Diseño de una arquitectura de aviónica tolerante a fallas basada en componentes COTS para vehículos satelitales de nueva generación

Por ***Arias Emmanuel***

Presentado ante la Universidad Nacional de La Matanza y la Unidad de Formación Superior de la CONAE como parte de los requerimientos para la obtención del grado de

MAGISTER EN DESARROLLOS INFORMATICOS DE APLICACION ESPACIAL

UNIVERSIDAD NACIONAL DEL CÓRDOBA

Mayo, 2017

DIRECTOR
Gustavo Wiman
INVAP, Bariloche, Provincia de Rio Negro

Dedicado a
alguien

Abstract

Keywords:

Resumen

Esta tesis se trata de

Agradecimientos

Muchas gracias

Tabla de Contenidos

Índice de figuras

Índice de tablas

Lista de acrónimos

SW Software

HW Hardware

FT Tolerancia a Fallas

FA Evitación de Fallas

FR Eliminación de Fallas

FF Predicción de Fallas

MTBF Tiempo Medio Entre Fallas

MTTF Tiempo Medio de Fallas

MTTR Tiempo Medio de Reperación

CMF Modo Común de fallas

FDIR Detección, Aislación y Recuperación de Fallas

CONAE Comisión Nacional de Actividades Espaciales

UNLAM Universidad Nacional de La Matanza

INVAP Investigación Aplicada

NASA National Aeronautics and Space Administration

COTS Commercial Off-The-Shelf

TT Time-Trigged

CST Complete Stack-Tree

CAN Controller Area Network

CPU Central Proceesing Unit

Todo list

Capítulo 1

Introducción

En el marco del Plan Espacial Nacional, desarrollado por la Comisión Nacional de Actividades Espaciales (CONAE) de Argentina, y con el propósito de llevar a cabo actividades de investigación y aplicación, provenientes de la Universidad Nacional de La Matanza (UNLAM) se presenta este plan de tesis con el fin de ampliar los conocimientos y la participación de la CONAE y UNLAM, en el campo del Desarrollo Informático y Ciencias de la Computación.

Las actividades desarrolladas para este trabajo de tesis son realizadas, en su mayor proporción, en la Unidad de Desarrollo Investigación Aplicada (INVAP), ubicada en San Carlos de Bariloche, Provincia de Río Negro. Este trabajo se encuentra orientado a brindar un nuevo conocimiento, que ayude en cierta medida, en el desarrollo de los diferentes proyectos con los que cuenta actualmente esta empresa, agregando un grado de innovación en el resultado que se obtenga.

INVAP tiene como visión ser un referente en proyectos tecnológicos a nivel mundial INVAP (2016), por lo tanto, debe asegurarse que cada uno de los productos que se lleven a cabo sean competitivos. Para lograr cumplir con esto, es necesario que tales proyectos se encuentren a la vanguardia tecnológica y científica.

El desarrollo de proyectos satelitales conlleva costos de importante magnitud, y dependen de cada misión. Una parte importante de los costos está conformado por el desarrollo[1] y sobre todo los materiales que se utilizan para su fabricación. Esto es debido a que se utilizan componentes que son exclusivos para el ámbito espacial, en otras palabras que se encuentran "calificados para volar". Estos componentes son fabricados especialmente para soportar el ambiente hostil del espacio.

Si se considera al ámbito espacial como una industria, algo que ha sido demostrado en los últimos años; y si se tiene en cuenta las intenciones de crecimiento y competitividad de la empresa INVAP, de permitir el ingreso de nuestro pais en el mercado satelital INVAP (2016), resulta de gran importancia lograr reducir los costos en fabricación y desarrollo de vehículos satelitales.

La National Aeronautics and Space Administration (NASA) tiene un enfoque de desarrollo bajo el lema "faster, cheaper, better" Forsberg y Harold (1999), lo cual busca desarrollar sus proyectos y misiones de foma rápida, barata y mejor. Bajo este enfoque se han realizado diversos estudios e investigaciones dando resultados sumamente positivos Tai et al. (1999), Chau et al. (1999), Schneidewind y Nikora (1998), Forsberg y Harold (1999). En estos trabajos se utilizan componentes que no se encuentran "calificados para volar", los cuales también son llamados componentes Commercial Off-The-Shelf (COTS), o de estantería. Debe mencionarse, que también hubo algunos fracasos en su aplicación.

[1] *Nota: entiéndase por desarrollo al proceso de planificación, análisis, diseño e implementación.*

A simple vista, la utilización de estos componentes ayudaría a reducir costos. Sin embargo, esto no es tan directo. Los componentes COTS al no estar calificados, se les deben realizar tareas de calificación adicional. Además deben ser aplicados a un ambiente, que asegure que no fallarán durante la misión; o si fallan, no será motivo de pérdida de la misma.

Los componentes COTS suelen tener un costo de compra entre 100 y 1000 veces menores que aquellos que está califcados para volar. Por lo que el aumento en la utilización de estos componentes, aplicados al desarrollo de diferentes tipos de satélite, **permitiría reducir los costos y ahorrar algunos millones de dólares del proyecto satelital.** Esto facilitaría el ingreso de Argentina en un mercado altamente competitivo.

El desafío de este trabajo de tesis es analizar y estudiar arquitecturas que sean tolerantes a fallas, que permitan una correcta comunicación entre los diferentes subsistemas de un vehículo espacial de nueva generación, y que tenga como característica principal un cierto grado de confiabilidad, de modo tal que pueda ser aplicado con componentes COTS.

1.1. Motivación

Los costos de un proyecto satelital se pueden clasificar, a grandes rasgos, en 5 grupos:

- Desarrollo
- Materiales
- Ensamblado, integración, y tests
- Lanzamiento
- Operaciones

Este trabajo de tesis se centrará principalmente en el desarrollo (proceso de planificación, análisis, diseño e implementación.), y en los materiales utilizados en la fabricación de vehículos satelitales.

No se puede mencionar a ciencia cierta cuál es el costo "verdadero" de desarrollar un satélite. Este depende exclusivamente del tipo de satélite y de la misión. Lo que si se debe tener en claro es que las tareas de desarrollo representan una parte muy importante del costo total del proyecto.

Desarrollar un vehículo espacial con componente COTS, en un principio podría representar costos adicionales, ya que se le deben realizar tareas de calificación adicional, debido a que no están "preparados" para resistir las condiciones hostiles del espacio.

Uno de los puntos positivos, y que motivan la aplicación de componentes COTS, es que a la hora de desarrollar varios satélites en base a la misma ingeniería, se puede ahorrar en gran medida en los materiales que se utilizan. Los componentes COTS suelen tener un costo de compra entre 100 y 1000 veces menores que aquellos que están calificados para volar. **Esto ayudaría a ahorrar algunos millones de dólares de los proyectos satelitales.**

Otra de las ventajas de utilizar componentes COTS, es que la mayoría cuentan con una tecnología más avanzada que aquellos que son calificados para volar. Esta tecnología permite:

- Aumentar prestaciones, mediante el incremento de las capacidades de procesamiento, memoria, velocidades de procesamiento, etc.
- Implementar funciones que son imposibles de aplicar en tecnologías viejas.

- Reducir tiempos de desarrollo.
- Reducir volumen, masa y consumo

El último punto mencionado anteriormente es de especial interés, ya que al reducir volumen y masa, permite reducir costos adicionales como el de lanzamiento.

Esta reducción de costos de proyectos satelitales tienen ventajas directas a la hora de introducir a Argentina en un mercado altamente competitivo, donde la mínima reducción de estos, representa ganancias económicas importantes.

Uno de los puntos en contra de la utilización de componentes COTS es que al no ser calificados para volar, es necesario llevar a cabo tareas y estrategias inteligentes, con el fin de hacer frente a esa "deficiencia". Por ello, se exige realizar una investigación y análisis de diferentes arquitecturas de aviónica, que puedan ser utilizadas para lograr que el sistema sea tolerante a fallas, y así, cumplir con los requerimientos de una misión satelital.

El estudio de arquitecturas tolerantes a fallas, no solamente tiene aplicación en el ámbito espacial, si no que también puede ser extendido a cualquier sistema crítico, los cuales necesitan ser robustos y tolerantes a fallas, como es el caso de aviones comerciales, plantas nucleares, automóviles, etc.

1.2. Hipótesis

La hipótesis de esta tesis es la siguiente: "Una arquitectura de aviónica basadas en componentes COTS, robusta y tolerante a fallas, es totalmente aplicable y utilizable en vehículos espaciales, con un alto nivel de confiabilidad, lo cual permite disminuir la complejidad de los sistemas actuales de aviónica".

1.3. Objetivo del trabajo y preguntas de investigación

En esta sección se mencionan el objetivo principal y específicos; como así también las preguntas de investigación, que guiarán la elaboración de esta tesis.

1.3.1. Objetivo

El objetivo de este trabajo es investigar y analizar arquitecturas de comunicación de los subsistemas de aviónica tolerante a fallas basada en componentes COTS para vehículos satelitales de nueva generación.

1.3.2. Objetivos Específicos

1. Realizar un estudio del estado de la cuestión sobre arquitecturas tolerantes a fallas para sistemas críticos.
2. Investigar y analizar arquitecturas tolerantes a fallas que aseguren la confiabilidad del sistema y que sean aplicables en la industria satelital.
3. Investigar y analizar protocolos de comunicación, para las capas superiores del modelo de OSI (modelo de interconexión de sistemas abiertos - ISO/IEC 7498-1), orientados a la tolerancia a fallas y confiabilidad de los sistemas. Realizar un estudio comparativo de los diferentes protocolos estudiados.
4. Investigar una metodología para lograr una medición de la tolerancia a fallas en arquitecturas de aviónica.

5. Desarrollar un estudio comparativo de arquitecturas tolerantes a fallas con el fin de obtener ventajas y desventajas de cada una de ellas.
6. Diseñar modelos alternativos de arquitecturas tolerantes a fallas, que tenga un grado de confiabilidad tal que permita la aplicación de componentes COTS.
7. Evaluar la confiabilidad de los modelos de arquitecturas (mediante métrica desarrollada en este trabajo o siguiendo otras estrategias).
8. Proponer el diseño de una nueva arquitectura tolerante a fallas, con un grado de confiabilidad suficiente para la aplicación de componentes COTS en aviónicas de vehículos satelitales.
9. Simular la arquitectura planteada para medir su grado de tolerancia a fallas y perfomance.

1.3.3. Preguntas de investigación

Ver qué preguntas se lograron responder y mostrarlas en algún apartado

Para este trabajo de tesis se plantearon las siguientes preguntas de investigación, que se fueron respondiendo a lo largo de este trabajo

- ¿Es posible la realización de un método de medición del grado de tolerancia a fallas de una arquitectura de aviónica?
- ¿Cuál es la estrategia más indicada de tolerancia a fallas que permita brindar un alto grado de confiabilidad en la utilización de componentes COTS en sistemas críticos?
- ¿Cuál es la arquitectura más indicada que permita desarrollar tolerancia a fallas en sistemas críticos basados en componentes COTS?
- ¿Es factible la utilización de componentes COTS en sistemas espaciales?

Capítulo 2

Marco Teórico

I recall seeing a package to make quotes

Snowball

2.1. Terminología

Existe una importante diferencia entre los significados de las palabras falla, error y avería[1], que es importante destacar antes de comenzar con el desarrollo de este trabajo.

Un **avería** de sistema ocurre cuando el servicio prestado por el sistema ya no coincide con las especificaciones del mismo (Hanmer, 2007). Esto quiere decir que existe un problema que tiene una consecuencia negativa en el sistema completo, logrando que este ya no logre cumplir con sus especificaciones. Cuando el sistema no se comporta de la manera que es especificada, este ha fracasado. Esto significa que lo que se espera de un sistema se encuentra descripto, comúnmente en especificaciones o requerimientos (Pullum, 2001).

Para la IEEE (1990) avería es "la inhabilitación de una sistema o componente a llevar a cabo las funciones requeridas en los requerimientos específicos de perfomance del mismo".

Hanmer (2007) ejemplifica averías de sistemas cuando: el sistema se bloquea y se detiene cuando no debería hacerlo, el sistema calcula un resultado incorrecto, el sistema no está disponible, el sistema es incapaz de responder a la interacción con el usuario. Cuando el sistema no hace lo que debe hacer, el sistema ha fracasado. Las averías son detectados por los usuarios mientras usan el sistema.

Las averías son causados por los errores. Un **error** es una parte del estado del sistema que es susceptible de provocar un avería en el sistema, un error que afecta al servicio es una indicación de que un avería se ha producido (Hanmer, 2007). Un error se puede propagar, es decir dar a lugar otros errores (Pullum, 2001).

IEEE (1990) define error como "la diferencia entre un valor computado, observado o medido, con el valor verdadero, especificado o el teóricamente verdadero".

Los errores se pueden clasificar en dos tipos: errores de tiempo y valores (Hanmer, 2007). Los errores de valores son aquellos que se manifiestan como valores discretos incorrectos o estados del sistema incorrecto. En cambio, los errores de tiempo pueden incluir aquellos que no cumplen con el total de las tareas.

[1] En inglés: fault, error y failure.

Hanmer (2007) especifica los siguiente casos más comunes de errores:

- Timing: existe una falta de sincronización en la comunicación de los procesos.
- Bucles infinitos: ejecución de un bucle sin detenerse, esto consume memoria, y la avería del sistema.
- Error de protocolo: errores en el flujo de comunicación ya que no coinciden los protocolos. Mensajes enviados en formato diferente, en tiempos diferentes, a lugares de sistemas incorrectos.
- Inconsistencia de datos: errores son diferentes en diferentes lugares.
- Sobrecarga de sistema: el sistema es incapaz de hacer frente a la sobrecarga de actividades a la que es expuesta.

La causa adjudicada o la hipótesis de un error es una **falla**, también llamado "bugs". Una **falla activa** es aquella que produce un error (Pullum, 2001). Una falla es un defecto que está presente en el sistema y que puede causar un error (Hanmer, 2007). Es la desviación actual de lo correcto Hanmer (2007).

Según IEEE (1990) una falla es "un defecto en un dispositivo de hardware o componente; como por ejemplo un corto circuito o un cable cortado". También realiza una segunda definición diciendo que falla es "un paso incorrecto, proceso, o definición de dato en un programa de computadora" IEEE (1990). Esta última afirmación es la que se usa en el ámbito de este trabajo.

Algunas fallas introducidas en el Software (SW) se detallan en Hanmer (2007), lo cual señala que pueden incluir:

- Especificaciones incorrecta de requerimientos
- Diseño incorrecto
- Errores de programación

Entonces, como lo indica Pullum (2001) con la tolerancia a fallas, lo que se busca es prevenir la avería mediante la "tolerancia" de fallas, las cuales son detectables cuando un error aparece. Las fallas son el motivo de errores y los errores son motivos de avería (Dubrova, 2013).

También se suele utilizar el término anomalía en las operaciones de vehículos espaciales para referirse a comportamientos anómalos o no esperados del sistema (David M. Harland, 2005)

En Dubrova (2013) se describe un ejemplo para diferenciar correctamente estos conceptos. Se considera el SW de una planta nuclear, en la cual existe una computadora que es responsable de controlar la temperatura, la presión y demás variables de interés para la seguridad del sistema. Se da el caso de que uno de los sensores detecta que la turbina principal se encuentra girando a una velocidad menor a la correcta. Esta falla hace que el sistema envíe una señal para aumentar su velocidad (error). Esto produce un exceso de velocidad en la turbina, lo cual tiene como consecuencia que la seguridad mecánica apague la turbina. En esta situación el sistema no está generando energía. Esto se considera un avería, porque el sistema no está entregando el servicio según lo establecido por los requerimientos. Pero es un avería salvable.

Otro concepto es el de **mantenibilidad**, este es la capacidad de un sistema, bajo condiciones normales, de ser restaurado a un estado en el cual puede realizar sus funciones requeridas, cuando se realiza el mantenimiento (Rausand y Hoyland, 2004).

En secciones posteriores se ven los conceptos de confiabilidad, disponibilidad y seguridad (Sección 2.5.1, 2.5.2, 2.5.3, respectivamente).

2.2. La fiabilidad en el software

El objetivo final de la Tolerancia a Fallas (FT), es el desarrollo de un sistema fiable (Dubrova, 2013). Teniendo en cuenta que SW que se encuentran dentro de las naves espaciales, como satélites, lanzadores, y sobre todo vehículos tripulados son críticos, ya que de ellos dependen el éxito o fracaso de una misión, se debe llevar a cabo un sistema fiable. La fiabilidad de un sistema es la capacidad del mismo de entregar a los usuarios un nivel deseado de servicio (Dubrova, 2013).

La fiabilidad también se la puede considerar como una propiedad global que permite justificar la confianza de los servicios de un sistema (Hitt y Mulcare, 2070). Por lo tanto, como lo indica Hitt y Mulcare (2070) la fiabilidad es un término amplio y cualitativo que está relacionado con atributos no funcionales (o "-ilities"), que buscan generar un sistema "ideal", especialmente cuando su funcionamiento es crítico.

Como se muestra en la Figura 2.1 la consecuencia de la fiabilidad es la relación entre la evitación de fallos y la reducción de fallos, así como también la FT.

