flusso (*error source*, *error path* e *error sink*). Inoltre, segnala qualsiasi errore che andrà poi a propagarsi su altri componenti del sistema. In questo caso, l'evento errore deve attivare un cambiamento di stato quando il componente propaga un errore specifico. Se una singola origine di errore o evento di errore è connesso a più flussi e alla fine raggiunge diversi *error sink*, l'analisi degli errori deve riportare ognuno di questi errori. In altre parole, per utilizzare l'analisi dell'impatto dei guasti sul modello AADL, è necessario dichiarare i flussi di errore con l'origine, il percorso e la fine dell'errore.

La **fault impact analysis** inizia da ciascun *error source* e dalle condizioni opzionali ad esse associate. L'elemento associato alla condizione, nel momento in cui si verifica un errore, diventa l'origine del guasto e le propagazioni in uscita da tale elemento vanno a identificare gli errori di primo livello. Se il source iniziale ha più tipi di errore che si verificano contemporaneamente, si viene a creare un flow per ogni tipo di errore.

Per ogni componente, le propagazioni in uscita seguono le connessioni o le associazioni dichiarate in fase di modellazione, seguendo tutto il flow, fino ad arrivare all'elemento di sink, cioè una propagazione dell'errore esterna al sistema o l'entrata all'interno di un ciclo di errori. Per ogni propagazione di errore in entrata ad un elemento, vengono anche identificate le differenti propagazioni in uscita. Se non vengono specificati percorsi di errore per un evento in entrata, si presume che tutte le propagazioni in uscita ne siano influenzate.

Nel momento in cui il flow dell'errore termina, un'etichetta indica il motivo della fine. Le etichette sono:

- **Masked**: indicazione che è stato raggiunto il sink dell'errore.
- **Unhandled Type**: quando non c'è consistenza nella propagazione dell'errore: il tipo di errore propagato in uscita non è elencato come tipo di errore in ingresso nella componente successiva.
- **Unhandled Failure Effect**: quando il tipo di errore propagato in ingresso non è elencato in alcun percorso di errore o propagazione di errore in uscita
- **External Effect**: impatto sul Sistema. Ovvero il flusso ha raggiunto una propagazione in uscita dal sistema di primo livello.
- **Propagation Cycle:** il flusso raggiunge un elemento che è stato già raggiunto in precedenza propagando lo stesso tipo di errore in uscita.
- **No feature with out propagation**: una propagazione in uscita ha una connessione in uscita a una funzione del sistema di primo livello, dove quella funzione non ha una propagazione degli errori dichiarata. La propagazione in uscita dell'origine della connessione rappresenta l'effetto esterno.
- **No Outgoing Connection**: vi è una propagazione in uscita su una caratteristica che non è collegata a un altro componente.

- **No In Propagation**: la destinazione di una connessione, non dispone di tale connesione in ingresso
- **No Binding**: la propagazione in uscita da una componente, fa parte di una associazione che non è stata ancora dichiarata completamente.

Etichette aggiuntive possono essere utilizzate per gli elementi intermedi della traccia di impatto per indicare quando un tipo in entrata o un insieme di tipi viene propagato come sottotipi separati (Subtype), se le dichiarazioni di flusso nel modello principale vengono utilizzate quando mancano flussi di errore (flowpath) o quando una propagazione in entrata viene mappata a tutte le propagazioni in uscita (All out props).

2.7 Riepilogo degli elementi utili per l'analisi

Nella tabella seguente sono elencati tutti gli elementi di modellazione utilizzati per ogni analisi. L'obiettivo è duplice: avere una piccola scheda per gli utenti AADL, ma anche mostrare che un approccio basato su modelli può semplificare l'analisi del sistema rendendo i risultati più coerenti. Innanzitutto, poiché le stesse informazioni vengono utilizzate tra le analisi, un approccio basato su modelli evita la necessità di immettere le informazioni due volte, come nel caso delle analisi manuali tipiche. In secondo luogo, poiché tutte le analisi riutilizzano automaticamente le stesse informazioni, tutte le informazioni provenienti dai documenti generati saranno disponibili, evitando potenziali errori umani che si verificano con documenti prodotti manualmente.

Analysis Method	Relevant EMV2 elements
Functional Hazard Assessment	• *error source* • *error events* • *emv2::occurrencedistribution* • *emv2::severity* • *emv2::likelihood* • *emv2::hazards*
Fault Tree Analysis	• Error flows: *error source, error path* and *error sink* • *error behavior state machine* • *error events* • *composite error behavior* • *emv2::occurrencedistribution*
Failure Mode and Effects Analysis	• Error flows: *error source, error path* and *error sink* • *error behavior state machine* • *error events*

Figura 11 Elementi della libreria EMV2 rilevanti per le differenti analisi

CAPITOLO 3: Risultati e Conclusioni

3.0 Istanza del sistema

Una volta che il sistema considerato è stato modellato con tutti gli elementi necessari alle analisi, si può creare un'istanza del sistema, seguendo questi step:

1. Selezionare una system implementation dalla visuale Outline di Osate
2. Selezionare l'opzione **Instatiate System**.

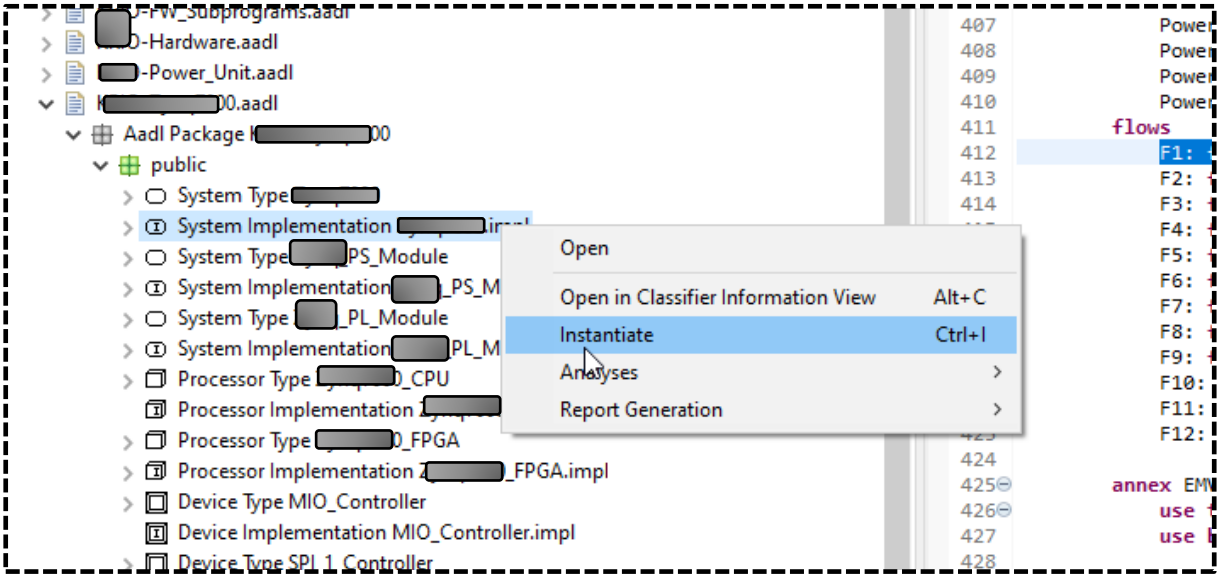


Figura 12 Creazione dell'istanza della parte di sistema scelta

Verrà creata una cartella *instances* al cui interno si troveranno tutte le istanze create dai componenti del sistema. Per eseguire l'analisi di latenza è stata creata l'istanza del processore, poiché il percorso end-to-end su cui si vuole eseguire l'analisi è al suo interno, mentre per le analisi di safety viene creata l'istanza del sistema completo.

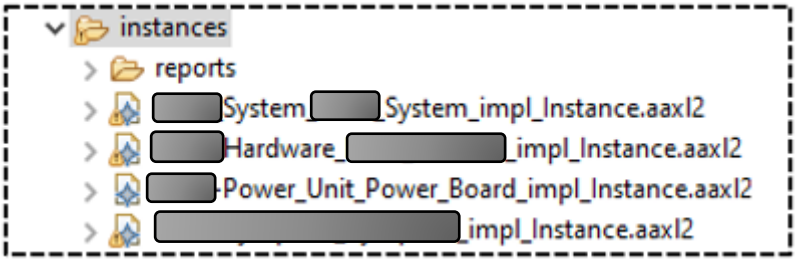


Figura 13 Istanze del sistema

Tramite il menu *Analyses* possiamo ora effettuare l'analisi sull'istanza creata scegliendo tra le

possibili analisi.
Le categorie di Analisi sono quattro:

- **Budget**
- **Semantic Checks**
- **Timing**
- **Safety**

Le analisi studiate in questa tesi sono:

- *Check Flow Latency (TIMING)*
- *Fault Tree Analisys (SAFETY)*
- *Fault Impact Analisys (SAFETY)*
- *Functional Hazards Assessment (SAFETY)*

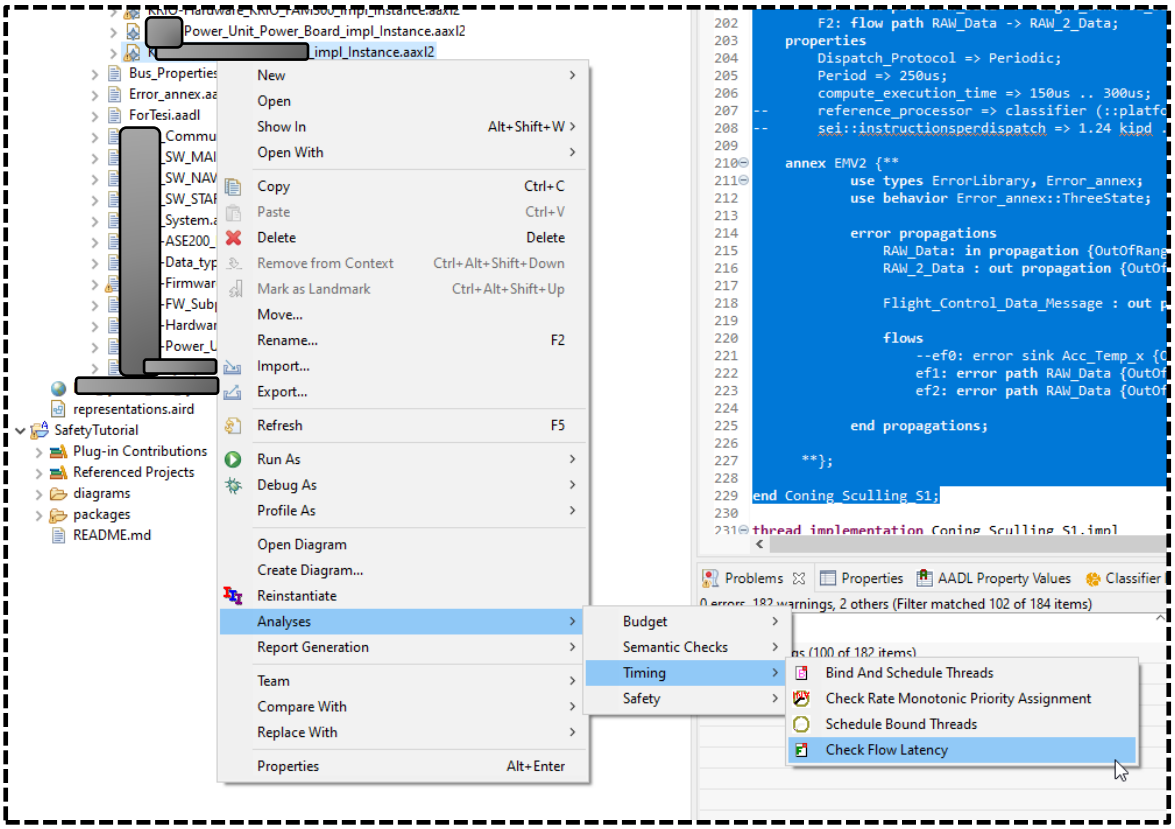


Figura 13 Menu dei tools di analisi forniti da OSATE

Verranno ora mostrati i risultati ottenuti dalle varie analisi applicate alle Istanze del nostro sistema, create dopo l'opportuna modellazione di tutti i componenti HW, SW e COMPOSITE.
All'interno della cartella *istances* viene creata una cartella *reports* non appena si esegue la prima analisi, al cui interno vengono create le cartelle contenenti i risultati suddivisi per tipologia.

3.1 Flow Latency Analysis

Viene analizzato un preciso end-to-end flow, quello rappresentativo del percorso dalla nascita del dato dall'accelerometro X a quando, dopo Firmware-Software-Firmware, il dato viene restituito su una porta SDLC all'utente.
Gli elementi fondamentali di questo percorso sono:

- i tempi di lettura del Firmware
- la velocità del bus (AXI BUS)
- i tempi di elaborazione delle varie parti del Software
- Le caratteristiche temporali del sensore
- Le connessioni tra le componenti e i bus a cui sono associate

I documenti del sistema reale affermano che il flusso di dati studiato ha una latenza massima di 3ms e che in media si hanno 1,5ms. L'analisi di latenza eseguita sul sistema modellato ha generato i report mostrati in figura 15 (quelli evidenziati si riferiscono al caso di studio).

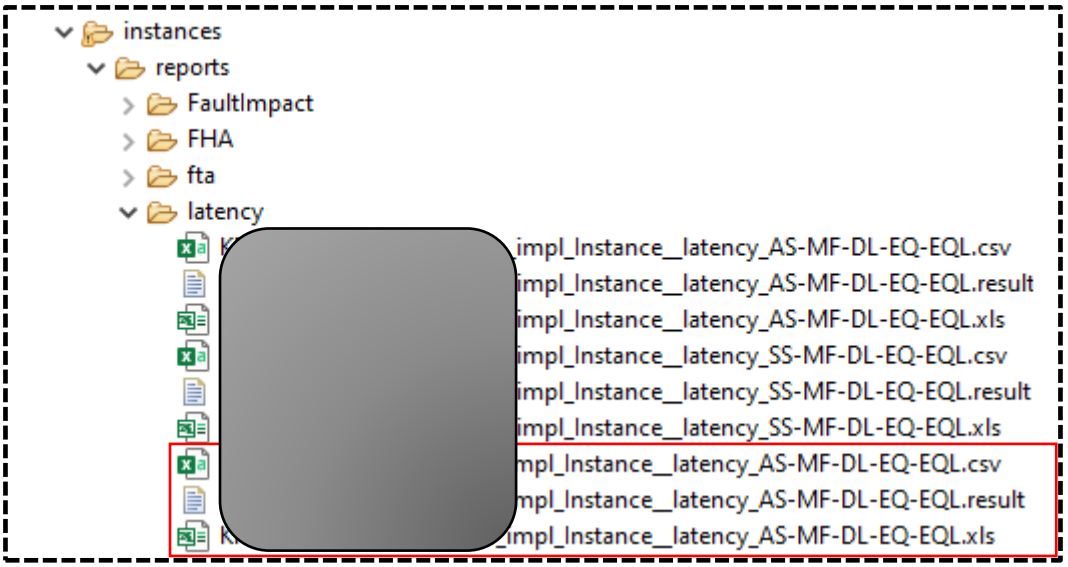
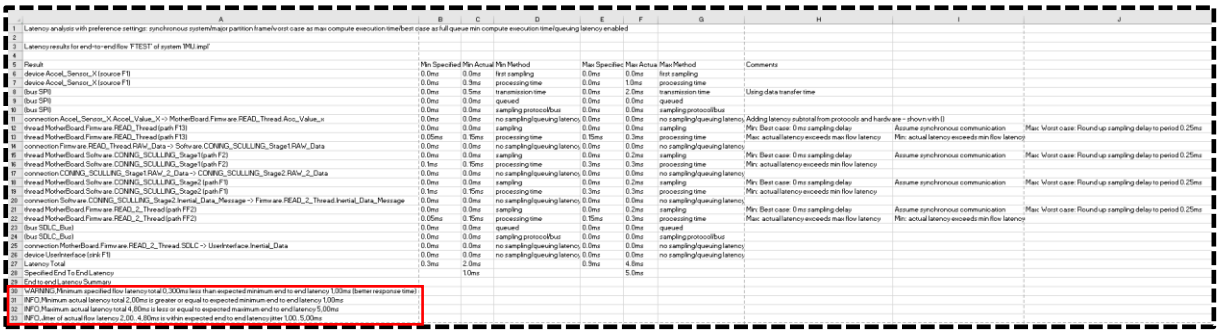