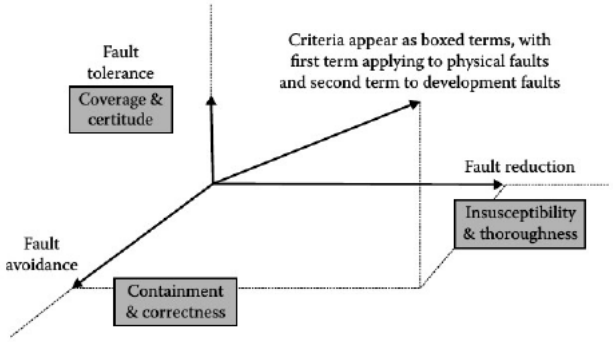

Figura 2.1: Fiabilidad (Hitt y Mulcare, 2070)

Según Pullum (2001) la fiabilidad puede ser clasificada en:

- Impedimentos: son aquellas cosas que se interponen en el camino de la fiabilidad. Son las fallas, errores y fracasos.
- Medios: los medios para lograr la fiabilidad, según el autor, se pueden dividir en dos grupos:
  1. Aquellos que son utilizados durante la construcción del SW (Evitación de Fallas (FA)[2] y FT).
  2. Aquellos que contribuyen con la validación del SW una vez desarrollado (Eliminación de Fallas (FR)[3] y Predicción de Fallas (FF)[4]).
- Atributos: describen las propiedades de la fiabilidad y proporcionan una forma de evaluar el logro de esas propiedades.

2.3. Impedimentos de la confiabilidad

El impedimiento de la confiabilidad o deteriodo de la confiabilidad es definido en términos de fallas, errores, y fracaso (Dubrova, 2013). Los mismos fueron desarrollados en la sección 2.1. Lo que tienen en común estos

[2]En inglés, Fault Avoidance
[3]En inglés, Fault Removal
[4]En inglés, Fault Forecasting

tres conceptos, es que avisan, o dan alerta cuando algo está mal (Dubrova, 2013). La diferencia radica, en que las fallas son a nivel físico; los errores se dan a nivel computacional; mientras que los fracaso se dan a nivel de sistema Dubrova (2013).

2.3.1. Orígenes de la falla

Existen diversos orígenes de fallas. Estas pueden provenir desde terceros, en el caso de productos comprados, pueden deverse a una falta del conocimiento del problema, falta de tiempo, etc. Dubrova (2013) clasifica el orígen de las fallas en cuatro grupos: *especificación incorrecta, implementación incorrecta, defectos de fabricación y factores externos*

Las *especificaciones incorrectas* son aquellas que surgen debidas a una incorrecta especificación de requerimiento o un mal diseño de una arquitectura o de un algoritmo (Dubrova, 2013). Estos orígenes de fallas son bastante comúnes en el desarrollo de sistemas. Un ejemplo típico citado por Dubrova (2013), es el caso de requerimientos que ignoran aspectos del medio ambiente en el que opera el sistema. Una mala redacción de un requerimiento o el olvido de uno de ellos, puede traer graves problemas, atrasos y pérdida de dinero, en el diseño y producción de un sistema espacial.

Las *implementaciones incorrectas*, se refieren a las *fallas de diseño*, surgen cuando el sistema implementado no cumple con los requerimientos (Dubrova, 2013).

Otro orígen de falla son los *defectos de los componentes* (Dubrova, 2013).Estos pueden incluir defectos de fabricación, defectos aleatorios dados en los componentes, etc.

Y por último se tienen las fallas que son causadas por *factores externos*, los cuales provienen del medio ambiente, usuarios u operadores (Dubrova, 2013). Ejemplos de estos factores externos pueden ser, vibraciones, cargas electroestáticas, temperatura, radición electromagnética, envío incorrecto de comandos, etc.

2.3.2. Modos comúnes de fallas

Un *Modo Común de fallas (CMF)*[5] es una falla que ocurre simultaneamente en dos o más componentes redundantes (Dubrova, 2013).

Gangloff (1975) define los CMF como múltiples unidades de fracaso debido a una sola causa.

CMF son causados por fenómenos que crean dependencias entre unidades redundadas, lo que causa la falla de estas unidades simultaneamente (Dubrova, 2013).

Según como lo indica Dubrova (2013) el único enfoque para combatir los CMF, es mediante el diseño en diversidad. Diseño en diversidad es la implementación de más de una variante de la función en cuestión (Dubrova, 2013). Esto se puede lograr variando los algoritmos que se utilizan, diferentes equipos de trabajo realicen las mismas partes del sistema, de manera tal de tener redundancia en código, etc.

2.3.3. Fallas en el Software

El SW difiere en gran medida con el Hardware (HW). En primer lugar el SW no envejece, no se deforma, tampoco se puede quebrar ni ser afectado por el medio ambiente. El SW es determinístico, siempre responde de la misma manera en el mismo ambiente, al menos que falle.

[5] En inglés, common-mode faults

Por otro lado el SW se lo puede actualizar varias veces a lo largo del su ciclo de vida.

En tercer lugar, arreglar bugs de SW **no significa que el mismo sea más confiable**, al contrario pueden ocurrir nuevos errores (Dubrova, 2013).

Por último el SW es mucho más complejo y menos regular que el HW. Tests tradicionales y métodos de debug pueden ser inadecuados para los sistemas de SW

2.4. Medios de fiabilidad

Los medios de confiabilidad son métodos y técnicas que permiten el desarrollo de un sistema confiable (Dubrova, 2013). Los medios se pueden dividir en dos grandes grupos (Pullum, 2001):

1. Aquellos que son empleados durante el proceso de construcción del SW (Pullum, 2001),
2. y a quellos que ayudan en la validación del SW después que fue desarrollado (Pullum, 2001).

Dentro del primer grupo se tiene:

- Evitación de Fallas
- Tolerancia a Fallas

Por otro lado, en el segundo grupo se puede mencionar los siguientes:

- Eliminación de Fallas
- Predicción de Fallas

2.4.1. Evitación de Fallas

FA son técnicas de mejoramiento de la fiabilidad utilizadas durante el desarrollo de SW para reducir el número de fallas introducidas durante la etapa mencionada (Pullum, 2001). Estas técnicas pueden estar presentes en las especificaciones y requerimientos del sistema, métodos de diseño de SW (Pullum, 2001).

Dubrova (2013) la denomina *prevensión de fallas*[6], y coincide con el autor anterior, definiendo FA como técnicas de control de calidad durante la especificación y fabricación de los procesos de diseño.

2.4.2. Tolerancia a Fallas

En la sección 2.6 (página 9) se discutirá con mayor detalle la FT. Esto es así ya que en este trabajo de tesis se tiene como principal punto de estudio la Tolerancia a Fallas.

[6]En inglés, Fault prevention

2.4.3. Eliminación de Fallas

La FR hace referencia a las técnicas utilizadas para mejorar la fiabilidad empleadas durante la validación y verificación del SW (Pullum, 2001). Estas técnicas mejoran la fiabilidad del SW mediante la detección de fallas, usando métodos de verificación y la validación, y eliminando las fallas que se van detectando (Pullum, 2001).

Por otro lado Dubrova (2013) indica que el FR se lleva a cabo durante fases de desarollo de SW tanto como durante el ciclo de vida de un sistema. Durante la fase de desarrollo, FR consiste en tres pasos: *verificación, diagnóstico y correción* (Dubrova, 2013). FR durante la vida operacional de un sistema, consiste en el mantenimiento preventivo y correctivo del mismo (Dubrova, 2013).

2.4.4. Predicción de Fallas

La FF se realiza mediante la realización de una evaluación del comportamiento del sitema con respecto a la ocurrencia o la activación de una falla (Dubrova, 2013).Esta evaluación puede ser:

- Cualitativa: que tiene como objetivo clasificar los modos de fallas o combinaciones de eventos que llevan al sistema al fracaso (Dubrova, 2013).
- Cuantitativa: que tiene como objetivo evaluar en término de probabilidad, el grado en el cual los atributos de fiabilidad son satisfechos (Dubrova, 2013).

FF incluye técnicas para aumentar la fiabilidad del sistema que son usados durante la validación del SW, con el objetivo de estimar la presencia de fallas y la ocurrencia o consecuencia de fracasos (Pullum, 2001)

2.5. Atributos de la fiabilidad

El objetivo final de la FT es desarrollar un sistema que sea fiable. Fiabilidad tiene muchas definciones, pero comúnmente es expresado como la probabilidad de **no fallar** (Hitt y Mulcare, 2070). “La fiabilidad es la probabilidad de que un sistema continúe funcionando correctamente durante un intervalo de tiempo particular” (Torres-Pomales, 2000).

La fiabilidad de un sistema SW puede ser descrita por una serie de atributos, los cuales son mencionados a continuación.

2.5.1. Confiabilidad

La confiabilidad es la probabilidad de que un sistema continua operando correctamente durante un intervalo de tiempo dado (Torres-Pomales, 2000).

Dubrova (2013) coincide que la confiabilidad *R(t)*[7] de un sistema es la probabilidad de que el sistema opere sin fracasos en el intervalo de tiempo *[0,t]*.

La confiabilidad es una medida de la entrega correcta del servicio que brinda un sistema (Dubrova, 2013).

[7] En inlgés, Reliability

En sistemas críticos como el SW de vehículo espacial, es sumamente necesario que tenga una alta tasa de confiabilidad, ya que por ejemplo, perder el contacto con la nave, podría representar la pérdida de la misión, o una gran cantidad de datos. Otro ejemplo que se puede mencionar es de un satélite geoestacionario de comunicación, la pérdida de este servicio debe ser baja, casi nula (idealmente).

Coincidiendo Pressman (2001), define la confiabilidad como la "probabilidad de tener operaciones libre de fallas de un programa de computadora, en un ambiente específico para un tiempo específico". El mismo autor también indica que la confiabilidad es la Tiempo Medio Entre Fallas (MTBF)[8]. Donde:

$$MTBF = MTTF + MTTR$$

Tiempo Medio de Fallas (MTTF) es el promedio de tiempo desde que empieza la operación del sistema hasta el tiempo que se produce la primera falla. Tiempo Medio de Reperación (MTTR) es el promedio de tiempo que se requiere para recuperarse, después de un fracaso, al correcto funcionamiento (Hanmer, 2007). MTBF es similar a MTTF, lo único que los diferencia es que MTBF es la suma de MTTF y MTTR. Según Hanmer (2007) MTBF es utilizado para aquellos sistemas que son reperables. Para el caso contrario se utiliza MTTF.

La IEEE (1990) define confiabilidad como "La capacidad del sistema o componente de realizar sus funciones requeridas bajo las condiciones establecidas durante un período de tiempo especificado".

2.5.2. Disponibilidad

Es la probabilidad de que el sistema esté operando correctamente en un determinado instante de tiempo (Torres-Pomales, 2000). La disponibilidad *A(t)* de un sistema en el instante de tiempo *t* es la probabilidad que el sistema esté funcionando correctamente en el instante *t* (Dubrova, 2013).

Dubrova (2013) realiza un defínción matemática de *A(t)*, llamandola también como, *punto de disponibilidad* o *disponibilidad instantanea*. Y la define como:

$$A(T) = \frac{1}{T}\int_0^T A(t)\,\mathrm{d}x$$

Para Hanmer (2007) la disponibilidad del sistema es el porcentaje de tiempo en el que es capaz de llevar a cabo una función determinada.

Para el caso de los sistemas que no pueden ser reperados el punto de disponibilidad es igual a la confiabilidad del sistema (Dubrova, 2013).

Los estados de disponibilidad pueden ser representados en términos de fuera de servicio por año. En la Tabla 2.1 que expone Dubrova (2013) se puede observar esta relación.

La IEEE (1990) define la disponibilidad como "El grado en el cual un sistema o componente se encuentra operativo y accesible cuando se requiere su uso. También es expresado en términos de probabilidad"

En satélites de órbita baja (LEO[9]), es necesario que el satélite se encuentre disponible al momento de su pasada por las estaciones terreras para poder descargar los datos que se fueron almacenando. Del mismo modo, el satélite debe estar disponible para poder realizar las funciones necesarias para poder cumplir con su misión (como por ejemplo la registración de imágenes en una determinada zona terrestre).

[8] Del inglés, mean-time-between-failure
[9] Del inglés, Low Earth Orbit

Disponibilidad	Fuera de servicio
90 %	36.5 días/año
99 %	3.65 días/año
99.9 %	8.76 horas/año
99.99 %	52 minutos/año
99.999 %	5 minutos/año
99.9999 %	31 segundos/año

Tabla 2.1: Disponibilidad en relación con su baja de servicio por año. Tabla modificada de Dubrova (2013)

Diferente es el caso para los satélites geoestacionarios, ya que estos deberían estar disponible la mayor parte del tiempo, ya que en la mayoría de los casos son de comunicación.

2.5.3. Seguridad

La seguridad se considera como una extensión de la confiabilidad (Dubrova, 2013). Seguridad *S(t)* es definidad como la probabilidad que el sistema sea capaz de realizar su función correctamente o discontinuar su función en una manera a prueba de fallas (Dubrova, 2013).

Según Torres-Pomales (2000) la seguridad es la probabilidad de que el sistema levará a cabo sus tareas de una manera no peligrosa. Un peligro se lo puede definir como "un estado o condición de un sistema, que juntos con otras condiciones ambientales de el sistema, conducirá inevitablemente a un accidente" (Torres-Pomales, 2000).

La seguridad es requerida para aquellas aplicaciones de seguridad crítica donde un fracaso puede resultar en lesiones humanas, pérdidas de vida o desastres ambientales (Dubrova, 2013).

Para satélites es importante que se tenga una alta seguridad, ya que una pérdida de una misión representa grandes cantidades de dinero perdido.

2.6. Tolerancia a falla

En sistemas críticos, como el de una planta nuclear, sistema médico, el sistema de vuelo de un avión, o el de un satélite, el SW (ni el hardware) deben fallar, ya que esto daría como resultado la pérdida de muchas vidas. Para el caso particular, del vehículo espacial (satélite, transbordador, lanzador), la falla del SW podría tener como consecuencia la pérdida de una misión, y/o una gran cantidad de dinero, y hasta vidas en algunos casos (vuelos tripulados). La principal diferencia entre el SW de una misión satelital, con la de un avión o una planta nuclear, o un sistéma médico, es que ante alguna falla o error, se torna complicado llegar hasta el satélite para realizar una actualización o cargar un parche de SW.

La IEEE (1990) define como SW crítico a "aquel cuyo fracaso puede tener un impacto en la seguridad, o puede causar grandes pérdidas financieras o sociales". El SW de estos sistemas críticos deben tener la capacidad de seguir funcionando, aún en la presencia de fallas, o errores. Imaginese el caso, de un avión comercial, con pasajeros a bordo, y de repente ocurre un problema debido al mal diseño del SW (por ejemplo un overflow de memoria). En esta situación es impensable que el SW se congele y que el piloto reinicie el sistema, esperar que se reestablezca al estado en el cual se encontraba antes del problema, para seguir funcionando. Lo mismo ocurre con el SW de naves espaciales, hay situaciones en la que no se puede esperar y es preferible que el sistema siga funcionando aún en la presencia de fallas.

Tal lo como indica Pressman (2001) las fallas de SW implica problemas cualitativos que son descubiertos después de que el SW es llevado a los usuarios y probados por ellos. Una gran cantidad de estudios indican que en las actividades de diseño se introducen entre un 50 y 65 porciento de errores del total de errores que se dan durante el proceso del SW (Pressman, 2001). Esto no debe ocurrir en el ámbito espacial, ya que una vez que el sistema es utilizado, es muy dificil corregir los errores que surgen

Cabe aclarar que el SW al no ser un componente físico, no puede ser tratado de la misma manera que un componenete hardware. Como ejemplifica Torres-Pomales (2000), las fallas que surgen a nivel de bit, como por ejemplo en un disco duro, son fallas del dispositivo de almacenamiento y pueden ser mitigadas con la aplicación de técnicas de redundancias. Esto no es así para el SW. Por lo tanto evitar los errores en el a nivel de SW no es tan trivial como en el hardware.

A nivel de SW las fallas son llamadas "bugs" (tal como se indica en la sección 2.1 en la página 2), y existe un solo tipo de fallas que es introducido durante el desarrollo del SW (Torres-Pomales, 2000). Las fallas en el SW son el principal motivo de que todo un sistema fracase.

La FT, puede ser utilizada como una capa más de protección (Torres-Pomales, 2000). Esta aplicada al SW se refiere al uso de técnicas que permiten seguir brindando el servicio en un nivel aceptable de perfomance y seguridad después que una falla de diseño ocurra.

Debe hacerse una diferencia entre FT y calidad. Hanmer (2007) lo define de la siguiente manera: "FT es la capacidad del sistema a ejecutarse apropiadamente a pesar de la presencia de fallas. FT ocurre en tiempo de ejecución". Cuando se habla que un sistema es tolerante a fallas, signfica que fue diseñado de tal manera, que puede seguir funcionando correctamente aún en la presencia de errores de sistemas (Hanmer, 2007).

En cambio calidad, tal como lo define Hanmer (2007), "se refiere a cuán libre de fallas está el sistema. Técnicas de calidad que indican cómo el SW es creado. Si el sistema fue testeado."

Un sistema de alta calidad tendrá menor número de fallas, que esto representa menor número de fallas en tiempo de ejecución. La reducción del número de fallas no implica que los resultados de los defectos son menos severos (Hanmer, 2007). El sistema debe tomar medidas para reducir el impacto de los errores y fallas, y es allí donde surge la FT.

Un sistema tolerante a fallas provee una continua y segura operación, aún durante la presencia de fallas. Un sistema tolerante a fallas, es un elemento crítico para una arquitectura de vuelo, lo cual incluye hardware, SW, timing, sensores y sus interfaces, actuadores, elementos y datos de comunicación con los diferenes elementos (Hitt y Mulcare, 2070).

Este tipo de sistemas debería detectar los errores causados por fallas, evaluar los daños producidos por la falla, aislar a la misma y por último recueperarse, en ese caso se habla de aquitectura o sistemas FDIR[10].

FT es la capacidad de un sistema a continuar funcionando a pesar de la ocurrencia de fallas (Dubrova, 2013). Un sistema tolerante a fallas debe ser capaz de manejar fallas tanto de hardware como de SW. La FT es necesaria debido a que es imposible construir un sistema perfecto.