Figura 14 Elenco dei reports analisi di latenza suddivisi per istanze

All'interno del file .csv troviamo riportati tutti gli elementi di interesse per la latenza del caso studiato. Consiste in sezioni separate per ogni end-to-end flow e per ogni operational mode se aggiunti al modello. L'intestazione di ogni report identifica il nome dell'end-to-end flow analizzato, la component implementation che contiene la dichiarazione dell'end-to-end flow, le operational mode che sono state analizzate (se sono state dichiarate nelle modalità di lavoro), e le preference settings utilizzate. Nella figura 16 viene mostrato il report completo con particolare attenzione al riquadro evidenziato (figura 17) che rappresenta il sommario dei risultati ottenuti dall'analisi svolta nell'end-to-end flow considerato.



	A	B	C	D	E	F	G	H	I	J
1	Latency analysis with preference settings: synchronous system/major partition frame/worst case as max compute execution time/best case as full queue min compute execution time/queuing latency enabled									
2										
3	Latency results for end-to-end flow 'FTEST' of system 'IMU.impl'									
4										
5	Result	Min Specified	Min Actual	Min Method	Max Specified	Max Actual	Max Method	Comments		
6	device.Accel_Sensor_X (source F1)	0.0ms	0.0ms	first sampling	0.0ms	0.0ms	first sampling			
7	device.Accel_Sensor_X (source F1)	0.0ms	0.3ms	processing time	0.0ms	1.0ms	processing time			
8	(bus SPI)	0.0ms	0.5ms	transmission time	0.0ms	2.0ms	transmission time	Using data transfer time		
9	(bus SPI)	0.0ms	0.0ms	queued	0.0ms	0.0ms	queued			
10	(bus SPI)	0.0ms	0.0ms	sampling.protocolBus	0.0ms	0.0ms	sampling.protocolBus			
11	connection:Accel_Sensor_X.Accel_Value_X -> MotherBoard.Firmware.READ_Thread.Acc_Value_u	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency	Adding latency subtotal from protocols and hardware - shown with ()		
12	thread MotherBoard.Firmware.READ_Thread (path F13)	0.0ms	0.0ms	sampling	0.0ms	0.0ms	sampling	Min: Best case: 0 ms sampling delay	Assume synchronous communication	Max: Worst case: Round up sampling delay to period 0.25ms
13	thread MotherBoard.Firmware.READ_Thread (path F13)	0.05ms	0.15ms	processing time	0.15ms	0.3ms	processing time	Max: actual latency exceeds max flow latency	Min: actual latency exceeds min flow latency	
14	connection:Firmware.READ_Thread.RAW_Data -> Software.CONING_SCULLING_Stage1.RAW_Data	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency			
15	thread MotherBoard.Software.CONING_SCULLING_Stage1 (path F2)	0.0ms	0.0ms	sampling	0.0ms	0.2ms	sampling	Min: Best case: 0 ms sampling delay	Assume synchronous communication	Max: Worst case: Round up sampling delay to period 0.25ms
16	thread MotherBoard.Software.CONING_SCULLING_Stage1 (path F2)	0.1ms	0.15ms	processing time	0.3ms	0.3ms	processing time	Min: actual latency exceeds min flow latency		
17	connection:CONING_SCULLING_Stage1.RAW_2_Data -> CONING_SCULLING_Stage2.RAW_2_Data	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency			
18	thread MotherBoard.Software.CONING_SCULLING_Stage2 (path F1)	0.0ms	0.0ms	sampling	0.0ms	0.2ms	sampling	Min: Best case: 0 ms sampling delay	Assume synchronous communication	Max: Worst case: Round up sampling delay to period 0.25ms
19	thread MotherBoard.Software.CONING_SCULLING_Stage2 (path F1)	0.1ms	0.15ms	processing time	0.3ms	0.3ms	processing time	Min: actual latency exceeds min flow latency		
20	connection:Software.CONING_SCULLING_Stage2.Inertial_Data_Message -> Firmware.READ_2_Thread.Inertial_Data_Message	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency			
21	thread MotherBoard.Firmware.READ_2_Thread (path FF2)	0.0ms	0.0ms	sampling	0.0ms	0.2ms	sampling	Min: Best case: 0 ms sampling delay	Assume synchronous communication	Max: Worst case: Round up sampling delay to period 0.25ms
22	thread MotherBoard.Firmware.READ_2_Thread (path FF2)	0.05ms	0.15ms	processing time	0.15ms	0.3ms	processing time	Max: actual latency exceeds max flow latency	Min: actual latency exceeds min flow latency	
23	(bus SDLC_Bus)	0.0ms	0.0ms	queued	0.0ms	0.0ms	queued			
24	(bus SDLC_Bus)	0.0ms	0.0ms	sampling.protocolBus	0.0ms	0.0ms	sampling.protocolBus			
25	connection:MotherBoard.Firmware.READ_2_Thread.SDLC -> UserInterface.Inertial_Data	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency			
26	device.UserInterface (sink F1)	0.0ms	0.0ms	no sampling/queuing latency	0.0ms	0.0ms	no sampling/queuing latency			
27	Latency Total	0.3ms	2.0ms		0.9ms	4.8ms				
28	Specified End To End Latency		1.0ms			5.0ms				
29	End to end Latency Summary									
30	WARNING,Minimum specified flow latency total 0,300ms less than expected minimum end to end latency 1,00ms (better response time)									
31	INFO,Minimum actual latency total 2,00ms is greater or equal to expected minimum end to end latency 1,00ms									
32	INFO,Maximum actual latency total 4,80ms is less or equal to expected maximum end to end latency 5,00ms									
33	INFO,Jitter of actual flow latency 2,00..4,80ms is within expected end to end latency jitter 1,00..5,00ms									

Figura 15 Report completo analisi di latenza

L'intervallo di latenza voluto, impostato sul modello, va da un minimo atteso di 1 ms ad un massimo di 5 ms (figura 40).
Come mostrato nelle figure 77 e 78 l'analisi di latenza non ha restituito gli stessi risultati ottenuti sul sistema reale. La differenza che c'è tra i risultati del sistema reale e i risultati di OSATE è dovuta alla semplificazione del modello rispetto alla realtà. In ogni caso i risultati non si distaccano troppo da quelli reali. Questo dimostra che AADL rimane uno strumento efficace ed efficiente anche se il sistema non è modellato a bassissimo livello. Infatti, pur essendo rimasti su una modellazione ad alto livello, i risultati forniti sono comunque coerenti e accettabili.

29	End to end Latency Summary
30	WARNING,Minimum specified flow latency total 0,300ms less than expected minimum end to end latency 1,00ms (better response time)
31	INFO,Minimum actual latency total 2,00ms is greater or equal to expected minimum end to end latency 1,00ms
32	INFO,Maximum actual latency total 4,80ms is less or equal to expected maximum end to end latency 5,00ms
33	INFO,Jitter of actual flow latency 2,00..4,80ms is within expected end to end latency jitter 1,00..5,00ms

Figura 16 Ingrandimento dell'end-to-end latency summary, riquadro in figura 76

Possiamo notare che il report avvisa con un **warning** quando un *Minimo* o un *Massimo* non rispetta l'intervallo scelto, fornisce le informazioni sul *Minimo* e *Massimo* attuali e sul *Jitter* della latenza attuale del flow. Si nota che il sistema un minimo maggiore di 1 ms e un massimo minore di 0.20 ms rispetto ai limiti impostati.
La figura 18 rappresenta il sommario di figura 17 fornito però da OSATE, esternamente al report, nella *problem view*.

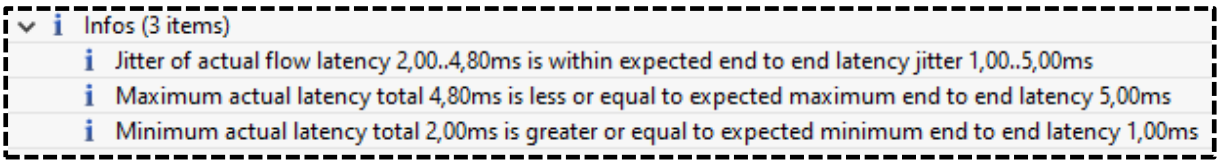


Figura 17 Problems view

3.2 Fault Tree Analysis

Questa tipologia di analisi è più complessa rispetto alle altre perché richiede un numero maggiore di elementi da modellare, poiché mostra le gerarchie degli *event error* ed *error sources* in relazione ad un particolare *error state*.

La FTA mostra le dipendenze tra *error events* and *error popagation* relativamente ad una particolare condizione, e risulta essere il miglior metodo per analizzare un particolare *error state*.

La costruzione dell'albero è relativa allo stato di **Failstop**.

In una prima analisi, l'albero risultante (figura 19) mancava di alcuni elementi, il modello non era del tutto completo e descritto. Non tutti i sottocomponenti presentavano le appropriate Error Behavior, ma nonostante questo il risultato ottenuto è abbastanza realistico e coerente con l'idea progettuale.

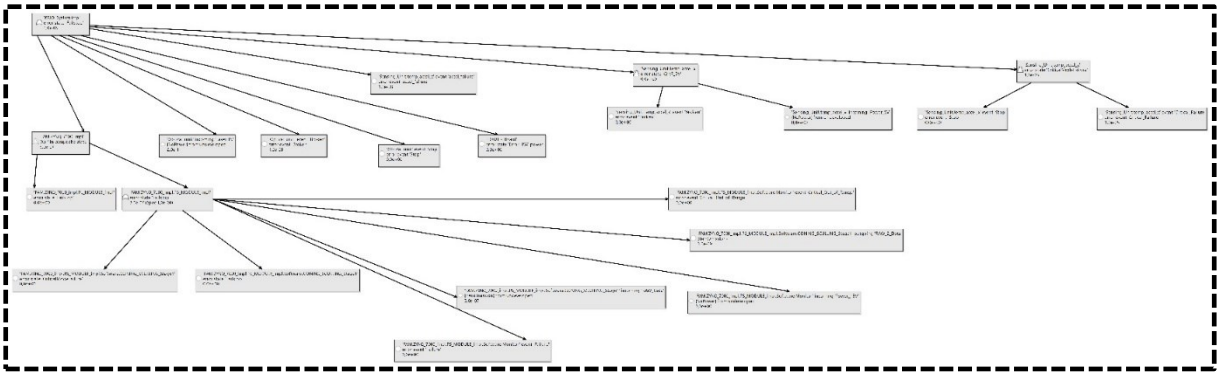


Figura 18 Primo Fault Tree creato

Successivamente sono stati modellati gli elementi mancanti e resi più precisi i modelli già presenti. L'albero ottenuto (figura 20) dopo una seconda lavorazione risulta essere più raffinato ed esplicativo, su ogni singola foglia è segnata la probabilità con cui il componente può ritrovarsi nello stato considerato.

La radice dell'albero rappresenta lo stato di **FailStop** del sistema totale IMU mentre tutte le singole foglie rappresentano lo stato di **FailStop** dei vari sottocomponenti. I rami descrivono come a causa di uno stato di **FailStop** in un componente figlio finisca nello stesso stato anche il componente padre.

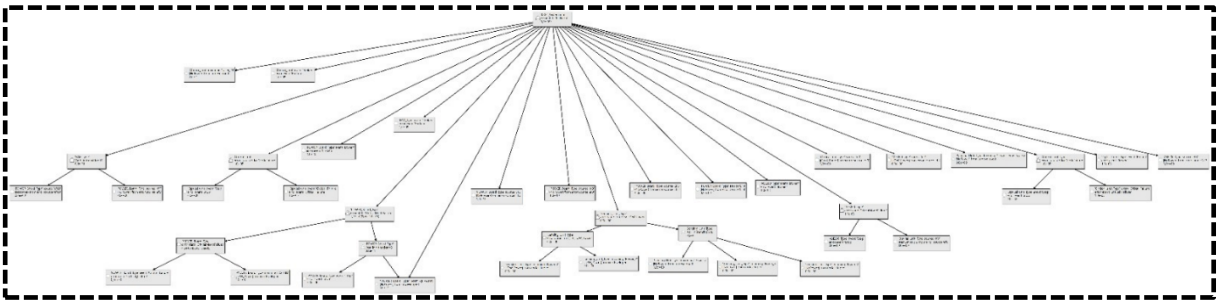


Figura 19 Fault Tree finale del sistema

La creazione di un albero degli errori partendo dal design e dalla documentazione è necessaria per certificare i sistemi come safety-critical nei campi avionico e aerospaziale, ma non solo, e AADL fornisce uno strumento efficace e affidabile per la creazione automatica degli alberi una volta modellato il sistema opportunamente