El objetivo de la FT es el desarrollo de sistemas los cuales funcionen correctamente en presencia de fallas (Dubrova, 2013). La FT es alcanzada mediate la utilización de algunos tipos de redundancias (Dubrova, 2013). *Redundancia* es la provisión de capacidades funcionales que sería innecesario para enternos libres de fallos (Dubrova, 2013). Esto significa tener hardware adicionales, check bits en una cadena de datos, o algunas líneas de código que verifica el correcto resultado del SW. La redundancia permite enmascarar una falla, o detectarla, para luego localizarla, contenerla y recuperarse de esta (Dubrova, 2013). Las técnicas de tolerancia de fallas se emplean durante la adquisición, o desarrollo del SW. Permite al SW tolerar fallas después que este haya sido desarrollado (Pullum, 2001). Cuando una falla se da, las técnicas de FT proveen mecanismos al sistema de SW

[10]FDIR, del inglés: Failure detect, isolate and recover

para prevenir el fracaso del sistema (Pullum, 2001).

2.7. Clasificación de un sistema de control tolerante a fallas

2.8. Redundancia en el software

A pesar de lo comentado anteriormente, sobre la importancia del SW, todavía existe una creencia, de que el SW aparece por arte de magia, y que los programadores no son nunca, lo suficientemente capaces, de hacer un SW libre de errores. Salvo aquellas empresas u organizaciones que tienen un proceso maduro de desarrollo de SW, el resto cae en el error de pensamiento mencionado anteriomente.

Dubrova (2013) explica que la FT aplicado en el SW no está tan entendido, ni maduro, como es en el caso de la FT aplicada en hardware. Si una falla exisitiera en el SW, esta se haría "visible", solo cuando las condiciones relevantes ocurran (Dubrova, 2013). Y muchas veces por tiempo o costo, no se realizan los tests cubriendo todos los posibles ambientes reales, lo cual tiene consecuencias desastrosas, tal como se expone en la sección 1.1 (página).

Para sistemas complejos o grandes, donde existe una gran cantidad de estados, implica que solo una pequeña porción del SW puede ser verifcada correctamente (Dubrova, 2013). Los tests tradicionales y métodos de depuración actuales, no alcanzan para grandes sistemas (Dubrova, 2013). La utilización de métodos formales para describir las característica requeridas por el comportamiento del SW, exigen gran complejidad computacional, y solo son aplicables en ciertas situaciones (Dubrova, 2013).

Las técnicas de FT pueden dividir en dos grupos:

- técnicas de una sola versión, se utilizan cuando existe una sola versión del SW en el sistema.
- técnicas multi-versión, se utilizan cuando se desarrollan varias versiones de una misma función.

Estas se explican en las siguientes secciones.

2.8.1. Técnicas single version

Estas técnicas son utilizadas para tolerar parcialmente las fallas del diseño de SW (Pullum, 2001). Técnicas single-version de FT se basa en el uso de redundancia aplicada a una única versión de una pieza de SW para detectar y recuperarse de fallas (Torres-Pomales, 2000).

Estas técnicas a los software que cuentan con una sola versión, un número capacidades funcionales que no serían necesarias dentro de un ambiente libre de fallas (Dubrova, 2013).

2.8.1.1. Estructuras de software

En Torres-Pomales (2000) se mencionan dos técnicas de estructuración del SW que son muy buenas a la hora de mantener FT en el SW.

La defincón de una arquitectura en el software es de suma importancia ya que proveen las bases para la implementación de FT (Torres-Pomales, 2000). Una de las técnicas utilizadas en el desarollo del software es la modularización. Esta consiste en descomponer el problema en componentes manejables. Esto tiene como resultado que sea más eficiente la aplicación de la FT en el diseño de un sistema (Torres-Pomales, 2000).

El particionado es otra técnica mencionada en Torres-Pomales (2000), lo cual provee aislación entre módulos independentes del sistema. Esta técnica permite descomponer al problema en partes separadas (Pressman, 2001). El particionado puede ser horizontal u vertical. En el primero se descomponen el problema moviendose en forma horizontal en la jerarquía, mientras que el segundo se parte de lo más general hasta llegar a lo detallado, moviendose verticalmente en la jerarquía (Pressman, 2001).

Sistema de cierre es un principio de FT, en el cual ninguna acción es permitible sin una autorización expresa (Torres-Pomales, 2000). Siguiendo este principio ninguna de las funciones que componen al sistema deberían tener más capacidad de la necesaria (Torres-Pomales, 2000). Las ventajas de desarrollar un sistema bajo este pricipio, es que es sencillo el manejo de errores, y evitar la propagación de fallas si ocurriesen.

2.8.2. Técnicas multi-version

Las técnicas de multi-version utilizan dos o más versiones diferentes del mismo módulo de SW (Dubrova, 2013) (Torres-Pomales, 2000), lo cual satisface el requerimiento de diversidad.

El objetivo de utilizar diferentes versiones de SW es que es construido de diferentes maneras, por lo tanto fallarían de diferente maneras (Torres-Pomales, 2000).

2.8.3. Técnicas de detección de fallas

Para los SW tolerantes a fallas, de una sola versión, se suelen utilizar varios tests de "aceptación" para detectar fallas (Dubrova, 2013). Es necesario que estos SW cuenten con dos propiedades: auto protección[11] y auto check[12] (Torres-Pomales, 2000). La auto protección significa que los componentes de sistema tienen la capacidad de protegerse así mismo mediante la detección de errores (Torres-Pomales, 2000). La propiedad de auto check significa que los componente son capaces de detectar fallas internas y tomar las acciones necesarias para evitar la propagación del error.

El resultado del sistema depende del resultado de los tests. Si el resultado pasa exitosamente el test, este es el correcto, caso contrario significa la presencia de fallas (Dubrova, 2013). Un test es más efectivo si se puede calcular de una manera simple (Dubrova, 2013).

Las técnicas utilizadas son las siguientes:

- *Timing checks:* se agrega a los sistemas una restricción de tiempo. Basado en esa restricción se puede deducir si el comportamiento del sistema se desvió (Dubrova, 2013). Los más utilizado es el *watchdog timer*, este es un contador, actualizado con un *timer* que detecta si un módulo de SW se bloqueó o congeló, entonces se reincia ese módulo o el sistema.
- *Coding checks:* se utiliza en los sistemas donde los datos se codifican usando técnicas de redundancia de datos (Dubrova, 2013).
- *Reversal checks:* son aquellos donde se toma los valores de salida, y con ellos se busca encontrar cuáles fueron los datos de entrada. Si los datos de entrada reales coinciden con los calculados (para una misma salida), este se encuentra libre de fallas (Dubrova, 2013).
- *Reasonableness checks:* usa propiedades semánticas en los datos para detectar fallas (Dubrova, 2013).

[11] En inglés, self-protection
[12] En inglés, self-checking

- *Structural checks:* se basa en el conocimiento de las propiedades de la estructura de datos (Dubrova, 2013).
- *Replication checks:* se basa en la comparación de resultados de varios componentes (Torres-Pomales, 2000).

Se suelen utilizar árboles de fallas, como una técnica auxiliar en el desarrollo de sistemas para la detección de fallas (Torres-Pomales, 2000). El árbol de falla permite obtener un enfoque top-down de las diferentes fallas que se pueden dar. El árbol no cubre todas las fallas que puedan darse, pero si ayudan en un alto grado en el desarrollo de SW tolerante a fallas (Torres-Pomales, 2000).

2.8.4. Técnicas de recuperación de fallas

Una vez que la falla es detectada, el sistema debe proceder a recuperarse de aquella, y volver a un estado operacional normal (Dubrova, 2013). Si los mecanismos de detecćión y contención de fallas fueron desarrollados correctamente, esta es contenida dentro de un set de módulos en el momento de la detección (Dubrova, 2013).

2.8.4.1. Manejo de excepciones

En muchos SW y lenguajes de programación, se logra recuperarse mediante el manejo de excepciones. El manejo de excepciones es la interrupción del funcionamiento normal para responder a un funcionamiento anormal del sistema (Torres-Pomales, 2000). Los posibles eventos que pueden lanzar una excepción son:

1. Excepciones de interfaces, son lanzadas por un módulo cuando se da una solicitud inválida de algún servicio (Dubrova, 2013).
2. Excepciones locales, son lanzadas por algún módulo cuando sus propios mecanismos de detección de fallas encuentran un problema interno (Dubrova, 2013).
3. Excepciones de fracaso, son lanzadas cuando un mecanismos de detección encuentra una falla, per es imposible recueperarse de esa falta (Dubrova, 2013).

2.8.4.2. Checkpoint y Restart

Para los software de una sola versión existen pocos mecanismos de recuperación. Checkpoin y restart es uno de ellos. También es conocido como *backward error recovery* (Dubrova, 2013). La mayoría de las fallas que se dan en los SW son debido a fallas que provienen del diseño, tal como se mencionó anteriomente. Estas fallas son activadas por entradas al sistema (Dubrova, 2013).

Este mecanismo cuenta con el módulo principal que se encuentra en ejecución combinado con un bloque que realiza tests de aceptación. Si se detecta una falla, en el bloque de testeo, se envía una señal de “reincio”, para que el módulo principal vuelva al estado anterior, es decir, antes de producirse el error. Este estado anterior se encuentra almacenado en una memoria checkpoint (Dubrova, 2013). En la figura 2.2[13], se muestra la representación de este mecanismo.

Existe dos tipos de checkpoints, estáticos y dinámicos. Los checkpoints dinámicos toman una “fotografía” del estado del sistema antes de comenzar la ejecución del SW y lo guarda en memoria (Dubrova, 2013). Si

[13] Basado en Dubrova (2013) y Torres-Pomales (2000)

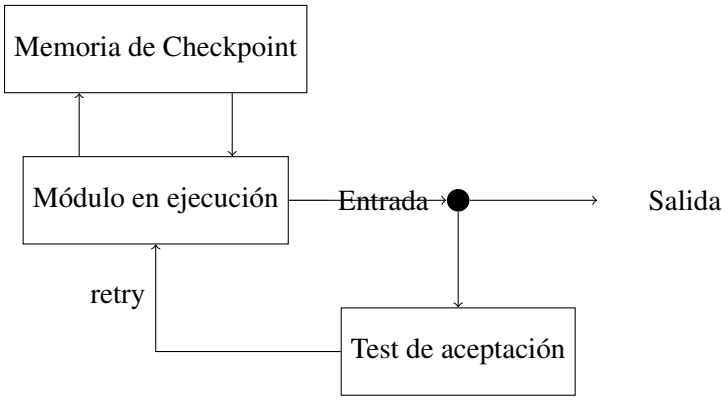

Figura 2.2: Representación de checkpoint y restart

se detecta una falla, el sistema regresa a ese estado y comienza de nuevo su ejecución (Dubrova, 2013). Los checkpoints estáticos se basan en regresar el módulo a un estado predeterminado (Torres-Pomales, 2000). Se puede regresar a un estado inicial o a un set de estados predeterminados (Torres-Pomales, 2000).

Por otro lado se encuentran los checkpoints dinámicos. Estos usan checkpoints creados dinámicamente. Estas son imágenes del estado del sistema en varios puntos durante la ejecución (Torres-Pomales, 2000).

Hay tres formas de crear los checkpoints dinámicamente:

1. Equidistantes, en el cual los intervalos que se crean los checkpoints son siempre iguales, los intervalos se elijen teniendo en cuenta el rate de falla (Dubrova, 2013).
2. Modular, en el cual los checkpoints se crean al principio o al final de la ejecución de un módulo.
3. Random, los checkpoints se crean aleatoriamente en el tiempo.

2.8.4.3. Procesos pares

Los procesos pare utilizan dos versiones identicas de un proceso de SW que corre en procesadores separados (Dubrova, 2013) (Torres-Pomales, 2000). El mecanismo de recuperación que se utiliza es el de checkpoint y restart (Torres-Pomales, 2000).

Como se puede observar en la figura 2.4[14] el primer procesador se encuentra activo. Este envía un checkpoint al segundo procesador. Si una falla se detecta, el primer procesador se apaga y se cambia al segundo procesador. El segundo procesador carga el checkpoint y continua con la operación.Toma el rol del primer procesador (Torres-Pomales, 2000). Luego el primer procesador realiza un auto test para verificar si el problema continua. Si se encuentra que este procesador sigue teniendo problema, se continúa trabajando con el segundo procesador (Dubrova, 2013).

La principal ventaja que brinda este mecanismo según Dubrova (2013) es que permite entregar el servicio ininterrumpidamente.

2.8.4.4. Diversidad de datos

La diversidad es una técnica utilizada para mejorar la eficiencia en los checkpoint y restart, usando diferentes entradas por cada reinicio (Dubrova, 2013). Esto se basa en que las fallas en el SW son dependientes de

[14] Basada en Dubrova (2013) y Torres-Pomales (2000)

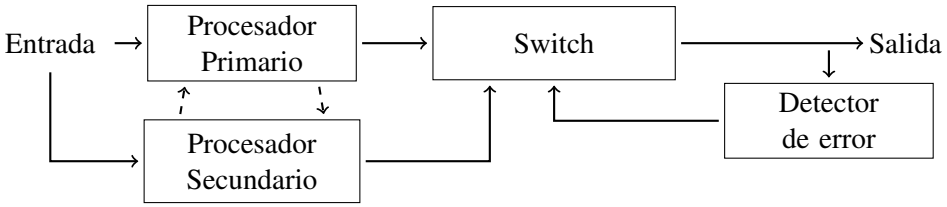

Figura 2.3: Representación del proceso pares

las entradas (Dubrova, 2013). Es poco probable que la misma falla se de con la misma secuencia de entrada (Dubrova, 2013).

2.8.4.5. Bloques de recuperación

Esta técnica combina las bases de la técnica de checkpoints y restart enfocada con múltiples versiones de un componente de SW en el sentido de que una versión de SW diferente es lanzada cada vez que se encuentra una falla (Torres-Pomales, 2000). Los checkpoints son creados antes de que una versión de SW se ejecuta (Torres-Pomales, 2000). La ejecución de las múltiples versiones pueden ser secuencial o paralelas dependiendo de la disponibilidad de la capacidad de procesamiento y perfomance requerida (Torres-Pomales, 2000).

La representación de esta técnica se puede observar en el figura [AGREGAR IMAGEN]. Las versiones son diferentes implementaciones de un mimo programa. Solo uno de estas versiones provee la salida del sistema. Si un error es detectador por el test de aceptación, se vuelve hacia atrás, se retoma el último checkpoint, y se vuelve a ejecutar el módulo de SW pero con una versión diferente a la que se ejecutó anteriomente (Dubrova, 2013).

Los checks del test de aceptación deben mantenerse simples para mantener la velocidad de la ejecución (Dubrova, 2013).

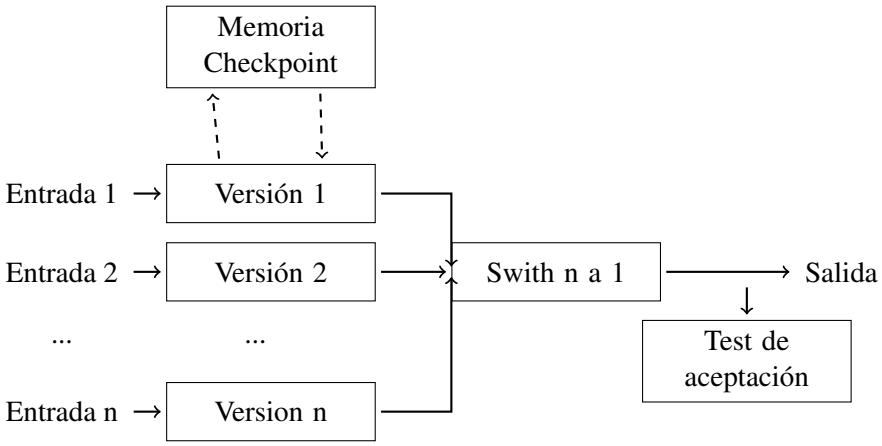

Figura 2.4: Configuración de bloques de recuperación

2.8.4.6. Programación N-version

2.8.4.7. Programación N-Auto Checking

2.9. Técnica de evaluación de fiabilidad

La evaluación de la fiabilidad es de suma importancia para el desarrollo de sistemas críticos, ya que permite identificar que aspectos del comportamiento del sistema juega un papel importante (Dubrova, 2013).

1. Modelado de un sistema en la fase de diseño.
2. Aseguramiento del sistema en la fases finales de desarrollo (testing).

El análisis confiabilidad tiene tres enfoques importantes:

- Confiabilidad de HW
- Confiabilidad de SW
- Confiabilidad humana

En este trabajo de tesis se pondrá énfasis en el estudio en la confiabilidad del SW a nivel de sistema.

La evaluación de la fiabilidad tiene dos aspectos. En primer lugar se tiene una *evaluación cualitativa* que permite identificar, clasificar y medir modos de fallas, o eventos combinacionales que puedan provocar una falla. El otro aspecto es la *evaluación cuantitativa*, la cual permite evaluar en términos de probabilidad los atributos de la fiabilidad (Sección 2.5), disponibilidad, seguridad.

El análisis de confiabilidad es de gran importancia ya que provee información que es la base de la toma de desición. Esto es aplicado a diferentes áreas, tales como análisis de riesgos, protección ambiental, calidad, optimización de mantenimientos y operaciones y diseño de ingeniería (Rausand y Hoyland, 2004).

2.10. Medidas comunes de fiabilidad

Las medidas de fiabilidad más comunes son las siguientes: failure rate, tiempo medio a la falla, tiempo medio de reparación y tiempo medio entre fallas.