3.3 Functional Hazard Assessment

Il report dell'FHA (figura 21) è un file .csv che contiene l'elenco dei componenti con il loro nome, il modello d'errore e la porta su cui si verifica, una descrizione dell'errore, la *severity,* la tipologia di *faiulure*, e tutte le altre proprietà e caratteristiche attribuibili all'errore nelle apposite sezioni nelle definizioni dei componenti.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	C	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1	Component	Error Model Element	H	Description	Crossreference	Failure	Fa	Oper	El	R	M	F	Severity	Likelihood	I	Ti	Ta	I	Va	S	Comment						
2	Sensing_Unit	"ItemOmission on Accel_Temp_x"		"No data from the Temp_Sensor_X"	"T_AccX_1.00 v2.34"	"ItemOmission"		"all"					3	B							"Reset the sistem if the Temp_Sensor is not working as well"						
3	Sensing_Unit	"OutOfRange on Accel_Temp_x"		"Data from the X-Accelerometer Temp Sensor is Out of Range"	"T_AccX_1.00 v3.34"	"OutOfRange"		"all"					3	B							"Reset the sistem if the accelerometer is not working as well"						
4	Sensing_Unit/accel_x	"ItemOmission on acc_out_data"		"No data from the X-Accelerometer"	"Accel_X_1.1 "	"ItemOmission"		"all"					3	B							"Restart the sistem if the accelerometer is not working as well"						
5	Sensing_Unit/accel_x	"OutOfRange on acc_out_data"		"Data from the X-Accelerometer is Out of Range"	"Accel_X_1.2"	"OutOfRange"		"all"					3	B							"Look at alimentation or restart the system with a new sensor calibration"						
6	Sensing_Unit/temp_accel_x	"accel_failure on accel_failure"		"No data from the Temp_Sensor_1"	"F_SNSX 1.0.0"	"break"		"all"					3	B							"Reset the sistem if the Temp_Sensor is not working as well. IF necessary re						
7	Sensing_Unit/temp_accel_x	"break on accel_failure"		"No data from the Temp_Sensor_1"	"F_SNSX 1.0.0"	"break"		"all"					3	B							"Reset the sistem if the Temp_Sensor is not working as well. IF necessary re						
8	Sensing_Unit/temp_accel_x	"ItemOmission on Temp_Data"		"No data from the Temp_Sensor_X"	"T_AccX_1.00 v2.34"	"ItemOmission"		"all"					3	B							"Reset the sistem if the Temp_Sensor is not working as well"						
9	Sensing_Unit/temp_accel_x	"OutOfRange on Temp_Data"		"Data from the X-Accelerometer Temp Sensor is Out of Range"	"T_AccX_1.00 v3.34"	"OutOfRange"		"all"					3	B							"Look at alimentation or restart the system and recalibrate sensor"						
10	Sensing_Unit/temp_accel_y	"ItemOmission on Temp_Data"		"No data from the Temp_Sensor_Y"	"T_AccY_1.00"	"ItemOmission"		"all"					3	B							"Reset the sistem if the accelerometer is not working as well"						
11	Sensing_Unit/temp_accel_y	"OutOfRange on Temp_Data"		"No data from the Temp_Sensor_Y"	"T_AccY_1.00"	"ItemOmission"		"all"					3	B							"Reset the sistem if the accelerometer is not working as well"						
12	Sensing_Unit/temp_accel_y	"OutOfRange on Temp_Data"		"Data from the Y-Accelerometer Temp Sensor is Out of Range"	"T_AccY_1.00"	"OutOfRange"		"all"					3	B							"Look at alimentation or restart the system and recalibrate sensor"						
13	Sensing_Unit/temp_accel_z	"ItemOmission on Temp_Data"		"No data from the Temp_Sensor_Y"	"T_AccZ_1.00"	"ItemOmission"		"all"					3	B							"Reset the sistem if the accelerometer is not working as well"						
14	Sensing_Unit/temp_accel_z	"OutOfRange on Temp_Data"		"Data from the Y-Accelerometer Temp Sensor is Out of Range"	"T_AccZ_1.00"	"OutOfRange"		"all"					3	B							"Look at alimentation or restart the system and recalibrate sensor"						
15	POWER_Board	"NoPower on Power_15V"		"Alimentation for 15V device not working"	"F_PWR 15V 1.00"	"15V_OFF"		"all"					3	B							"Check the Power Unit and control the main power supply"						
16	POWER_Board	"LOW_Power on Power_15V"		"The 15V Power supply is LOW																							
17	Sensing_Unit_Type	"ItemOmission on Accel_Temp_x"		"No data from the Temp_Sensor_X"	"T_AccX_1.00 v2.34"	"ItemOmission"		"all"					3	B							"Reset the sistem if the Temp_Sensor is not working as well"						
18	Sensing_Unit_Type	"OutOfRange on Accel_Temp_x"		"Data from the X-Accelerometer Temp Sensor is Out of Range"	"T_AccX_1.00 v3.34"	"OutOfRange"		"all"					3	B							"Reset the sistem if the accelerometer is not working as well"						
19	POWER_Board_Type	"NoPower on Power_15V"		"Alimentation for 15V device not working"	"F_PWR 15V 1.00"	"15V_OFF"		"all"					3	B							"Check the Power Unit and control the main power supply"						
20	POWER_Board_Type	"LOW_Power on Power_15V"		"The 15V Power supply is LOW																							

Figura 20 Report FHA

Per l'analisi, si è posta l'attenzione sugli errori che derivano dai dati provenienti dal sensore di temperatura e dalla alimentazione a 15 V. Sul dato fornito dal sensore di temperatura sono stati modellati due possibili errori: *ItemOmission* e *OutOfRange (figura 22)*. Mentre sul dato fornito dal sistema di alimentazione a 15 V gli errori modellati sono: *NoPower* e *LOW_Power (figura 23).*

Sensing_Unit/temp_accel_x	"ItemOmission on Temp_Data"	"No data from the Temp_Sensor_X"	"T_AccX_1.00 v2.34"	"ItemOmission"
Sensing_Unit/temp_accel_x	"OutOfRange on Temp_Data"	"Data from the X-Accelerometer Temp Sensor is Out of Range"	"T_AccX_1.00 v3.34"	"OutOfRange"

Figura 21 Report FHA riferito al sensore di temperatura dell'accelerometro X

POWER_Board_Type	"NoPower on Power_15V"	"Alimentation for 15V device not working"	"F_PWR 15V 1.00"	"15V_OFF"
POWER_Board_Type	"LOW_Power on Power_15V"	"The 15V Power supply is LOW		

Figura 22 Report FHA riferito all'alimentazione a 15V proveniente dalla Power Board

Grazie a questo tool OSATE fornisce una lista dettagliata ed esaustiva (in base a quanto è dettagliato ed esaustivo il modello) di tutti gli errori che si possono verificare all'interno del sistema. Utile come elemento di partenza per tutte le altre analisi di safety.

3.4 Fault Impact Analysis

Il report della Fault Impact elenca le tracce di tutti gli *error* sources dichiarati all'interno dell'istanza del sistema e le tracce dell'impatto degli *error sources* esterni identificati come *error propagation* in entrata. Ciascuno elemento rappresenta una traccia dell'impatto della propagazione dell'errore:

- Il primo elemento di una riga identifica il componente che ha dato origine all'errore o il primo componente influenzato nel caso in cui l'errore ha avuto origine all'esterno (figura 24 – colonna A).
- Il secondo elemento identifica la modalità dell'errore iniziale, ovvero il tipo di errore (se compare "{ }") o lo stato dell'errore (se non compare "{ }") come specificato nella condizione (figura 24 – colonna A).
- Il terzo elemento descrive l'effetto del primo livello identificando la propagazione dell'errore in uscita, il tipo di errore propagato, il componente di destinazione e le propagazioni in entrata che saranno interessate (figura 24 – colonna C).

Le successive coppie di elementi (figure 24, 25, 66 – colonne "D - E", "F - G", "H - I") e rappresentano la modalità di guasto e il successivo livello di effetto, quando l'impatto viene tracciato lungo percorsi di propagazione dell'errore determinati da connessioni, associazioni o percorsi di propagazione dichiarati dall'utente nelle sottoclausole.

A	B	C	D
Fault Impact of System Internal Error Sources			
Component	Initial Failure Mode	1st Level Effect	Failure Mode
Sensing_Unit_Type	{OutOfRange}	{OutOfRange} Accel_Temp_x -> FAM_Type:Accel_Temp_x	FAM_Type {OutOfRange}
Sensing_Unit_Type	{ItemOmission}	{ItemOmission} Accel_Temp_x -> FAM_Type:Accel_Temp_x	FAM_Type {ItemOmission}
POWER_Board_Type	{HardwareFailure}	{NoPower} Power_Good_5V -> FAM_Type:Power_Good_5V	FAM_Type {NoPower}
POWER_Board_Type	{Power_Issue}	{LOW_Power} Power_Good_5V -> FAM_Type:Power_Good_5V	FAM_Type {LOW_Power}

Figura 23 Report Fault Impact- Parte 1

E	F	G
second Level Effect	Failure Mode	third Level Effect
{OutOfRange} FAM_Type.SDLC_Data -> MAIN_CONNECTOR.SDLC_Data[No In Propagation]		
{ItemOmission} FAM_Type.SDLC_Data -> MAIN_CONNECTOR.SDLC_Data[No In Propagation]		
{NoPower} Power_Temp_Acc_5V -> Sensing_Unit_Type:Power_Temp_Acc_5V	Sensing_Unit_Type {NoPower}	{ItemOmission} Accel_Temp_x -> FAM_Type:Accel_Temp_x
{LOW_Power} Power_Temp_Acc_5V -> Sensing_Unit_Type:Power_Temp_Acc_5V	Sensing_Unit_Type {LOW_Power}	{OutOfRange} Accel_Temp_x -> FAM_Type:Accel_Temp_x

Figura 24 Report Fault Impact- Parte 2

H	I
Failure Mode	4th Level Effect
FAM_Type {ItemOmission}	{ItemOmission} FAM_Type.SDLC_Data -> MAIN_CONNECTOR.SDLC_Data[No In Propagation]
FAM_Type {OutOfRange}	{OutOfRange} FAM_Type.SDLC_Data -> MAIN_CONNECTOR.SDLC_Data[No In Propagation]

Figura 25 Report Fault Impact- Parte 3

Dal report, seguendo il percorso dell'errore sull'alimentazione generatosi nella *POWER_ Board,* si può notare come questo errore si propaga ed influenza a vari livelli i vari sottocomponenti, generando, da un errore di alimentazione, un errore sui dati di temperatura. Sul modello infatti è stato definito che da una mancanza di alimentazione si genera una omissione del dato di temperatura, mentre se si presenta una bassa tensione sull'alimentazione si presentano dei fuoriscala. Questi errori si propagano fino al main connector, cioè fino all'utente.