2.10.1. Failure rate

Failure rate λ es el número esperardo de fallas por unidad de tiemp (Dubrova, 2013). Es usual utilizar la dimensión *fallas/horas*.

Generalmente, λ se encuentra a nivel de componente. Para conocer el failure rate del sistema completo, se puede realizar (a groso modo) una sumatoria de los λ de los componentes que integran el sistema.

$$\lambda = \sum_{i=1}^{n} \lambda_i$$

.

La evolución de λ a través del tiempo, no tiene el mismo comportamiento tanto para HW como para SW Si se devide el ciclo de vida de un sistema en las siguientes fases: mortalidad prematura (I), vida útil (II), desgaste (II) (Dubrova, 2013) se aprecia, para el caso del HW, lo que se denomina *curva de la bañera* la cual puede observarse en la Figura 2.5. En una primera fase, λ decrese, ya que a través de los procesos de testing se van descubriendo y resolviendo los errores. Luego se da un periodo de estabilización. Y al final, el HW sufre el paso del tiempo, y se desgasta, aumentando la tasa de fallas.

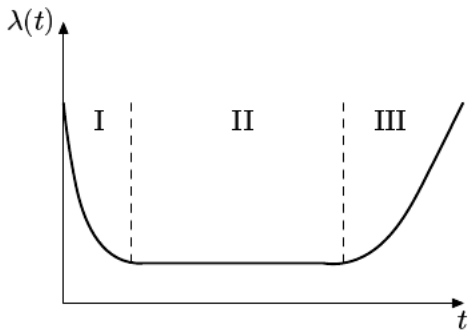

Figura 2.5: Failure rate de HW vs tiempo

Para el SW es totalmente diferente. En primer lugar cuando se realiza una actualización, se aumenta la complejidad, como así también la probabilidad de fallas, con ello el failure rate. Otra diferencia sustancia con el HW es que el SW no se desgasta con el tiempo. En la Figura 2.6 se aprecia a través de tiempo. Esta curva suele llamarse curva serrucho. El failure rate del SW decrece en función del tiempo. En estos tipo de sistemas la tasa de falla depende de varios factores como pueden ser el proceso utilizado en el diseño y codificación, complejidad del SW, tamaño del SW,etc (Dubrova, 2013).

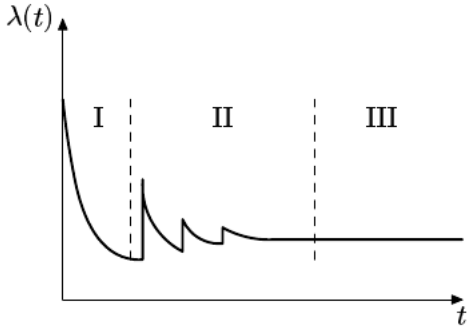

Figura 2.6: Failure rate SW vs tiempo

A lo largo de la vida de sistema se supone el failure rate λ como constante. Por lo tanto la confiabilidad del sistema varía exponencialmente con respecto al tiempo (Dubrova, 2013):

$$R(t) = e^{-\lambda t}$$

Esto se conoce como *ley de la falla exponencial*(Dubrova, 2013). El gráfico de confiabilidad $R(t)$ vs tiempo se muestra en la Figura 2.7.

Figura 2.7: Confiabilidad vs tiempo

2.10.2. Tiempo medio medio de falla

El tiempo medio de falla (MTTF[15]) de un sistema es el tiempo esperado que transcurra hasta la primera falla que se detecte en el sistema. En terminos de confiabilidad, MTTF se define de la siguiente manera (Dubrova, 2013) (Rausand y Hoyland, 2004)

$$\int_0^\infty R(t)dt$$

2.10.3. Tiempo medio de reparación

El tiempo medio de reparación (MTTR[16])de un sistema, es el promedio de tiempo que se requiere para reparar al sistema. MTTR se especifica en términos de la tasa de reparación μ (Dubrova, 2013) (Rausand y Hoyland, 2004), el cual es el número esperarado de reparaciones por unidad de tiempo:

$$MTTR = \frac{1}{\mu}$$

El MTTR depende de los mecanismos de recuperación ante fallas que se utilicen en el sistema, localización del sistema, scheduler de mantenimiento (Dubrova, 2013). Con esto se puede definir la disponibilidad como sigue:

$$A(\infty) = \frac{MTTF}{MTTF + MTTR}$$

Muchas veces se utiliza MDT ([17]) en vez de MTTR, para denotar más claro que es el tiempo medio que el sistema se encuentra fuera de sericio.

2.10.4. Tiempo medio entre fallas

El tiempo medio entre fallas (MTBF[18]) de un sistema es el tiempo promedio entre dos fallas del sistema.

$$MTBF = MTTF + MTTR$$

2.10.5. Cobertura de fallas

La cobertura de fallas es la probabilidad de que el sistema no interrumpirá su actividad cuando una falla se presente. En términos matemáticos la cobertura de fallas la probabilidad condicional $P(A|B)$. Existen diferentes coberturas de fallas, dependiendo de si se está tratando con detección de fallas, localización de fallas, contención de fallas o recuperación de fallas (Dubrova, 2013). Siendo A detección, localización, contensión o recuperación de fallas, y B la existencia de fallas.

2.11. Métodos de cálculos de fiabilidad

Para evaluar la fiabilidad de sistemas se pueden utilizar diagramas de bloque de confiabilidad y procesos de Markov (Dubrova, 2013)

[15] Del inglés, Mean Time To Failure
[16] Del inglés, Mean Time To Repair
[17] Del inglés, Meann Downtime
[18] Del inglés, Mean Time Between Failure

2.11.1. Diagramas de bloques de confiabilidad

2.11.1.1. Cálculo de confiabilidad

Para medir la confiabilidad de un sistema mediante diagrama de bloques, se debe dividir el sistema objetivo en partes paralelas y en serie. Se computa la confiabilidad de las partes. La confiabilidad del sistema estará compuesta por la confiabilidad de ambas partes (Dubrova, 2013). Entonces

$$R(t) = \begin{cases} \prod_{i=1}^{n} R_i(t) & \text{para estructuras en serie} \\ 1 - \prod_{i=1}^{n} (1 - R_i(t)) & \text{para estructuras en paralelo} \end{cases}$$

Esto nos indica que un sistema paralelo, es más confiable que uno en serie, aún así si sus componentes son menos confiables. Tal como ejemplifica Dubrova (2013), si se diseña un sistema en serie de 100 componentes, con una confiabilidad de 0.999, el sistema completo tendrá una confiabilidad de: $0{,}999^{1}00 = 0{,}905$. Mientras que, para un sistema paralelo, con solo cuatro componentes, con una confiabilidad menor (0.95) la confiabilidad del sistema será $1 - (0{,}95)^4 = 0{,}99999375$. El punto en contra de los sistemas paralelos, es que representan un costo mayor que las estrucutras en serie (Dubrova, 2013).

2.11.1.2. Cálculo de disponibilidad

Si se asume que el tiempo de falla y de recuperación son independientes, entonces se puede utilizar diagramas de bloques para calcular la disponibilidad del sistema (Dubrova, 2013). Se puede observar que el cálculo es similiar al calculo de confiabilidad.

$$A(t) = \begin{cases} \prod_{i=1}^{n} A_i(t) & \text{para estructuras en serie} \\ 1 - \prod_{i=1}^{n} (1 - A_i(t)) & \text{para estructuras en paralelo} \end{cases}$$

2.11.2. Utilización de procesos de Markov

El principal objetivo del análisis de los procesos de Markov es calcular $P_i(t)$ la probabilidad de que el sistema se encuentre en el estado i en el tiempo t. Con esto se puede calcular facilmente la confiabilidad, disponibilidad o seguridad del sistema (Dubrova, 2013).

Para determinar $P_i(t)$ se debe derivar una serie de ecuaciones diferenciales, una por cada estado del sistema. Estas ecuaciones se denominan ecuaciones de estado de transición. Las ecuaciones de los estados de transición se representan en una matriz M denominada *matriz de transición*. Cada elemento m_{ij} de la matriz M es un rate de transición entre los estados i y j

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{k1} \\ m_{12} & m_{22} & \dots & m_{k2} \\ & & \dots & \\ m_{1k} & m_{2k} & \dots & m_{kk} \end{bmatrix}$$

Haciendo $P(t)$ un vector en el cual el $i-simo$ elemento es la probabilidad $P_i(t)$ de que el sistema se encuentra en el estado i en el tiempo t. Con ello se tiene:

$$\frac{d}{dt}P(t) = MP(t)$$

(Dubrova, 2013).

Para calcular confiabilidad, disponibilidad y seguridad solo es necesario reemplazar en la matriz M los rate correspondientes.

2.12. Modelos de falla

En esta sección se explica con un poco más de detalle algunos conceptos que se presentaron en secciones anteriores. Los conceptos que se detallarán son los siguientes:

- Función de confiabilidad $R(t)$
- Función de tasa de falla λ
- Tiempo medio hasta la falla (MTTF)
- Vida residual media (MRL)

También existen varias distribuciones de probabilidad que son utilizados para modelar el tiempo de vida de componentes de sistemas. Algunos de estos modelos de distribución del tiempo de vida son los siguientes:

- Distribución binomial
- Distribución exponencial
- Distribución gamma
- Distribución Weibull
- Distribución normal
- Distribución Birnbaum-Saunders
- Distribución inversa de Gauss

2.12.1. Tiempo hasta la falla

El tiempo hasta la falla (T) de un componente del sistema, es el lapso de tiempo desde que el componente empieza a operar hasta la primera falla que se produzca (Rausand y Hoyland, 2004).

Nota: Se asume que el componente que falla, no puede ser recuperado. Por otro lado, si un sistema falla, puede recuperarse (a través de tecnicas de reconfiguración, componentes back-up).

T no significa que es solo una medida de tiempo. T puede significar (Rausand y Hoyland, 2004):

- Cantidad de kilómetros de un automóvil
- Número de rotación de un motor
- Número de ciclos de trabajo

Debe destacarse que esta no solo es una variable discreta. Una variable discreta puede ser aproximada a una variable continua. Suponiendo que T es una distribución continua con densidad de probabilidad $f(t)$ y una función de distribución:

$$F(t) = P(T) = \int_0^t f(u)du \text{para t} > 0$$

$F(t)$ demuestra la probabilidad de que un componente falle dentro de un intervalo de tiempo $(0,t]$ (Rausand y Hoyland, 2004).

La función de densidad de probabilidad $f(t)$ se define como sigue (Rausand y Hoyland, 2004):

$$f(t) = \frac{d}{dt}F(t) = \lim_{\Delta t->0} \frac{F(t+\Delta t) - F(t)}{\Delta t} = \lim_{\Delta t->0} \frac{P(t < T \leq t+\Delta t)}{\Delta t}$$

Esto indica que para un Δt pequeño,

$$P(t < T \leq t+\Delta t) \approx f(t)\Delta t$$

2.12.2. Función de confiabilidad

La función de confiabilidad se define como sigue:

$$R(t) = 1 - F(t) = P(T > t)$$

para $t > 0$.

2.12.3. Tasa de falla

La probabilidad de que un componente falle en un intervalo de tiempo $(t, t+\Delta t]$ dado que el componente ha funcionando hasta t, se tiene:

$$P(t < T \leq t+\Delta t | T > t) = \frac{P(t < T \leq t+\Delta)}{P(T > t)} = \frac{F(t+\Delta t) - F(t)}{R(t)}$$

Si se divide esta probabilidad por un intervalo de tiempo Δt, y haciendo $\Delta t - > 0$, se obtiene la función de tasa de falla.

2.12.4. Tiempo medio hasta la falla

El MTTF de un componente está definido por (Dubrova, 2013) Rausand y Hoyland (2004):

$$MTTF = E(T) = \int_0^\infty t f(t) dt$$

Se demuestra en Rausand y Hoyland (2004) que

$$MTTF = \int_0^\infty R(t) dt$$

2.12.5. Vida restante media

Considerando un comoponente con tiempo hasta la falla T, que es colocado en opración en el tiempo $t = 0$ y continúa funcionando en el instante t. La probabilidad de que el componente en de edad t sobrevive en un intervalo x, entonces se tiene (Rausand y Hoyland, 2004):

$$R(x|t) = P(T > x+t | T > t) = \frac{P(T > x+t)}{P(T > t)} = \frac{R(x+t)}{R(t)}$$

Entonces:

$$MRL(t) = \frac{1}{R(t)} \int_t^{\infty} R(x)dx$$

Nótese que cuando t = 0, el componente es nuevo, y entonces $\mu(0) = \mu = MTTF$.

2.12.6. Distribución binomial

La distribución binomial es uno de lo más utilizados (Rausand y Hoyland, 2004). Esta distribución es utilizada en los siguientes casos:

- Cuando se tienen n ensayos independientes
- Cada ensayo tiene dos posibles resultaods
- La P(A) = p es la misma en todos los ensayos

Entonces

$$P(X = x) = \binom{n}{x} \cdot p^x (1-p)^{n-x}$$

Donde $\binom{n}{x}$ es el coeficiente binomial y X es el número de n ensayos para alcanzar un resultado A (Rausand y Hoyland, 2004).

2.12.7. Distribución exponencial

Considerando un componente que entra en operación en el instante $t = 0$. El tiempo hasta la falla T del componente tiene una función de densidad de probabilidad de la siguiente forma Rausand y Hoyland (2004):

$$f(t) = \begin{cases} \lambda e^{\lambda t} & \text{para t >0, } \lambda > 0 \\ 0 & \text{para cualquier otro caso} \end{cases}$$

Esta distribución es la que se denomina distribución exponencial con parámetro λ.

La confiabilidad de esta función es:

$$R(t) = P(T > t) = \int_t^{\infty} f(u)du = e^{\lambda t} \text{para t>0}$$

El MTTF es:

$$MTTF = \int_0^{\infty} R(t)dt = \int_0^{\infty} e^{\lambda t} dt = \frac{1}{\lambda}$$

La tasa de falla de esta distribución es λ

La distribución exponencial es lo que se utiliza más comúnmente en los análisis de confiabilidad.

El MTTF es:

$$MTTF = \int_0^{\infty} R(t)dt = \int_0^{\infty} e^{-t} = \frac{1}{\lambda}$$

Si hacemos:

$$R(MTTF) = R(\frac{1}{\lambda}) = e^{-1} \approx 0{,}3679$$

se puede calcular la probabilidad de que un componente sobreviva durante el tiempo medio hasta la falla (Rausand y Hoyland, 2004).

Mientras que la tasa de falla es:

$$z(t) = \frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$

NOTA IMPORTANTE: Como se mencionó anteriormente, existen numerosas distribuciones de probabilidad para medir el tiempo de vida de los componentes. En este trabajo de tesis, se trabaja únicamente con la distribución exponencial, debido a que es una de las más sencillas,y la más utilizada en la bibliografía.

2.13. Protocolos de comunicación de tiempo real

2.13.1. Sistemas de tiempo real

Un sistema de tiempo real es un sistema de computadora que depende del correcto comportamiento funcional con respecto al dominio del tiempo, esto signfica que el sistema debe llevar a cabo sus funciones y obtener como resultados (correctos) dentro de las restricciones de tiempo (Lisner, 2007).

Los sistemas de tiempo real se dividen en *sistemas de tiempo real "soft"* y *sistemas de tiempo real "hard"*. En los soft real-time systems es importante cumplir con los tiempos del planificador, pero el no cumplimiento no tiene un impacto en la seguridad del sistema. Por otro lado, en los hard real-time systems, el no cumplimiento de las restricciones del tiempo, tiene como consecuencia un impacto catastrófico en el sistema.

Existen numerosos protocolos de comunicación, la mayoría abocados en la implementación de la capa 1 y 2 del modelo de ISO/OSI. Estos protocolos especifican las restricciones de hardware, la topología de red, la arquitectura del nodo, acceso al medio, los mecanismos de detección de error, etc. (Lisner, 2007). En los sistemas críticos como en el caso de los vehículos satelitales, es necesario la aplicación de sistemas de tiempo real, que garantice una máxima latencia y un comportamiento determinístico.

La tolerancia a fallas aplicadas en protocolos de comunicación de tiempo real, permiten que la red continúe funcionando aún cuando algunos de sus componentes fallaron. Los sistemas tolerantes a fallas son diseñados a partir de un modelo de falla dado (Lisner, 2007). El modelo de falla describe la estrucutra del sistema y los tipos de fallas que pueden ocurrir (Lisner, 2007).

Otro aspecto importante de los protocolos de comunicación tolerantes a fallas de tiempo real, es conocer el momento en el que un nodo está disponible para enviar mensajes. Esto es llevado a cabo por las estrategias de acceso al medio. Los métodos de acceso al medio pueden ser clasificados como event-triggered y time-triggered. En el primero, un mensaje es enviado, si algún nodo requiere ese mensaje. Este no es una buena opción para los sistemas de tiemp real, ya que no asegura los límites de tiempo (Lisner, 2007).

Los métodos time-triggered es hecho en ciclos. Los nodos tiene acceso al medio a través de intervalos periódicos de tiempo (Lisner, 2007). Una de las principales ventajas de este enfoque, es que es predecible, los intervalos son pre configurados. Como desventaja se puede mencionar que se hace un uso deficiente del medio, cuando algún nodo no tiene nada para enviar en su ventana de tiempo.