Anche questa volta l'analisi fornita da OSATE si è dimostrata coerente ed esaustiva il modello del sistema e con l'idea progettuale alla base del modello stesso.

3.5 Conclusione

AADL e il suo ambiente di sviluppo OSATE2 sono risultati essere uno strumento estremamente utile ed efficiente nello studio di sistemi safety-critical e non solo, permettendo di modellare qualsiasi tipo di sistema, mettendo in evidenza le componenti HW e SW e le interazioni fra di esse. Il linguaggio fornito da AADL permette la creazione di modelli estremamente realistici, poiché fornisce gli strumenti per una modellazione a bassissimo livello e quindi di realizzare modelli fedelissimi alla realtà.

I benefici nell'usare AADL includono:

1. Predizione e validazione di caratteristiche runtime come accessibilità, tempestività di risposta e sicurezza.
2. Validazione di architetture di sistema e implementazioni
3. Potenzia lo sviluppo di processi attraverso un singolo modello architetturale dettagliato
4. Portabilità dei modelli su altre piattaforme di lavoro e altri linguaggi
5. Permette analisi valide di sistemi real-time, embedded e ad alta fedeltà.

Le analisi eseguite sul modello della IMU di Civitanavi sono risultate ottimali rispettando le aspettative richieste.

Una volta compresa la grammatica di AADL risulta molto semplice creare modelli ad alto livello e questo velocizza le operazioni di progettazione, permettendo delle analisi preliminari senza troppo dispendio di energie e risorse.

Ma AADL risulta essere efficiente e valido anche nella modellazione a bassissimo livello e le conseguenti analisi, rendendolo così un utile e consigliatissimo strumento per la progettazione di sistemi di qualsiasi azienda o industria.

APPENDICE

APPENDICE 1: Tabelle riassuntive delle gerarchie fra elementi AADL

Category Group	Component Category	Permitted Subcomponents	Permitted Subcomponent of
Software	process	thread data thread group	system
	thread	data	process thread group
	data	data	process thread data thread group system
	thread group	data thread thread group	process thread group
	subprogram	None allowed	None
Execution Platform	processor	memory	system
	memory	memory	processor memory system
	bus	None allowed	system
	device	None allowed	system
Composite	system	process data processor memory bus device system	system

Feature		Allowed Feature of Component or Component Category
port **port group**	all port types	• system • process • thread • thread group • processor • device
	• event port • event data port • port group (events only)	subprogram (component)
subprogram	server	• system • process • thread • thread group • processor • device
	subprogram (data)	data
access	provides data	• system • process • thread • thread group • data
	requires data	• system • process • thread • thread group • subprogram (component)
	provides bus	system
	requires bus	• system • processor • memory • bus • device
parameter		subprogram (component)

Category Group	Component Category	Allowed Features
Software	process	• server subprogram • port/port group • provides data access • requires data access
	thread	• server subprogram • port/port group • provides data access • requires data access
	data	• subprogram • provides data access
	thread group	• server subprogram • port/port group • provides data access • requires data access
	subprogram	• out event port • out event data port • port group (event only) • requires data access • parameter
Execution Platform	processor	• server subprogram • port/port group • requires bus access
	memory	requires bus access
	bus	requires bus access
	device	port/port group • server subprogram • requires bus access
Composite	system	• server subprogram • port/port group • provides data access • provides bus access • requires data access • requires bus access

APPENDICE 2: Libreria EMV2 ridotta e modificata per il caso di studio

Pacchetto contenente la libreria degli errori modificata da librerie già esistenti, contiene le tipologie di errore e le macchine a stati relative al Sistema.
Sono stati creati sia gli eventi sia gli stati che descrivono nella maniera più realistica possibile il comportamento reale del Sistema, il tutto ovviamente semplificato per ragioni di studio.

```
package Error_annex
  public
       annex EMV2 {**
             error types
                   NoPower: type;
                   NoService: type;
                   ValueError: type;
                   NoValue: type extends ValueError;
                   IncorrectData: type;
                   PlatformFailure: type;
                   HardwareFailure: type extends PlatformFailure;
                   SoftwareFailure: type extends PlatformFailure;
                   LOW_Power: type;
                   Power_Issue: type;
             end types;

             error behavior ThreeState
                   events
                         Failure: error event;
                         Red_Failure: error event;
                         Critical_Failure: error event;
                         Error_Power_15V: error event;
                         Error_Power_6V: error event;
                         Error_Power_5V: error event;
                         Stop: error event;
                         Brake: error event;
                         ResetEvent: recover event;
                         Reset: recover event;
                   states
                         Operational: initial state;
                         Failstop: state;
                         Error_15V_power: state;
                         Error_15V_LOW_Power: state;
                         Error_6V_power: state;
                         Error_6V_LOW_Power: state;
                         Error_5V_power: state;
                         Error_5V_LOW_Power: state;
                         Gyro_error_power: state;
                         NonCriticalModeFailure: state;
                         NCritF_15V: state;
                         NCritF_6V: state;
                         NCritF_5V: state;
                         CritF_15V: state;
                         CritF_6V: state;
                         CritF_5V: state;
                         CriticalModeFailure: state;

             end behavior;


error behavior ThreeErrorStates
```

```
                  events
                        BadValueEvent: error event;
                        NoValueEvent: error event;
                        LateValueEvent: error event;
                  states
                        Operational: initial state;
                        BadValueState: state;
                        NoValueState: state;
                        LateValueState: state;
                  transitions
                  BadValueTransition: Operational -[BadValueEvent]->
BadValueState;
                  NoValueTransition: Operational -[NoValueEvent]-> NoValueState;
                  LateTransition: Operational -[LateValueEvent]->
LateValueState;

            end behavior;

            error behavior Simple
                  states
                        Operational: initial state;
                        Failstop: state;
                        Failed: state;
            end behavior;


            error behavior SimpleAndTransient
                  events
                        Failure: error event;
                        FailureTransient: error event;
                        Warning_Out_of_Range: error event;
                        Critical_Out_of_Range: error event;
                        Critical_Error: error event;
                        ResetEvent: recover event;
                  states
                        Operational: initial state;
                        Failed: state;
                        Warning_Error: state;
                        TransientFailure: state;
                  transitions
                    transerr: Operational -[Failure]-> Failed;

                    transerr2: Operational -[FailureTransient]->
                  TransientFailure;
                    treset: TransientFailure -[ResetEvent]-> Operational;
            end behavior;
      **};
end Error_annex;
```

APPENDICE 3: tipologie di componenti

COMPOSITE

SYSTEM

```
FAM: system IMU::Hardware::IMU_Hardware.impl;
ASE200: system IMU::ASE200_Board::IMU_ASE_Board.impl;
Sensing_Unit: system IMU::Hardware::Sensing_Elements.impl;
POWER_Board: system IMU::Power_Unit::Power_Board.impl;
SoC: system IMU::SoC::SoC.impl;
Gyroscope: system Gyro.impl;
```

HARDWARE

PROCESSOR

```
cpu: processor SoC_CPU.impl;
fpga: processor SoC_fpga.impl;
cpu: processor IMU::SoC::SoC_CPU.impl;
```