2.14. Estrategia de acceso al medio

2.14.1. CSMA

El esquema CSMA (Acceso Múltiple por Detección de Portadora[19]) proviene de ideas del Ethernet alámbrico. En esta técnica los nodos esperan durante un intervalo corto de tiempo y aleatorio antes de transmitir (Tanenbaum, 2003). Este método es utilizado comúnmente en redes inalámbricas. Al protocolo CSMA se la puede dividir en:

- CSMA con Detección de Pportadora
- CSMA con Detección de Colisiones

En la primera, el nodo intenta enviar un mensaje cuando el canal está inactivo. Si otro nodo lo está usando espera hasta que se desocupa y luego transmite. En el caso de una colisión de mensajes, espera un tiempo aleatorio antes de enviar el mensaje nuevamente. Esta tiene algunas desventajas, que no la hacen apropiadas para su aplicación en sistemas críticos. Para hacer frente a las desventajas del CSMA con detección de portadora, se utiliza CSMA/CD (CSMA con Detección).

Por otro lado, también existe otro método denominado CSMA/CA (CSMA con Evitación de Colisiones), utilizado en el 802.11. Este protocolo que es similar al CSMA/CD de Ethernet, con detección de canal antes de enviar y retroceso exponencial desopués de las colisiones (Tanenbaum, 2003). Esta estrategia además de ser utilizada en el Wireless, se utiliza en CAN.

2.14.2. TDMA

TDMA (time division multiple acccess) es una técnica de multiplexación del tiempo que es utilizada en redes inalámbricas, comunicación de satélites y en diferentes protocolos de tiempo real (Lisner, 2007).

Los accesos al medio se realiza mediante ciclos, en el cual se subdivide dentro de slots de tiempo de ancho estático. Solo un nodo está permitido enviar en un solo slot.

2.14.3. Minislotting

El uso de slots de tiempo con un ancho fijo y estático del tiempo, como es el que se utiliza en TDMA, puede convertirse en un problema. En el TDMA los nodos tiene que esperar todo un ciclo para poder enviar un mensaje. Además, el nodo debe enviar mensajes vacios (*null* durante su slots, cuando no tiene nada que enviar. Solución de este problema es el minislotting, ya que permite la utilización de slots con ancho variable. Esto logra recortar los tiempos de espera cuando no se está utilizando el medio (Lisner, 2007). Este método permite un uso más eficiente de los ciclos. La transmisión puede tener diferntes anchos, y no hay necesidad de enviar mensajes null. Este método se basa en la prioridad y para evitar la monopolización del medio, algunos protocolos como el ARIN 629, utiliza timeouts para denegar el acceso al medio después de un determinado tiempo (Lisner, 2007).

[19] Del inglés, Carrier Sense Multiple Access

2.15. Revisión de protocolos

2.15.1. CAN

CAN es un protocolo de comunicación desarrollado por Bosch para ser utilizado en automóviles. Existe un estándar de ISO para CAN. CAN es un protocolo del tipo event-triggered y utiliza CSMA/CA.

CAN utiliza diferentes mecanismos de detección de errores. Utiliza un CRC de 15 bits. Cada nodo puede enviar un mensaje de diagnóstico de errores, además de que lleva un contador de errores propios. Si este número de errores de grande, el mismo nodo entra en modo *error activo* (envía flags de "error activo"), *error pasivo*(envía flags de "error pasivo" y espera 8 bit times antes de repetir el envío del mensaje) y *bus off*(el nodo se apaga) (Lisner, 2007)

2.15.2. byteflight

Este se basa en el protocolo Minislotting y utilizado por BMW en automóviles. Este utiliza un clock master de alta presición para la sincronización.

Tanto CAN como byteflight, no poseen un mecanismo o técnica para proteger el canal de alguna falla de los nodos (Lisner, 2007).

2.15.3. ARINC 659 o SAFEbus

SAFEbus es una implementación del estandar ARINC 659 para su utilización en aviones comerciales. También se suele utilizar en vehículos espaciales. SAFEBus es una arquitectura que es altamente redundante.

2.15.4. TTP/C

TTP/C proviene de la familia TTP. Este protocolo utiliza el esquema TDMA en una arquitectura de doble canal. La configuración se encuentra pre definida. Esto permite al sistema ser prdecible, lo cual facilita la tolerancia a fallas. Su alto grado de tolerancia a fallas, permite desarrollar aplicaciones más confiables que otros protocolos (Lisner, 2007).

2.16. Posibles fallas en una red

Las principalles fallas que se pueden producir en una red son debidas al canal de comunicación y los nodos/controladores, esto se debe a que son los componentes más importantes de toda red (Lisner, 2007).

Las fallas en el nodo pueden ser múltiples tipos. El nodo puede codificar erroneamente un mensaje; se pueden dar errores en el timing, es decir actúan en tiempo equivocado; el nodo puede tener un comportamiento inusual y se bloquea el servicio que está brindando.

Asimismo, las fallas en el canal de comunicación pueden ser de diferentes tipos. El canal puede generar ruido, este ruido puede modificar un mensaje (hasta el momento un mensaje correcto); el mensaje no llega a los destinatarios.

También se pueden observar que tanto errores en el nodo como en el canal se produzcan simultaneamente (Lisner, 2007).

2.17. Protocolo CAN

2.17.1. Consideraciones previas

En es este trabajo de tesis se decidió continuar el desarrollo de la arquitectura con BUS-CAN ya que este es de interés para los proyectos desarrollados en INVAP. Cabe aclarar que el objetivo principal de esta tesis no es llevar a cabo una comparación exhaustiva de protocolos de comunicación tolerantes a fallas, sino elegir la más indicada y que se adapte a las necesidades de INVAP. Por tal motivo, se expone, en esta sección, las características importantes de CAN.

2.17.2. Introducción

El bus CAN (Controller Area Network) fue desarrollado por Bosh. Comenzó su desarrollo en 1983 y tuvo su primer release en 1986. Fue estandarizada por la Organización Internacional de Estandarización (ISO) bajo el nombre de ISO 11898. El Bus CAN surgió como respuesta al rápido crecimiento de la elctrónica en la industria automotriz (ESD Electronics, 2017). Este protocolo se definió con el objetivo de proveer comunicación determinística de sistemas distribuídos, y que necesiten un alto grado de confiabilidad. CAN permite la conectividad vía bus serial. El bus está compuesto por 2 cables que pueden estar blindados o on (ESD Electronics, 2017).

Las características de CAN que la convierten en un tentadora opción para la aplicación en el sector espacial, son:

- Bajo costo
- Operabilidad en ambientes eléctricos complicados
- capacidades de tiempo real.
- facilidad en el uso

La especificaciones de CAN, originalmente desarrollados por Bosch cubre solamente las capas físicas y de enlaces de datos del modelo de referencia de ISO/OSI. Luego de la estandarización ISO provee detalles adicionales de la capa física de CAN.

A diferencia de otros protocolos como USB o Ethernet, CAN no envía grandes bloques de datos de un punto a otro. CAN envía mensajes cortos como temperatura, o RPM (Revoluciones por minutos), y son enviados como broadcast a todo el sistema (Corrigan, 2002).

El protocolo de comunicación CAN, en su estandar ISO 11898, explica como la información viaja entre los dispositivos conectados a una arquitectura siguiente el modelo de OSI. La arquitectura planteada por la ISO 11898 se puede observar en la FIGURA (Corrigan, 2002)

En resumen, la capa física de CAN describe la definición del bit timing, bit enconding y la sincronización, los niveles aceptables de la señal (corriente y voltaje), los conectores y características físicas de los cables (Corrigan, 2002). Por otro lado la capa de enlaces de datos, provee todos los servicios necesarios para la transmisión de un stream de bits desde un nodo a otro.

El bus CAN es un bus de tipo broadcast, es decir que todos los nodos escuchan todas las transmisiones. No hay manera de enviar un mensaje a un determinado nodo, por lo tanto no es necesario el direccionamiento de nodos. (Kvaser, 2017).

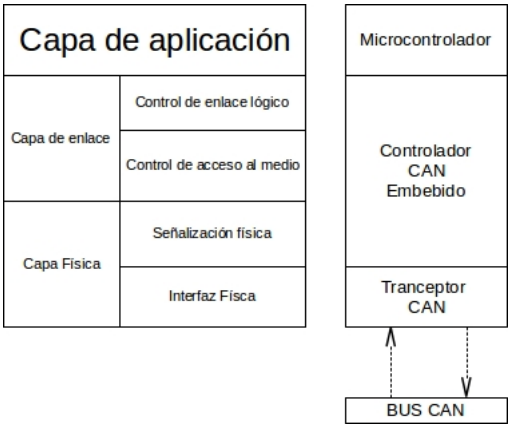

Figura 2.8: Arquitectura estándar propuesta por ISO898

Actualmente la industria aeronáutica la utiliza. También el área agrícola en sus maquinarias, control de tráfico, sistemas de control industrial, sistemas domóticos y además, en equipamientos médicos.

2.17.3. Elementos necesarios

Como se puede observar en la Figura 2.8

Cada nodo conectado a la red necesita de

- Central Proceesing Unit (CPU), o microprocesador, que decida que mensajes debe recibir y que debe enviar
- Controlador CAN, muchas veces integrado en el controlador, el cual recibe la serie de bits del bus. También es el encargado de transmitir mensajes (cuando se requiera), esto significa que al mensaje lo convierte en una serie de bits.
- Tranceiver, definido en el ISO 11898.

2.17.4. Capa física

El estándar de CAN define el bit enconding, el timing, y la sincronización. La capa física es la encargada de convertir 1 y 0 en pulsos eléctricos para enviar mensajes por el canal, y también el proceso inverso cuando recibe mensajes. La capa física es implementada enteramente en HW (Corrigan, 2008).

2.17.4.1. Bus de comunicación

Para iniciar una transmisión de mensaje es necesario como mínimo dos nodos conectados al bus CAN. Esto se debe a que el dispositivo que envía un mensaje está también recibiendo su propio mensajes, con esto puede chequear cada bit que ha enviado. De esta manera, un segundo nodo responde un un ACK mientras el bit todavía está siendo transmitido por el primer nodo. Esto demuestra por qué se necesitan de dos nodos para completar la transmisión de mensajes (Corrigan, 2008).

En la Figura 2.9 se observa un ejemplo extraído de (Corrigan, 2008). En esta se muestra un nodo A que envía un mensaje. Luego se ve que que B y C contestan con un ACK indicando que el mensaje fue recibido

sin errores. Luego B y C comienzan a transmiti hata que C gana el bus debido a que tiene un bit dominante, y termina este completando su mensaje.

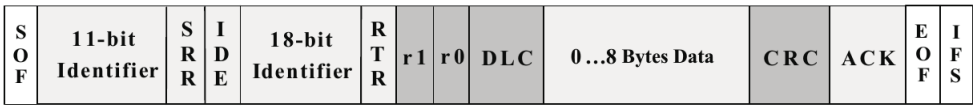

Figura 2.9: Tráfico en el BUS CAN

El médio físico es una línea de bus con dos cable, terminada por una resistencia. Los cables pueden ser paralelos, trenzados y/o blindandos. Los segmentos de cable para la conexión de los nodos deben ser lo más cortos posibles.

Para mayores detalle técnico referentes al bus será necesario estudiar el estándar ISO 11898 o en Corrigan (2008).

El largo del bus va a determinar el bitrate de la comunicación. Para alcanzar un bitrate de 1Mbps el máximo del canal de comunicación es de 40m. Si se tiene un bus de 1km el bitrate es de 0.05Mbps. Con esto se puede observar que el bitrate decae cuando la distancia se incrementa. Para el CAN el bitrate también está determinado por el total del delay del sistema (Corrigan, 2008). Esto es la suma de los delays del nodo, y el delay propio del cable.

Debe destacarse qde que existen diferentes capas físicas para el protocolo CAN:

- El tipo más común es el establecido en el estanar ISO 11898-2. Este protocolo cuenta con dos cables, cada uno identificado como CAN_H y CAN_L. También es llamado *CAN de alta velocidad* (1 Mbit/s). Este requiere que el alrgo del cable sea como máximo 40m.
- También en el estándar de ISO se estable el ISO 11898-3, el cual define otro esquema de doble cable balanceado pero para bajas velocidades. Este es tolerante a fallas si alguno de los cables entra en corto circuito. Este es llamado también como *CAN de baja velocidad* (125 kbit/s)
- Se define el SAE J2411 que utiliza un solo cable para la capa física. Este es utilizado en autos (velocidad por encima de los 50 kbit/s). No tiene demasiados requerimientos en cuanto al bitrate ni el largo del canal de comunicación. EL estandar define 32 nodos por red.
- Existen modificaciones del estandar RS485 para su funcionamiento con CAN

2.17.4.2. Bit timing

El tipo de señal es codificado con Non Return Zero (NRZ). Los bits son codificados en dos estados llamados “recesivo” y “dominante”(el bit 0 es asociado al bit dominante). El protocolo pemite acceso al bus multi-master con una resolución determinística ante colisiones. Para lograr la sincronización, el protocolo sincroniza durante las transiciones. Por tal motivo, deben evitarse el envío de largas cadenas de bits en un mismo estado. CAN utiliza una técnica que se denomina “bit stuffing” o “bit padding” en la cual luego de enviar 5 bits con el mismo estado, se insertan bits de relleno.

2.17.4.3. Asignación eficiente del bus

La asignación del bus depende de su aplicación. Generalmente existen 2 tipos de clases de asignación.

- **Asignación en un tiempo fijo**: La asignación se hace secuencialmente a cada participante sin importar si este necesita el bus en ese momento. Esta técnica, asigna tiempo del bus a cada nodo, y si no tiene nada que enviar, el bus se encuentra ocioso.
- **Asignación en base a sus necesidades**: el bus se asigna según las necesidades del nodo (utilizando CSMA o CSMA/CD). En CAN la asignación del bus es negociado entre los mensajes que esperan ser transmitido. CAN utiliza esta técnica.

2.17.5. Capa de Enlace

CAN utiliza el control de acceso al medio tipo *CSMA/CS+CR* (Acceso múltiple con detección de portadora, detección de colisión más resolución de colisión). CAN resuelve el problema de la colisión con la supervivencia de una de las tramas que chocan en el bus. La trama "ganadora.[es] aquella que tiene mayor prioridad. Por lo tanto se puede indicar que CAN por naturaleza tiene en cuenta la prioridad.

Como ya se mencionó anteriormente el bit *dominante* es el 0 y el bit *recesivo* es el 1, la resolución se realiza con una operación lógica AND de todos los bits transmitidos simultaneamente. Cada transmisor se encuentra continuamente observando y comprobando que el bit enviado se corresponda con el bit que envía. Cuando no coincide, el controlador retira el mensaje del bus y se convierte automáticamente en receptor. Como puede observarse la capa de enlace se comparta de manera similar a la capa física.

La única diferencia que presenta la capa de enlace de CAN es que todos los errores a nivel de un solo bits son detectados. Los errores de múltiples bits son detectados con una alta probabilidad (CAN-CIA, 11 de abril del 2017) s

2.17.6. Formato del mensaje

CAN utiliza un formato de mensajes cortos (94 bits) En el mensaje no está explícito ninguna dirección, por este motivo el mensaje puede ser escuchado por todos los nodos de la red (Kvaser, 2017).

Los tipos de mensajes son los siguientes:

- Frame de datos (Data Frame)
- Frame remoto (Remote Framte)
- Frame de error (Error Frame)
- Frame de sobrecarga (Overload Frame)

2.17.6.1. Data Frame

Este es el frame más común. Las partes más importantes son:

- Campo de arbitraje, el cual determina la prioridad del mensaje
- El campo de datos, que contiene desde cero hasta ocho bytes de datos
- El CRC, que está conformado por 15 bits utilizados para calcualr el checksum del mensaje

- Un campo de ACK. Cualquier controlador que haya recibido el mensaje envía un bit de acuse de recibo al final de cada mensaje. El transmisor comprueba la presencia del bit ACK. En caso de no detectar este bit reenvía el mensaje. Al no poder conocer la dirección de los nodos, no se sabe si el mensaje fue recibido correctamente por el node receptor, solo se sabe que el mensaje fue recibido por uno o más nodos.

2.17.6.2. Remote Frame

El frame remoto es un Data Frame con dos diferencias, es marcado como Remote Frame, esto es el bit RTR es recesivo; y por otro lado no hay un campo de datos. Este frame es utilizado para pedir la transmisión de un determinado frame de datos. Por ejemplo si el nodo A transmite un Remote Frame con el campo de arbitraje en 234, entonces el nodo B, reponderá con un frame de datos con el campo de arbitraje seteado a 234 (Kvaser, 2017). Este frame es poco utilizado.

2.17.6.3. Error Frame

Este frame se envía cuando un node detecta alguna falla en el mensaje. El envío de este frame provoca la retransmisión inmediata del mensaje. El Error Frame consiste en una bandera, el cual está compuesto por 6 bits del mismo valor, y un Error Delimiter que está compuesto por 8 bits recesivos.

2.17.6.4. Overload Frame

Este frame es similar al frame de error con el mismo formato. Es enviado cuando el nodo está ocupado. Este Frame es muy poco usado. El único controlador que generaba Overload Frame está obsoleto (Kvaser, 2017)

2.17.6.5. CAN estándar y CAN extendido

El protocolo de comunicación CAN, es un protocolo de multiple acceso con detección de colisión y arbitraje según la prioridad de los mensajes (CSMA/CD+AMP) (Corrigan, 2002). CSMA ya fue estudiado en 2.14.1. CD+AMP signfica que las colisiones son resueltas mediante arbitración por corrimiento de bits. El identificador con mayor prioridad es el que siempre gana el bus.