DEVICE

```
EMI_FLT: device EMI_Filter.impl;
SURGE_STOP: device Surge_Stopper.impl;
DCDC_15V: device DC_DC_15V.impl;
DCDC_6V: device DC_DC_6V.impl;
DCDC_5V: device DC_DC_5V.impl;
channel_8_AD_1: device channel_8_AD.impl;
channel_8_AD_2: device channel_8_AD.impl;
FOG_AD_Converter: device FOG_AD.impl;
DIGITAL_PWR: device Digital_Power_Supply.impl;
accel_x: device accel.impl;
accel_y: device accel.impl;
accel_z: device accel.impl;
temp_accel_x: device Temperature_sensor.impl;
temp_accel_y: device Temperature_sensor.impl;
temp_accel_z: device Temperature_sensor.impl;
Coil_x_Temp_Sensor: device Temperature_Sensor.impl;
Coil_y_Temp_Sensor: device Temperature_Sensor.impl;
Coil_z_Temp_Sensor: device Temperature_Sensor.impl;
MIOC_x: device MIOC_XYZ.impl;
MIOC_y: device MIOC_XYZ.impl;
MIOC_z: device MIOC_XYZ.impl;
Fiber_Coil_x: device Fiber_Coil.impl;
Fiber_Coil_y: device Fiber_Coil.impl;
Optical_unit: device IMU::ASE200_Board::Optical_Circuit.impl;
Fiber_Coil_z: device Fiber_Coil.impl;
Pinfet: device Pinfet_Circuit.impl;
Filter_1: device IMU::Hardware::Filter.impl;
Fiber_Coil_z: device Fiber_Coil.impl;

…
…
```

…

MEMORY

```
SDRAM: memory Flash_SDRAM.impl;
```

…

BUS

```
AXI_BUS: bus AXI_bus.impl;
```

…

SOFTWARE

PROCESS

```
firmware: process IMU::Firmware::IMU_firmware.impl;
IMU_START_CSCI: process IMU::IMU_SW_START_CSCI::IMU_START_CSCI.impl;
IMU_MAINT_CSCI: process IMU::IMU_SW_MAINT_CSCI::IMU_MAINT_CSCI.impl;
IMU_NAV_CSCI: process IMU::IMU_SW_NAV_CSCI::IMU_NAV_CSCI.impl;
```

THREAD

```
BOOT: thread Boot.impl;
CSCI_Selection: thread CSCI_Selection.impl;
BIT_Stream_VAL_ACT: thread BIT_Stream_VAL_ACT.impl;
CSCI_VALIDATION: thread CSCI_Validation.impl;
CSCI_ACTIVATION: thread CSCI_Activation.impl;
```

SUBPROGRAM

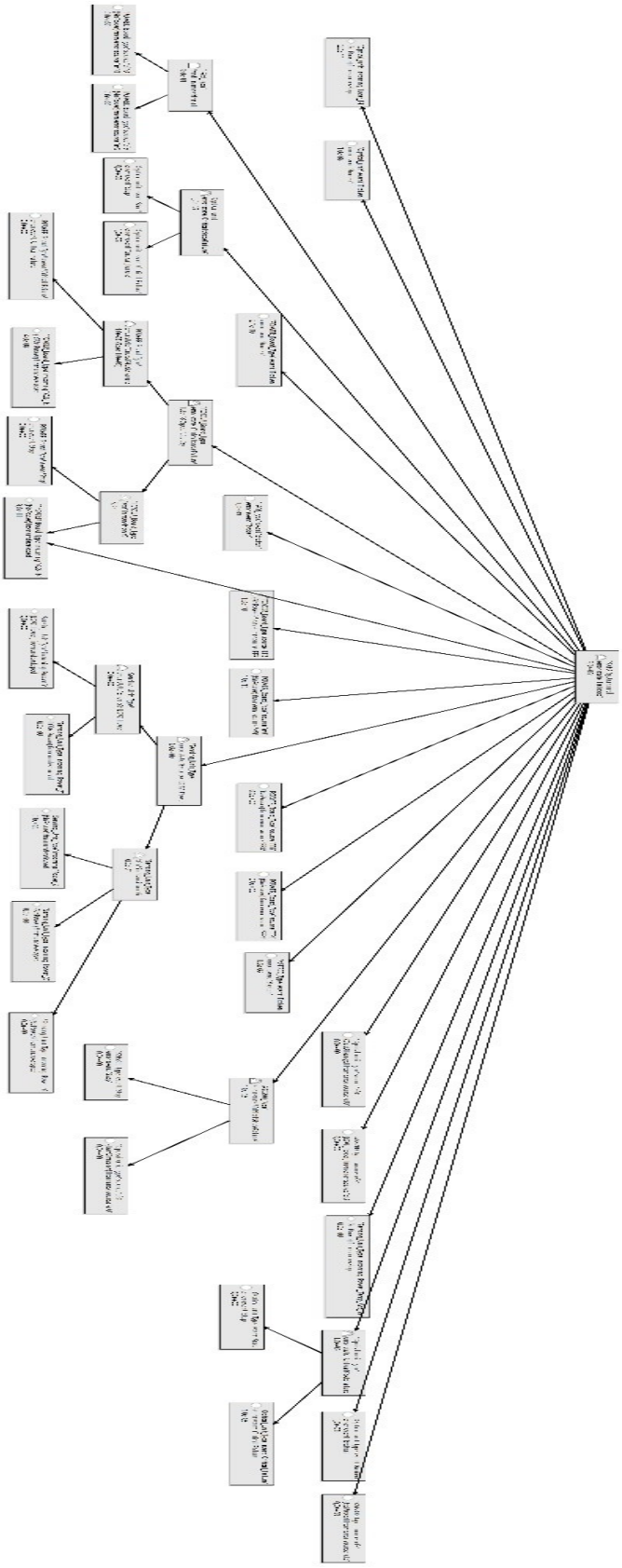
DATA

```
data UINT32_TYPE_AADL
  properties
    Type_Source_Name => "__po_hi_uint32_t";
    Source_Text => ("types");
    Data_Size => 4 Bytes;
  end UINT32_TYPE_AADL;

  data FLOAT32_TYPE_AADL
  properties
    Type_Source_Name => "__po_hi_float32_t";
    Source_Text => ("types");
    Data_Size => 4 Bytes;
  end FLOAT32_TYPE_AADL;
end IMU::Data_types;
```

…

APPENDICE 4: Fault Tree



APPENDICE 5: Macchine a stati dei vari elementi che contribuiscono allo stato di Failstop nel FTA

IMU_System

```
composite error behavior
    states

        [POWER_Board_Type.Failstop]-> Failstop;
        [Optical_unit.Failstop]-> Failstop;
        [FAM_Type.Failstop]-> Failstop;
        [ASE200_Type.Failstop]-> Failstop;
        [Sensing_Unit_Type.Failstop]-> Failstop;
        [Optical_unit_Type.Failstop]-> Failstop;

end composite;
```

FAM Type

```
component error behavior
    events
        Failure: error event;
        Critical_Failure: error event;
        Reset: recover event;
        Stop: error event;
    transitions
        t0: Operational -[Power_15V{NoPower}]-> Failstop;
        t1: Operational -[Power_6V{NoPower} and Power_5V{NoPower}]-> Failstop;
        t2: Operational -[Power_5V{NoPower}]-> CritF_5V;
        t3: Operational -[Power_15V{LOW_Power}]-> NCritF_15V;
        t4: Operational -[Power_6V{LOW_Power}]-> NCritF_6V;
        t5: Operational -[Power_5V{LOW_Power}]-> NCritF_5V;
        t01: Operational -[Power_Good_15V{NoPower}]-> Failstop;
        t12: Operational -[Power_Good_6V{NoPower}]-> Failstop;
        t23: Operational -[Power_Good_5V{NoPower}]-> Failstop;
        t34: Operational -[Power_Good_15V{LOW_Power}]-> NCritF_15V;
        t45: Operational -[Power_Good_6V{LOW_Power}]-> NCritF_6V;
        t56: Operational -[Power_Good_5V{LOW_Power}]-> NCritF_5V;
        t67: Operational -[Broken]-> Failstop;
        t6: CriticalModeFailure -[Reset]-> Operational;
        t7: Failstop -[Reset]-> Operational;

end component;
```