El CAN estandar (Figura 2.10) tiene los siguientes campos:

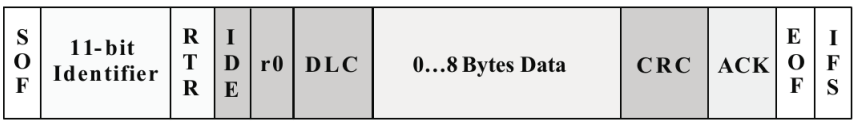

Figura 2.10: Frame de mensaje del CAN estándar

- SOF: es el bit dominante de inicio de frame, este marca el comienzo de un mensaje.
- Identificador: este es un identificador de 11 bits, en el cual el valor binario más bajo es el de mayor prioridad.
- RTR: El bit denominado Remote Transmission Request (RTR) es dominante cuando la información es requerida desde otro nodo. Todos los nodos reciven el pedido, pero el identificador determina node específico.

- IDE: Es una extensión del bit identificadorque signfica que se está por transmitir un un identificador estandar CAN sin extensión.
- r0: Bit reservado.
- DLC: 4 bits que contiene el número de bytes que van a transmitirse.
- Data: Hasta 64 bits, es el mensaje que se transmite.
- CRC: Son 16 bits que conforman el Cyclic Redundancy Check (CRC) para chequear la existencia de algún error.
- ACK: Todos los nodos que reciben el mensaje correctamente sobreescribe este bit recesivo, indicando que han recibido el mensaje sin errores. Si un nodo detecta un error y detecta la persitencia de este bit recesivo. Con esto, se descarta el mensaje enviado y el nodo transmisor vuelve a enviar el mensaje.
- IFS: Son 7 bits que representan el espacio entre frames.Es el tiempo que requiere el controlador para mover el mensaje hasta el buffer de mensajes recibidos.

El CAN extendido (Figura 2.11)es idéntico al CAN estándar, salvo por algunos detalles:

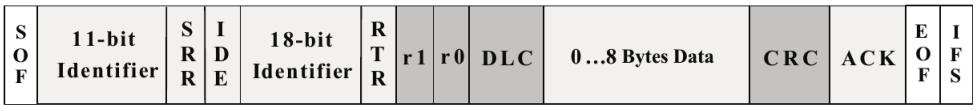

Figura 2.11: Frame del mensaje del CAN extendido

- SRR: Es el bit de peteción remota subtituta que remplaza al bit RTR del mensaje estándar.
- IDE: Un bit recesivo en el identificador de extensión (IDE) indica que siguen más bits de identificación. A este le siguen 18 bits.
- r1: Este bit viene después del RTR, es un bit adicional reservado.

2.17.6.6. Arbitraje

Cualquier controlador CAN, cuando detecta que el bus está desocupado puede comenzar a transmitir. Esto puede ocacionar que al mismo tiempo, dos o más controladores comiencen a transmitir al mismo tiempo. Este conflicto se resuelve de una manera ingeniosa. El nodo transmisor a medida que transmite los bits, se encuentra monitoreando el bus. Si por ejemplo, el nodo detecta un bit dominante cuando este envío un bit recesivoa, el nodo inmediatamente deja de transmitir y se convierte en un receptor. Cuando el bus se encuentra desocupado, recién vuelve a transmitir (Kvaser, 2017).

Una condición importante que se debe cumplir es que dos o más nodos no pueden transmitir el mismo campo de arbitraje.

Capítulo 3

Estado del arte

3.1. Árboles binarios

El concepto de arquitecturas de árboles binarios es aplicable en el desarrollo de sistemas de computadoras jerárquicas, y sobre todo en computadoras de alta perfomance (Raghavendra et al., 1984). Existen dos diferentes mecanismos de tolerancia a fallas (Raghavendra et al., 1984):

1. Esquemas con back up.
2. Esquemas con degradación de perfomance.

Teniendo en cuenta que estas arquitecturas son aplicadas principalmente en la construcción de circuitos VLSI[1] (Singh y Youn, 1991), se asume su aplicabilidad a arquitecturas de aviónica.

La FT y la perfomance de los sistemas dependen de las capacidades de las redes que se utilizan para la comunicación entre unidades de procesamiento (Raghavendra et al., 1984).

Un árbol binario está compuesto por nodos y enlaces (links). Existe un nodo central dónde se desprenden dos nodos hijos, estos se encuentran enlazados al nodo padre. Así recursivamente, se van generando dos nuevos hijos, por cada uno de los nodos. Los árboles binarios están divididos en niveles, que representa cada una de las generaciones de nodos.

Este tipo de topología tiene algunas problemas que la FT debe hacer frente. En las arquitecturas basadas en árboles binario, existe una cierta probabilidad de que un nodo o un link falle (Raghavendra et al., 1984). Las arquitecturas de árbol binario son en general físicamente estáticas. Por lo tanto, cualquier falla en uno de sus nodos (o links) demandaría una avería a nivel sistema, lo cual daría lugar a una pérdida de misión. Para ello se debe dotar a la arquitectura de un mecanismo de reconfiguración.

La tolerancia a fallas en arquitecturas binarias ya fueron estudiadas en profundidad en Hayes (1976), Raghavendra et al. (1984), Singh y Youn (1991). Para lograr FT en estas arquitecturas se las deben diseñar con un número mínimo de nodos de backup y links redundantes, de modo tal de hacer frente cualquier punto de falla simple en la arquitectura.

[1] Del inglés, Very Large Scale Integration

3.1.1. Esquema de árbol binario con backups

El esquema planteado por Raghavendra et al. (1984) es similar al que se muestra en al Figura 3.1. En este esquema se agrega nodos y links redundantes como técnica de FT. Existe un nodo de backcup por cada nivel del árbol.

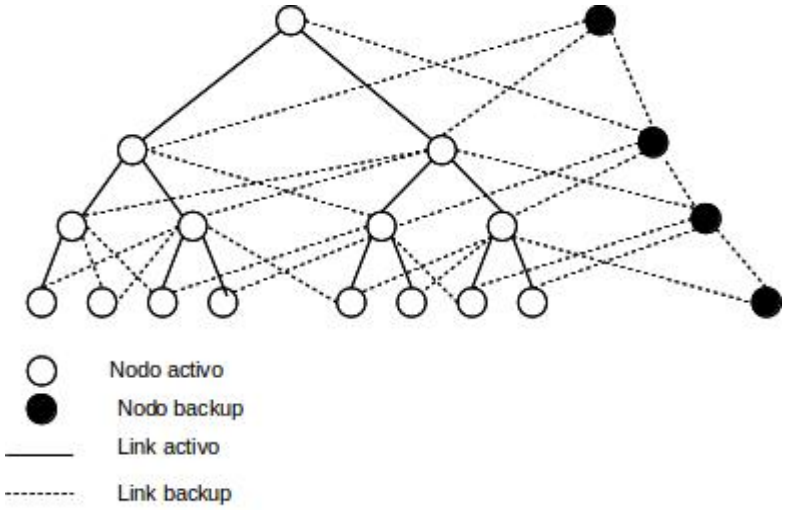

Figura 3.1: Árbol binario de 4 niveles

Esta arquitectura cuenta con una restricción, la cual indica que solo se puede tolerar una falla singular, por cada nivel de la arquitectura. Arquitecturas de este tipo, podrían tolerar más de una falla, sólo si se dan en diferentes niveles del árbol (Raghavendra et al., 1984). Para agregar más tolerancia, se deberían agregar más redundancias.

Notese en la Figura 3.1 que cuando una unidad de procesamiento (nodo) falla, todos los links se deben reajustar hacia el nodo de la derecha. En este punto es importante mencionar, que ante esta situación se requiere una reconfguración a nivel de HW, además de una reconfiguración a nivel de SW. Es necesario algoritmos de ruteo dinámicos, de modo tal de conocer los nuevos caminos que intercomunican nodos, para mantener la operabilidad del sistema.

3.1.1.1. Estimación de la confiabilidad de un árbol binario con backup

Se asume que la probabilidad de fallas de los links es muy baja en comparación con la de los nodos. Teniendo que el rate de falla es de λ, la confiabilidad de un nodo es $R = e^{-\lambda t}$. También se sabe que un árbol binario con n niveles, se tiene $2^n - 1$ nodos en total. Entonces la confiabilidad de todo el sistema es:

$$R_{nr} = R^{2^n - 1}$$

Raghavendra et al. (1984) incluye en sus cálculos un factor de cobertura c, el cual es la probabilidad condicional de que se lleve a cabo una recuperación exitosa, luego de que una falla se haya detectado. Entonces la confiabilidad del sistema para una arquitectura de árbol binario con redundancias (un nodo backup por nivel) es el siguiente:

$$R_{sys} = \prod_{k=0}^{n-1} [R^{2k+1} + R^{2k}(1-R) + 2^k c R^{2k}(1-R)]$$

.

Simplificando:

$$R_{sys} = R^{2n+1} \prod_{k=0}^{n-1} [(2^k c + 1) - 2^k cR]$$

En la Figura 3.2, se observa la confiabilidad de una arquitectura de árbol bianrio de 4 niveles, con una cantidad de $2^4 - 1 = 15$ nodos. En color negro se grafica una arquitectura sin redundancia, mientras que en color rojo, azul y cyan, se muestra arquitecturas redundadas con diferentes c (0.98, 0.99, 1, respectivamente).

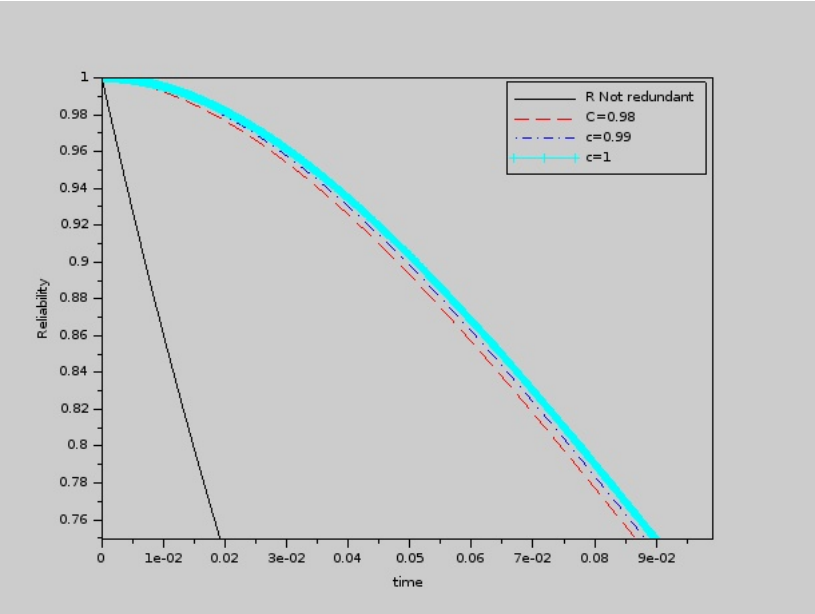

Figura 3.2: Confiabilidad con respecto al tiempo de una arquitectura de árbol binario de 4 niveles

Con esto podemos indicar, como es de esperarase, una arquitectura de árbol binario redundanda permite mantener un alto nivel deseable de confiabilidad, durante un mayor lapso de tiempo, a diferencia de un sistema no redundando. También se puede concluir que con un factor de cobertura c más próximo a uno, maximiza los niveles de confiabilidad con respecto al tiempo.

3.1.1.2. Extensión de la arquitectura con backup

3.1.2. Esquema de árbol binario con degradación de perfomance

3.2. Sistemas Hypercube

Escribir algo

3.3. Redes distribuídas

Una red de computadoras hace referencia a un conjunto de computadoras autónomas que se encuentran interconectadas, esto quiere decir que pueden intercambiar información (Tanenbaum y Wetherall, 2012). Una red distribuída tiene como principal caracteerística que no existe un nodo central que "gestione"toda la red. Todas las cargas de las tareas y/o actividades son distribuídas entre los nodos que forman parte de la red.

Una red distribuída es tolerante a fallas si los nodos pueden formar subredes (Stivaros, 1992). Es decir, la red se debe mantener activa y conectada, con diferentes topologías e interconexiones, que permitan tolerar posibles fallas producidas en algunos nodos y permitir mantener la perfomance (Stivaros, 1992). Debido a que el procesamiento de la red se encuentra distribuída en todo el sistema, esto brinda una ventaja por encima a los sistemas centralizados desde el punto de vista de la confiabilidad (Pradhan y Reddy, 1982). Un componente importante de las redes distribuídas tolerantes a fallas, es la topología del sistema (Pradhan y Reddy, 1982).

Siguiendo la notación de Pradhan y Reddy (1982) para describir la topología del sistema se utiliza, un gráfico sin direccionamiento G = <V,E>, donde V representa un set de nodos y E representa un set de relaciones. Stivaros (1992) agrega que $V(G) = \{v_1, v_2, v_3, ...v_n\}$ representa un vector de nodos, los cuales tiene probabilidades de operación $P = (p_1, p_2, p_3, p_n)$; y define una función de asignación π, la cual es una función que asigna V con una probabilidad $P_{\pi(v)}$.

La FT de la red G, dado el vector de probabilidades y una función de asignación π, tal como se viene discutiendo anteriormente, es la probabilidad de que la red continúe funcionando, es decir continúe conectada, aún en la falla (aleatoria) de alguno de sus nodos, esto se denota $FT(G;\vec{P},\pi)$.

Se dice que un subset de nodos S, es un estado tolerante del sistema G, cuando estos nodos se mantienen conectados y funcionales. Se utiliza θ para indicar el conjunto de todos los S posibles. Un estado tolerante S contribuye $\prod_{v\in S} P_{\pi(v)} \prod_{v\notin S}(1 - p_{\pi(v)})$ a la probabilidad de FT.

Para calcular el total de la FT del sistema, incluyendo todo los S, se hace:

$$FT(G;\vec{P};\pi) = \sum_{S\in\theta} Pr(S) = \sum_{S\in\theta} \prod_{v\in S} P_{\pi(v)} \prod_{v\notin S} (1 - p_{\pi(v)})$$

Una relación entre nodos es representado como ij, lo cual representa un enlace bidireccional entre nodos. El grado del nodo i representa el número de relaciones que inciden en ese nodo, el cual se escribe d_i. Así, d_i está limitado por el número de puertos de entrada y salida disponibles por cada nodo (Pradhan y Reddy, 1982). k_{ij} representa el número mínimo de *hop* (hop representa la transmisión a través de un link de datos)(Pradhan y Reddy, 1982)

Stivaros (1992) y Pradhan y Reddy (1982) mencionan la existencia de varias topologías que pueden ser aplicadas en una red distribuída tolerante a fallas. Una topología estrella posee una baja distancia entre nodos, pero una pobre tolerancia a fallas (Pradhan y Reddy, 1982) (Stivaros, 1992). La topología anillo, permite un simple ruteo, pero existen grandes distancias internodo. Un sistema completamente interconectado, presenta buenas características tolerantes a fallas, pero tiene un alto costo (Pradhan y Reddy, 1982). Pradhan y Reddy (1982) propone una topología para una arquitectura distribuída de comunicación. Esta es una topología robusta, y puede llegar a ser compleja a medida que aumentan los nodos.

3.3.1. Algoritmo de ruteo

Los algoritmos de ruteos son necesarios en el desarollo de una arquitectura tolerante a fallas y reconfigurable. Los algoritmos de ruteo se los pueden dividr en dos: *algoritmos primarios* y *algoritmo alternativo*. El primero es utilizado cuando no hay fallas de nodos. Se deben mantener los caminos para llegar, correctamente, al nodo destino. El segundo se utiliza en la presencia de alguna falla, este requiere que sea capaz de detectar la presencia de fallas, para luego reconfigurar el sistema. Estos algoritmos deben ser simples y requerir una mínima cantidad de HW y SW.

Agregado a lo mencionado en el párrafo anterior, deben existir algoritmos de diagnóstico distribuído de falas, basados en los algoritmos de ruteo (Pradhan y Reddy, 1982).

3.4. Redes Ethernet en aviónica

TTEthernet es una tecnología de red de computadoras comercializada por TTTech Computertechnik AG para el desarrollo de aplicaciones seguras. SAE International[2] estandarizó esta red como SAE AS6802. TTEthernet se basa en el Ethernet clásico, en el cual se pone énfasis en las características principales que deben respetarse en sistemas críticos, tales como latencias de mensajes determinísticos, presición de tiempo real, tolerancia a fallas (Loveless, Andrew T., 2015). Tiene la capacidad de transmitir datos 100 veces más rápido que la que lo hacen las tecnologías tradicionales tales como el MIL-STD-1553.

El Ethernet clásico presenta ventajas, tales como su alta velocidad de transmisión de datos, flexibilidad, y su disponibilidad y bajo costo (ya que se trata de un componente COTS)(Loveless, Andrew T., 2015), hacen deseable su aplicación en el área espacial. fue utilizado en diferentes proyectos aeroespaciales y en misiones importantes tales como el Space Shuttler y la Estación Espacial Internacional (ISS) (Loveless, Andrew T., 2015). A pesar de esto, el Ethernet no cumple con el determinismo requerido por las aplicaciones de tiempo real de un vehículo espacial. Por tal motivo, se desarrolla el sistema TTEthernet, el cual introduce un reloj de sincronización descentralizado, permitiendo la transmisión mensajes Time-Trigged (TT). En este tipo de red, existe una herramienta de planning que asigna a cada dispositivo un intervalo de tiempo, en el cual puede utilizar para transmitir frames. Estos sistemas utilizan Links Virtuales (VL) para permitir el envío de mensajes time-trigged. Cada VC es asociado con un time-trigged frame através de un identificador de tráfico crítico (CTID), este reemplaza el control de acceso al medio (MAC) (Loveless, Andrew T., 2015). Una red simple se muestra en la Figura 3.3.

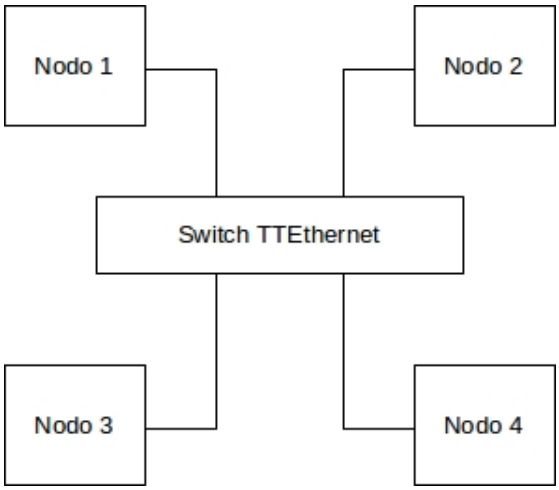

Figura 3.3: Arquitectura básica TTEthernet

TTEtheret puede actuar en dos clases de tráfico, con el obejtivo de soportar diferentes niveles de cricticidad de mensajes. Estas clases de tráficos son las siguientes (Loveless, Andrew T., 2015) (Steiner, Willfried, 2013):

- Time-Triggered, permite enviar mensajes de acuerdo a una planificación predefinida (scheduling),
- Rate-Constrained (RC), en el cual se llevan algunas restricciones de tamaño y rate de transmisión de frames,
- Best-Effort (BE), el cual se comporta de manera similar que el Ethernet

El paquete TTEthernet tiene una gran similitud con el frame del estandar IEEE 802.3 (Ethernet). Se denota

[2]http://www.sae.org

en la Figura 3.4 que en lugar de la MAC del 802.3 frane se divide en el CT Marker y el CTID. El primero es un identificador estático utilizado para distinguir paquetes TT de otros tipos de tráficos.

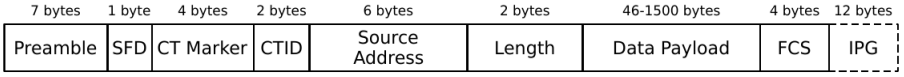

Figura 3.4: Frame de mensaje TTEthernet

El estandar SAE AS6802 define el protocolo de sincronización del TTEthernet. El protocolo TTP fue diseñado para reducir la complejidad de las arquitecturas distribuídas tolerantes a fallas (TTTech, 20 de Enero del 2017). Existe un grupo de dispositivos que relojes locales que permite la sincronización requerida por la comunicación TT, a esto se lo denomina dominio de sincronización (Loveless, Andrew T., 2015). Cada dominio de sincronización es asignado a uno de los siguientes roles:

- Compression Master (CM)
- Synchronization Master (SM)
- Synchronization Cliente (SC)

3.4.1. Experiencia de vuelo

Loveless, Andrew T. (2015) demuestra la aplicación de esta tecnología para computadoras de vuelo redundantes en misiones de naves simuladas.Esta tecnología toma tanto interés que el Sistema de Exploración Avanzada (AES)[3] de la NASA, lleva a cabo un proyecto denominado Avionics and Software (AS). TTEthernet también fue utilizada en una arquitectura tolerante a fallas en El Integrated Power, Avionics and Software (IPAS) del Johnson Space Center (JSC) y en el Core Flight Software (CFS) del Asteroid Redirect Mission (ARM) simulado[4] (Loveless, Andrew T., 2015). También fue utilizado exitosamente durante la misión Orion (TTTech) .

TTEthernet simplifica el diseño de sistemas espaciales que deben contar con tolerancia a falla y una alta disponibilidad. La seguridad y la redundancia se mantiene sin ningún tipo de aplicación extra.

3.5. Arquitectura de red basada en BUS

En Tai et al. (1999) se presenta una arquitectura tolerante a fallas basada en BUS. Esta topología forma parte de un programa denominado X2000, el cual tiene como objetivo desarrollar una arquitectura tolerante a fallas y basada en componentes COTS. la cual pertenece a NASA. Esta este programa se desarrolla bajo una filosofía de misiones espaciales la cual dice: "Más rápido, mejor, más económica"[5].

Esta arquitectura utiliza una topología novedosa denominada *stack-tree topology* (Chau et al., 1999) (Tai et al., 1999). Una stack-tree es un árbol donde cada nodo rama se encuentra conectado como mucho a tres otros nodos de los cuales, como máximo, dos son nodos ramas (Tai et al., 1999). Una Complete Stack-Tree (CST) es aquella en donde cada nodo rama está conectado al menos 1 nodo rama (Tai et al., 1999).

En las Figura 3.5 se pueden observar ejemplos de stack-tree. Mientra que en esta otra Figura 3.6 no es una stack-tree. Las imagenes fueron extraídas de Tai et al. (1999)

[3] Del ingles, Advanced Exploration Systems
[4] Los mencionados anteriormente son ejemplos de la utilización de TTEthernet
[5] En ingles, faster, better, cheaper

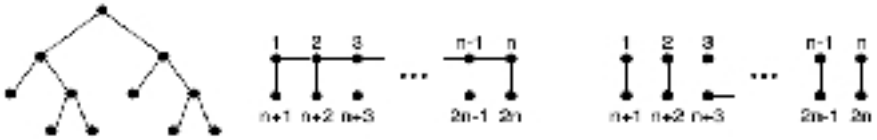

Figura 3.5: Arquitecturas stack-trees

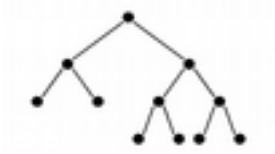

Figura 3.6: Arquitecturas que no responden al modelo stack-trees

Como el objetivo principal es desarrollar una arquitectura tolerante a fallas, esto es, que aún cuando un nodo o un link entre nodos falle, el sistema completo debe continuar funcionando, sin ningún tipo de degradación en su servicio. Para cumplir con este objetivo Tai et al. (1999) trabajan con un esquema de denominado CST de equema dual (CST_D) [6]. Esta puede observarse en la Figura 3.7 extraída de citeTai99.

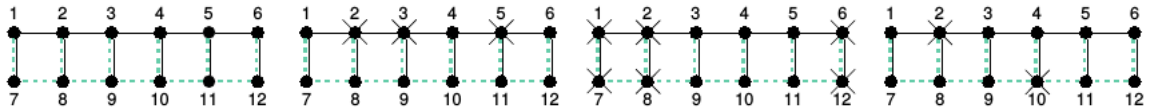

Figura 3.7: Esquema CST_D

Para aumentar la confiabilidad de la arquitectura ante la acurrencia de fallas, se agrega links de backups que unen los nodos iniciales con el final, dando un efecto 3D de anillo (Tai et al., 1999).

3.5.1. Evaluación de la confiabilidad de arquitecturas basadas en BUS

La confiabilidad de esta arquitectura (como ya se viene mencionando) es la probabilidad de que, a lo largo del tiempo de vida de la misión t, la arquitectura se encuentra funcional, en la cual todos los nodos (aquello que no hayn fallado) se encuentren conectados (Tai et al., 1999). (Tai et al., 1999) y (Chau et al., 1999) asumen que la probabilidad de que un nodo falle es mucho mayor a que un enlace (el bus físico) falle.

En Tai et al. (1999) indica que la confiabilidad de la red depende del tamaño k (k es el número de nodos ramas). La confiabilidad de una red basada en CST simplex es la probabilidad $U(k)$ de que los nodos no fallen, o que ante una falla se detecte y se lleve a cabo correctamente la reconfiguración del sistema.

$$U(k) = (1-q)\sum_{j=0}^{k}\binom{k}{j}(1-q)^{k-j}(cq)^{j}$$

donde $q = 1 - e^{-\lambda t}$ es la probabilidad de que un nodo falle durante el tiempo de vida de la misión f. Entonces

$$R_s^{CST} = \sum_{k=1}^{n}(n-k+1)U(k)(cq)^{2(n-k)}$$

donde $(cq)^{2(n-k)}$ es la probabilidad que los $2(n-k)$ nodos que conforman un cluster fallan, y esta falla es detectada y se lleva a cabo una correcta reconfiguración.

[6] En ingles,CST dual scheme

Para un CST dual ocurre de manera similar. Se define $V(k)$ de la siguiente manera:

$$V(k) = 2(1-q)^k \sum_{j=1}^{k} \binom{k}{j} (1-q)^{k-j} (cq)^q + (1-q)^{2k}$$

Entonces la medida de $R_D^{CST} = \sum_{k=1}^{n} (n-k+1) V(k) (cq)^{2(n-k)}$ p

3.6. Métrica y modelado de la confiabilidad de sistemas

Es de suma importancia llevar a cabo el análisis de la confiabilidad, disponibilidad y mantenibilidad (RAM[7]) de sistemas satelitales, durante la fase de diseño (Hoque et al., 2015), a fin de lograr la mínima cantidad de fallas o incrementar el MTBF (Peng et al., 2013) . Llevar a cabo esto, es de gran importancia, ya que permite el desarrollo de estrategias que permitan altos grados de confiabilidad, disponibilidad y mantenibilidad (Hoque et al., 2015).

Existen dos categorías de medición de la confiabilidad y predición (Schneidewind, 1997), estas son utilizadas para asegurar la seguridad del software de sistemas críticos (Schneidewind, 1997), las cuales son:

- medición y predicción que están asociadas con las fallas y errores residuales.
- medición y predicción que están asociadas con la disponibilidad del sistema a sobrevir durante la misión sin experimentar fallas en el fallas (o pérdidas) en el sistema.

Las dos categorías mencionadas anteriormente son explicadas en Schneidewind (1997).

Según Liu et al. (2014) las severidades de las fallas son clasificadas como críticas, peligrosas o triviales, teniendo en cuenta la contribución de esa falla a la pérdida de la misión. Es importante, además, conocer el riesgo de una falla. El riesgo, se define como la posibilidad de que una falla produzca una lesión (por ejemplo, un astronauta en vuelos tripulados), algún daño material (por ejemplo, la destrucción del satélite), o una pérdida (por ejemplo, la pérdida de la misión).

Dependiendo de la misión, un criterio para definir si un sistema es seguro o no, es reduciendo las fallas que pueden provocar pérdidas de vida, pérdida de la misión o la obligación de abortar una misión (Schneidewind, 1997). Schneidewind (1997) define dos criterios que deben satisfacerse:

- $r(t_t) < r_c$,
- $T_F(t_t) > t_m$

dónde t_t es el Tiempo total de testing (observado o predicto); $r(t_t)$ son las fallas restantes hasta t_t; r_c es una valor crítico de fallas restantes ;$T_F(t_t)$ es la métrica para medir el riesgo; y t_m es la misión de la duración.

Lo anterior signifca que un sistema crítico será seguro si: las fallas restantes en el tiempo de prueba son menores a un valor crítico de cantidad de fallas, o la duración de una misión es mayor al tiempo que se de la siguiente falla.

En la literatura se utilizan modelos matemáticos para modelar los sistemas críticos y calcular así su confiabilidad. La mayoría de ellos asumen, que todas las fallas tienen igual tasa de detección de fallas, como así

[7] Del ingles, Realibility, Availability, Maintainability

también la misma severidad, lo cual no es correcto Liu et al. (2014). Las técnicas de verificación formal tiene un fuerte enfoque para la verificación de sistemas complejos, y permitir verificar las propuedades deseadas de los sistemas (Peng et al., 2013). Es importante llevar a cabo una verificación de modelo[8], esta incluye técnicas de verificación automática para sistemas de estados concurrentes finitos. En estos modelos, las especificaciones de los sistemas son escritos en una lógica temporal y proposicional, y el procedimiento de verificación es una búsqueda del espacio de estado del diseño (Hoque et al., 2015).

En Hoque et al. (2015) se utiliza la *verificación probabilística de modelos*, la cual es utilizada para verificar sistemas, de los cuales los comportamientos son estocástico por naturaleza. Este se basa en la construcción y análisis de modelos probabilisticos como cadenas de Markov. En Hoque et al. (2015) se indica que estas técnicas fueron aplicadas en misiones de NASA. Este modelo se basa en la construcción y en el análisis de sistema de un modelo probabilístico de el sistema, típicamente una cadena de Markov (Hoque et al., 2015). Los modelos de Markov son muy utilizados para los análisis de confiabilidad y disponibilidad de sistemas complejos (Hoque et al., 2015).

En Hoque et al. (2015) se lleva a cabo un modelo simplificado de un sistema satelital este se muestra en la Figura 3.8. La descripción del modelo se llevó a cabo usando el lenguaje PRISM. El modelo que se describe en Hoque et al. (2015) y Peng et al. (2013) se refiere a un modelo a nivel de sistema/misión, muy diferente a lo que se pretender realizar en este trabajo de tesis, pero es factible basarse en los conceptos que describe la literatura, ya que lograron aumentar la confiabilidad del modelo propuesto, en contraposición del modelos çlásicos"

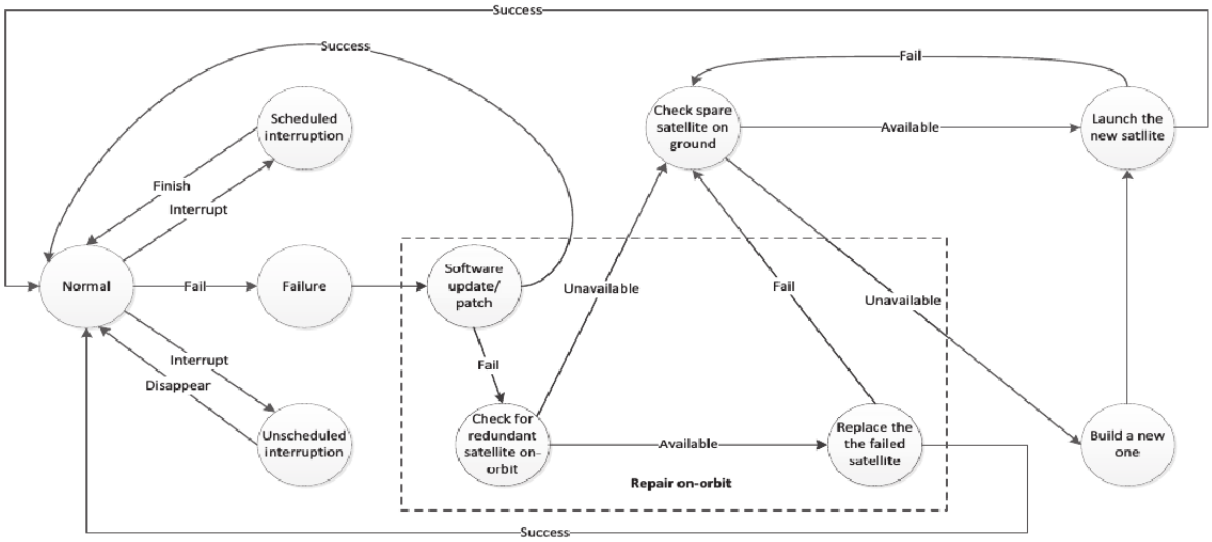

Figura 3.8: Modelo de sistema satelital (Hoque et al., 2015)

Hoque et al. (2015) y Peng et al. (2013) relacionan tasa de falla λ, confiabilidad R_e y MTBF con la siguiente ecuación

$$\lambda = \frac{-lnR_e}{MTBF}$$

Esteve et al. (2012) muestra el caso de estudio del desarrollo de una plataforma satelital a un alto nivel conceptual. Demuestran la correcta utilización de métodos formales y herramientas para la industria satelital. Se utiliza la herramienta desarrollada por el consorcio COMPASS[9] para el modelado formal y análisis, que es

[8]Model checking

[9]Correctness, Modeling and Performance of Aerospace Systems

utilizado por la industria espacial de Europa. Llegan a la conclusión de que mediante la utilización de modelos formales, se logra desarrollar un modelo completo, que cumple con todos los aspectos necesarios. Esto asegura la consistencia de los análisis (Esteve et al., 2012).

El estado de la cuestión sobre la temática tratada en esta sección, indica que debería utilizarse modelos formales para el análisis de los sistemas. Se llega a la conclusión de que no existe un método (o modelo) completo y exclusivo para lo que se pretende desarrollar en este trabajo de tesis. Por ello se deberá utilizar y moldear las metodolgías vistas en esta sección, para lograr cumplir con los obejetivos del presente trabajo.

3.7. CAN en la actividad espacial

agregar info

Capítulo 4

Análisis y desarrollo de arquitectura tolerante a fallas

4.1. Introducción

En este capítulo, en primer lugar, se plantea una serie de requerimientos (no estrictos) que ayudan en el desarrollo de una arquitectura para una misión satelital ficticia basada en componentes COTS. Luego, en base a estos requerimientos, se presentan modelos para la medición de confiabilidad de tres tipos diferentes de topologías de comunicación de subsistemas satelital. Con ello, se pudo elegir cual es la topología que asegura una mayor tolerancia a fallas.

Como siguiente paso

4.2. Requerimientos para el análisis

En esta sección no se pretende realizar una lista de requerimientos formales para el desarrollo de una arquitectura, esto se hace el capítulo siguiente. El objetivo de esta sección, es llevar a cabo una guía sencilla de las partes principales de un sistema satelital. Con esto en mente se podrá desarrollar diferentes topologías de comunicación, y luego analizar su tolerancia a falla.

Se supondrá una misión de 15 años (sin carga útil para simplificar el análisis) cuyo sistema estará compuesto por los siguientes subsistemas (en inglés para mantener correspondencia con la literatura) basado en Fortescue et al. (2003):

- Power Subsystem
- Atitude and Orbit control Subsystem
- Telemetry and Subsystem
- Thermal Subsystem
- Propulsion Subsystem
- Data Handlig Subsystem

El sistema, entonces, está compuesto por 6 nodos. Los nodos se suponen computadoras con capacidad de procesamiento suficiente para cada subsistema. Cada una de estas computadoras/nodos es un componente COTS con un cierto grado de confiabilidad (se suponen lo suficientemente bajo como para no ser utilizado en forma directa en el desarrollo de satélites). A nivel de HW estos nodos cuenta con tolerancia a fallas, lo que aumenta su confiabilidad. El subsistema de Data Handling tiene que tener comunicación con todos los subsistemas, ya que será la encargada de controlar el correcto funcionamiento del sistema, enviar comandos, y empaquetar telemetría de los sensores.