FAM Impl

```
composite error behavior
    states
        [                  impl.Operational]-> Operational;
        [                  impl.Failstop]-> Failstop;
        [                  impl.NonCriticalModeFailure]-> NonCriticalModeFailure;
        [                  impl.CriticalModeFailure and Z          _impl.NonCriticalModeFailure]-> Failstop;
        [                  impl.CriticalModeFailure ]-> CriticalModeFailure;

end composite;
```

Sensing Unit Type

```
component error behavior
    events
        Reset: recover event;
    transitions
        t0: Operational -[Power_X{NoPower} or Power_Y{NoPower} or Power_Z{NoPower}]-> Error_5V_power;
        t11: Operational -[Power_Temp_Acc_5V{NoPower}]-> Failstop;
        t1: Operational -[Power_Temp_Acc_5V{LOW_Power}]-> NCritF_5V;
        t2: Operational -[Power_Y{LOW_Power}]-> Error_5V_LOW_Power;
        t3: Operational -[Power_Z{LOW_Power}]-> Error_5V_LOW_Power;
        t4: Operational -[Error_Power_5V]-> Error_5V_power;
        t5: Error_5V_LOW_Power -[ResetEvent]-> Operational;
        t6: Error_5V_LOW_Power -[Power_X{NoPower} or Power_Y{NoPower} or Power_Z{NoPower}]-> FailStop;
end component;
```

Sensing Unit Impl.

```
composite error behavior
    states
        [temp_accel_x.Failstop or temp_accel_y.Failstop or temp_accel_z.Failstop]-> Failstop;
        [temp_accel_x.CritF_5V or temp_accel_x.CriticalModeFailure]-> CriticalModeFailure;
        [temp_accel_x.NCritF_5V]-> NCritF_5V;

end composite;
```

Sensore di Temperatura Accel_X Type

```
component error behavior
    events
        Reset: recover event;
    transitions
        t0: Operational-[accel_failure]-> FailStop;
        t01: Operational-[Power_5V{LOW_Power}]-> CriticalModeFailure;
        t02: Operational-[Power_5V{NoPower}]-> Failstop;
        t03: NCritF_5V-[Reset]-> Operational;
        t04: CritF_5V-[Broken]-> FailStop;
        t05: CritF_5V-[Reset]-> Operational;
        t1: FailStop -[Reset]-> Operational;
        t2: Operational-[Critical_Failure]-> CriticalModeFailure;
        t3: CriticalModeFailure-[Stop]-> FailStop;
end component;
```

Optical Unit Type

```
component error behavior
    events
        Reset: recover event;
    transitions
        t0: Operational-[Broken]-> FailStop;
        t01: Operational-[Laser_IN{NoPower}]-> FailStop;
        t02: Operational-[Laser_IN{LOW_Power}]-> CritF_15V;
        t04: CritF_15V-[Broken]-> FailStop;
        t05: CritF_15V-[Reset]-> Operational;
        t1: FailStop -[Reset]-> Operational;
        t2: Operational-[Critical_Failure]-> CriticalModeFailure;
        t3: CriticalModeFailure-[Stop]-> FailStop;
end component;
```

ASE Board Type

```
component error behavior
    events
        Reset: recover event;
    transitions
        t0: Operational-[Broken]-> FailStop;
        t01: Operational-[Optical_x_OUT{OutOfRange}]-> FailStop;
        t02: Operational-[Optical_x_OUT{ItemOmission}]-> CriticalModeFailure;
        t3: CriticalModeFailure-[Stop]-> FailStop;
end component;
```

Processore SoC Type

```
  component error behavior
      events
          Failure: error event;
          Critical_Failure: error event;
          Reset: recover event;
          Stop: error event;
      transitions
          t0_1: Operational -[Power_15V{LOW_Power}]-> NonCriticalModeFailure;
          t0: Operational -[Acc_Temp_x{OutOfRange}]-> NonCriticalModeFailure;
          t1: Operational -[Power_15V{NoPower}]-> Failstop;
          t2: Operational -[Acc_Temp_x{ItemOmission}]-> CriticalModeFailure;
          t3: NonCriticalModeFailure -[ResetEvent]-> Operational;
          t4: NonCriticalModeFailure -[Power_15V{NoPower}]-> Failstop;
          t5: CriticalModeFailure -[Stop]-> Failstop;
          t6: CriticalModeFailure -[Reset]-> Operational;
          t7: Failstop -[Reset]-> Operational;

  end component;
```

Processore SoC Impl

```
composite error behavior
    states

        [PS_MODULE_impl.Operational]-> Operational;
        [PS_MODULE_impl.Failstop or PL_MODULE_impl.Failstop]-> Failstop;
        [PS_MODULE_impl.NonCriticalModeFailure]-> NonCriticalModeFailure;
        [PS_MODULE_impl.CriticalModeFailure]-> CriticalModeFailure;

end composite;
```

PS MODULE Type e Impl.

```
    component error behavior
        events
            Failure: error event;
            Critical_Failure: error event;
            Reset: recover event;
            Stop: error event;
        transitions
        --  t0: Operational -[Failure]-> Operational;
            t1: Operational -[Red_Failure]-> NonCriticalModeFailure;
            t2: Operational -[Critical_Failure]-> CriticalModeFailure;
            t3: NonCriticalModeFailure -[ResetEvent]-> Operational;
            t4: NonCriticalModeFailure -[Critical_Failure]-> CriticalModeFailure;
            t5: CriticalModeFailure -[Stop]-> Failstop;
            t6: CriticalModeFailure -[Reset]-> Operational;
            t7: Failstop -[Reset]-> Operational;

    end component;

    composite error behavior
        states

            [Software.Operational]-> Operational;
            [Software.Failstop]-> Failstop;
            [Software.NonCriticalModeFailure]-> NonCriticalModeFailure;
            [Software.CriticalModeFailure]-> CriticalModeFailure;

    end composite;
```

APPENDICE 6: Lista delle Immagini

Bibliografia

- Delange J. AADL IN PRACTICE: design and validate the architecture of critical systems. United State of America. Reblochon Development Company. 2017.

- Feiler P. SAE AADL V2: An Overview. Pittsburgh. Software Engineering Institute-Carnegie Mellon University

- Feiler P, Hudak J, Delange J, Gluch D. Architecture Fault Modeling and Analysis with the Error Model Annex, Version 2. Pittsburgh. Software Engineering Institute-Carnegie Mellon University. 2016.

- Feiler P. Hansson J. Flow Latency Analysis with the Architecture Analysis and Design Language (AADL). Pittsburgh. Software Engineering Institute-Carnegie Mellon University. 2007.

- Feiler P. Hudak J. Developing AADL Models for Control Systems: A Practitioner's Guide. Pittsburgh. Software Engineering Institute-Carnegie Mellon University. 2007.

- Kordon F. Hugues J. Canals A. Dohet A. Part 4 AADL. Embedded Systems: Analysis and Modeling with SysML, UML and AADL. Wiley-ISTE. 2013.

- Larsen M. Modelling fiel robot software using AADL. Aarhus University. 2016.

- Adventium LABS. Integrated AADL Analysis. 2018

- Hansson J. Greenhouse A. Modeling and Validating Security and Confidentiality in System Architectures. Pittsburgh. Software Engineering Institute-Carnegie Mellon University. 2008.

- Bozzano M. Cimatti A. Katoen J.P. Noll T. Safety, Dependability and Performance Analysis of Extended AADL Models. The Computer Journal. 2011.

- Feiler P. Hudak J. The Architecture Analysis & Design Language (AADL): An Introduction. Software Engineering Institute-Carnegie Mellon University. 2006.

- Cicchetti S. CRANE-T Software Requirements Specification. Civitanavi Systems S.r.l. 2017

- Quatraro E. CRANE-T System Architecture Description. Civitanavi Systems S.r.l. 2018

- Camilletti M. CRANE-T Interface Control Document. Civitanavi Systems S.r.l. 2018