4.3. Nomenclatura

Durante el análisis se utilizará la siguiente nomenclatura:

T_m	Tiempo de la misión
$TTNF$	Tiempo hasta la siguiente falla
λ	Tasa de falla
$MTBF$	Tiempo medio entre fallas
$MTTR$	Tiempo medio de reperacion
$A(t)$	Disponibilidad
$R(t)$	Confiabilidad

Como se definió anteriormente el T_m es de 15 años. Para simplificar los trabajos de cálculos y ejecución del presente trabajo, se supone que la arquitectura puede fallar solo una vez durante la misión satelital. Por lo tanto, el $TTNF$ será de 15 años. La tasa de falla se define como:

$$\lambda = \frac{1}{15}$$

. Suponiendo que la arquitectura no debería fallar durante los 15 años de misiones se da la siguiente sisituación:

$$MTTBF = MTTR = 15$$

Por otro lado, La disponibilidad es $A(t) = 99\,\%$. La confibialidad, como se estudió en secciones anteriores, es $R(t) = e^{-\lambda t}$

4.4. Estudio de topologías de arquitecturas

Luego de un estudio exhaustivo del estado de la cuestión (Capítulo 3) se llegó a la conclusión de que las topologias más estudiadas, por ende más maduras y sencillas de aplicar a las activiades que se pretenden realizar en la presente tesis son:

- Árbol binario
- Red distribuída

- Arquitectura hypercube

En esta sección, se definirá modelos para medir la confiabilidad de cada una de las topologías de arquitectura que aseguren una mayor tolerancia a fallas a nivel de sistema, de modo tal que si se llega a producir una falla en cualquiera de los nodos (sistemas de procesamiento) de la arquitectura, esta puede reconfigurarse, pemitiendo que esta continúe funcionando, sin sufrir ningún tipo de degradación, aún en la presencia de fallas.

4.4.1. Árbol binario

En primer lugar se planteó un árbol binario de cuatro niveles con back up, basandose en el diseño de Raghavendra et al. (1984) 3.2. Este diseño cuenta con $2^n - 1 = 15$ nodos. De lo estudiado en Sección 3.1 la confiabilidad puede ser calculada de la siguiente manera:

$$R_{sys} = R^{2n+1} \prod_{k=0}^{n-1} [(2^k c + 1) - 2^k cR]$$

Con esto se puede observar la confiabilidad de la red con respecto al tiempo 4.1.

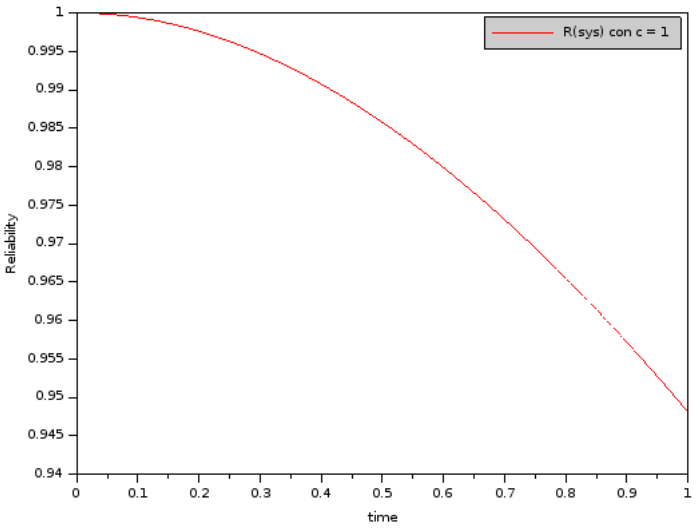

Figura 4.1: Confiabilidad con respecto al tiempo de árbol binario de 4 niveles

La cantidad de niveles que se eligió para esta topología depende del número de subsistemas que se requieren.

Los nodos back up se mantienen inactivos, es decir no participan en el procesamiento durante la vida normal del sistema. En caso de producirse una falla, se supone que estos nodos de redundancia, comienzan a funcionar automáticamente, sin ninún tipo de retraso. Esto, que no corresponde con la realidad, permite simplificar los cálculos para este trabajo de tesis.

4.4.2. Red distribuida

El estudio de una topología de red distribuída no es tan sencilla como la que se plantea para un árbol binario. Para el desarrollo de esta red, además de los 6 nodos que representa cada uno de los subsistemas requeridos, se agrega 2 nodos de redundancia. Por lo tanto se tiene una red de 8 nodos. Siguiendo la metodología de desarrollo presentado por Pradhan y Reddy (1982) se llevó a cabo la red que se presenta en la Figura 4.2.

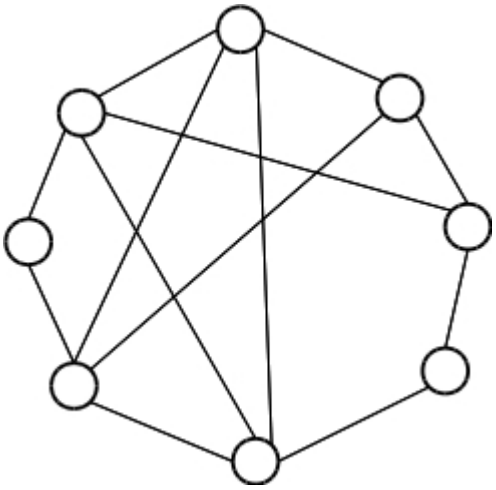

Figura 4.2: Red distribuida

Esta cuenta con 4 nodos de grado 4, 2 nodos de grado 3, y 2 nodos de grado 2. Ante cualquier falla de alguno de los nodos de la red, esta topología tiene la capacidad formar subredes, que mantien todos los nodos conectados (a excepción del fallado), asegurandose la funcionalidad y reconfiguración del sistema completo. Estratégicamente y para lograr la condición mencionada anteriormente, se crea la **condición** de que pueden fallar hasta 4 nodos simultaneamento. Esto aseguraría de que la red continuará funcionando aún en la presencia de fallas (tolerante a fallas).

Se modificó la fórmula desarrollada por Stivaros (1992), para lograr una cohrencia de los modelos que se vienen planteand en el presente trabajo. Teniendo en cuenta que la confiabilidad del sistema completa es:

$$R(t) = \prod_{v \in S} e^{-\lambda t}$$

donde v representa el nodo y S es el subsistema funcional. Es decir, que este modelo recorre todos los nodos funcionales. Cuando existen nodos con fallas, y que dejan de ser funcionales, el modelos es el siguiente:

$$R_{sys} = \sum_{i=0}^{k} ((\prod_{v \in S} R(t)) - (\prod_{v \notin S} (1 - R(t)c)))$$

donde c se definió como una *constante de degradación del sistema* para modelar una degradació del sistema.

En la Figura 4.3 se puede observar la *curva de confiabilidad* del sistema, para el caso en el que todos los nodos se encuentran funcionales.

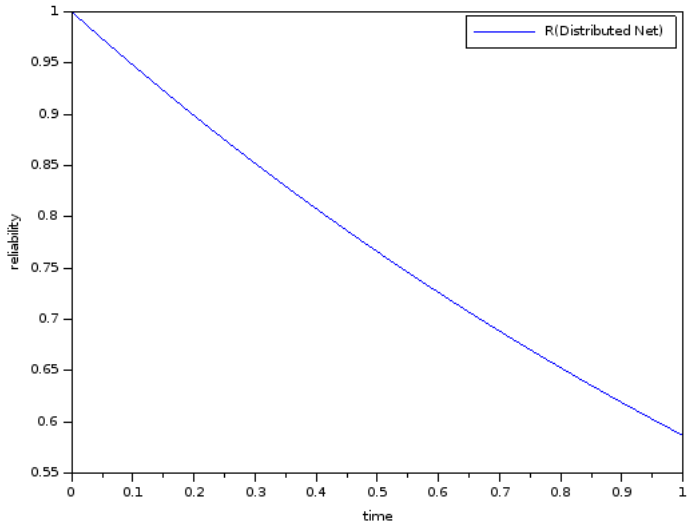

Figura 4.3: Confiabilidad de red distribuida

Para el caso de la falla de todos los nodos la *curva de confiabilidad* (Figura 4.4) muestra correctamente la degradación esperada de la confiabilidad, con respecto al sistema funcionando correctamente sin ninguna falla.

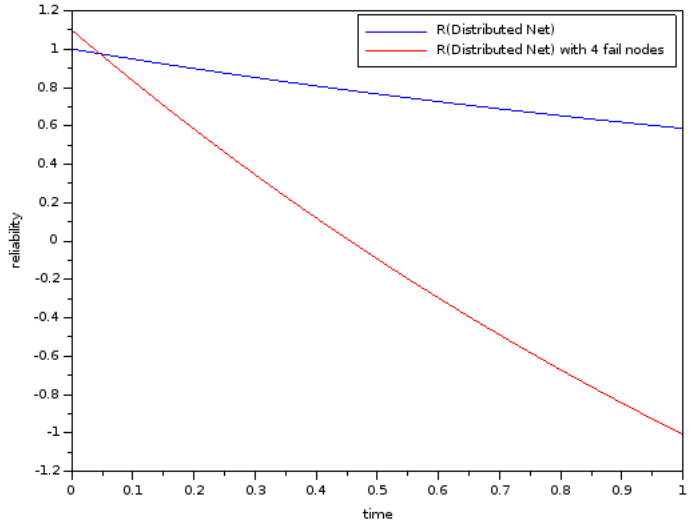

Figura 4.4: Confiabilidad de red distribuida con 4 nodos fallando

4.4.3. Red hypercube

Para el caso de la red hypercube se llevaron a cabo modificaciones al modelo planteado por Abd-El-Barr y Gebali (2014), con el propósito de mantener coherencia en los modelos y cálculos que se realizan en este trabajó. Se diseñó una red 3-dimensional, con 8 nodos, de los cuales 6 nodos corresponden a los diferentes subsistemas y 2 nodos son utilizados de redundancia (Figura 4.6).

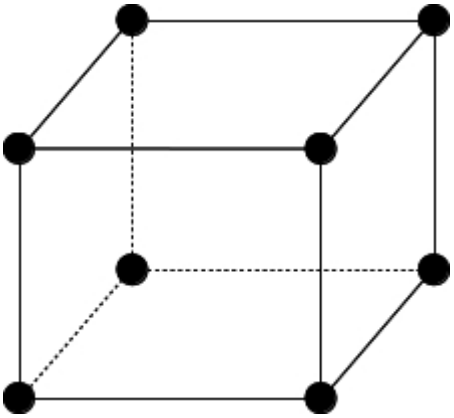

Figura 4.5: Red Hypercube

En este trabajo de tesis, la confiabilidad se calculó por medio del siguiente modelo:

$$R_{sys} = 1 - [N(1 - e^{-\lambda t}]$$

Como se puede observar el modelo es similar al modelo de árboles binario. Como se pudo estudiar en la bibliografía, árboles binarios y redes hypercube presentan varias similitudes, incluso se emebebe árboles binarios en este tipo de red. La *curva de confiabilidad* de esta red se la puede observar en la fgura

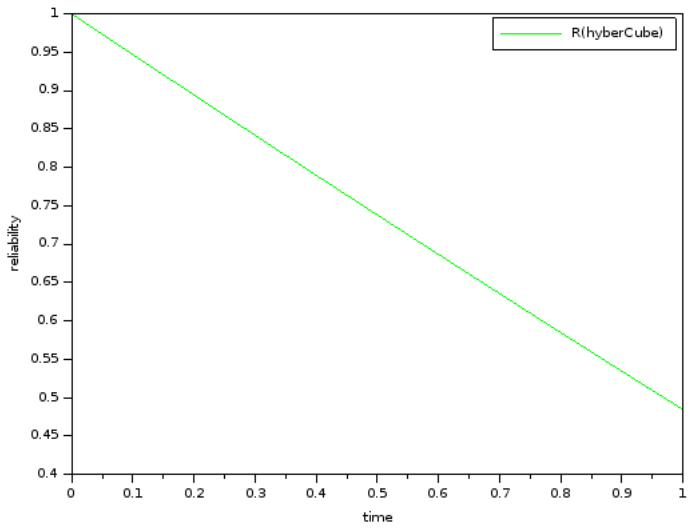

Figura 4.6: Confiabilidad de red hypercube

4.5. Topología utilizada en la arquitectura a diseñar

Teniendo en cuenta los modelos presentados anteriormente, se procede a realizar una comparación de las diferentes curvas. El resultado de esto, permitiré conocer de manera analítica qué topología presenta una mayor tolerancia a fallas a travez del tiempo. Además, teniendo en cuenta lo estudiado en el estado de la cuestión, se puede realizar una comparación conceptual de las tres topologías.

La aplicación de árboles binario podríar resultar una buena opción, ya que representa un desarrollo sencillo. En contraposición se puede indicar que existe un alto grado riesgo de que se produzca una falla en el nodo raíz y de su redundancia, lo cual pondría en peligro la misión. Así mismo, presenta otro punto negativo que se puede mencionar y es la gran cantidad de enlaces que esta necesita para mantener a todos los nodos de la red conectados.

Por otra parte, la red distribuída cuenta con la capacidad de distribuir el trabajo en todos sus nodos. Esto quiere decir, que si se produce una falla irrecuperable en cualquiera de sus nodos, la arquitectura podría continuar funcionando sin verse afectada por la ausencia de dicho nodo. Esto demanda un procesamiento computacional extra, y la necesidad de desarrollar algorimtos de ruteo especiales. Además, como punto negativo se puede mencionar que también, al igual que los árboles binarios, necesitan una gran cantidad de enlaces.

Por último, las topologías hypercube tiene un excelente respaldo teórico, exigen menor cantidad de enlaces, y pueden tolerar la falla de una gran cantidad de nodos (hasta el 50 % de los nodos). Su complejidad aumenta en gran medida, cuando se desarrollan arquitecturas de más dimensiones, sin embargo, esto aumenta su confiabilidad.

Como se mencionó en el primer párrafo, se realizó una comparación de módelos de confiabilidad de las tres topologías estudiadas. Se asumió que la distribución de la confiabilidad es exponencial, con una tasa de falla fija de 1/15, y se estudió su evolución en un rango $[0,1]t$. El resultado de esta comparación se la puede observar en la Figura 4.7 y en la Tabla 4.1.

Se observa que tanto las redes distribuidas como la hypercube presenta una mayor confiabilidad a través del tiempo que los árboles binarios. Si bien, las redes distribuidas y la hypercube tienen una curva similar, la primera presenta una mayor confiabilidad sostenida en el tiempo.

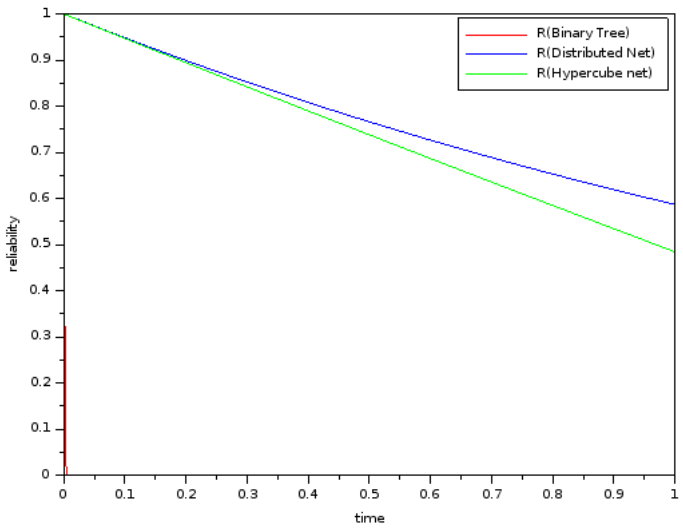

Figura 4.7: Comparación de confiabilidad

Tabla 4.1: Comparación de confiabilidad de topologías

	Topologías de red		
T	Tree Net	Distr Net	Hyper Net
0	1	1	1
0.001	0.803766	0.999947	0.999947
0.002	0.64604	0.999893	0.999893
0.003	0.519265	0.99984	0.99984
0.004	0.417368	0.999787	0.999787
0.005	0.335466	0.999733	0.999733
...	...	...	...
0.995	0	0.948317	0.947109
0.996	0	0.948266	0.947056
0.997	0	0.948216	0.947003
0.998	0	0.948165	0.94695
0.999	0	0.948115	0.946897

Con esto se puede concluir que la topología que presenta un mayor grado de confiabilidad es la que responde a una filosofía distribuida (bajo las condiciones en las que fueron estudiadas). Por lo tanto, la arquitectura satelital, tolerante a fallas y basada en componentes COTS que se desarrolla en la presenta tesis se basa en una **topología distribuída** para interconectar los diferentes subsistemas.

4.6. Topología propuesta

Sobre la base de los resultados presentados en (Arias y Wiman, 2017) se puede establecer que la topologóia propuesta es la más adecuada para el desarrollo de una arquitectura tolerante a fallas como se presenta en la Figura 4.8. En esta se puede observar que cada subsitema (térmico, power, telemetría, etc.) tiene su propia CPU controladora. Estas CPU se conectan a los nodos.

El modelo exige como requerimiento que cada nodo debe estar compuesto por una computadora (componente COTS) que es la encargada de realizar el procesamiento de las tareas. También, debe exitir un puente de comunicación entre la red y la CPU del subsistema. De este modo se hace frente a posibles fallas en la computadora del nodo. Esta conexión se observa en la Figura 4.9

4.7. Protocolo de comunicación

Aquí explico resumidamente el protocolo que “diseñe” en el apéndice A

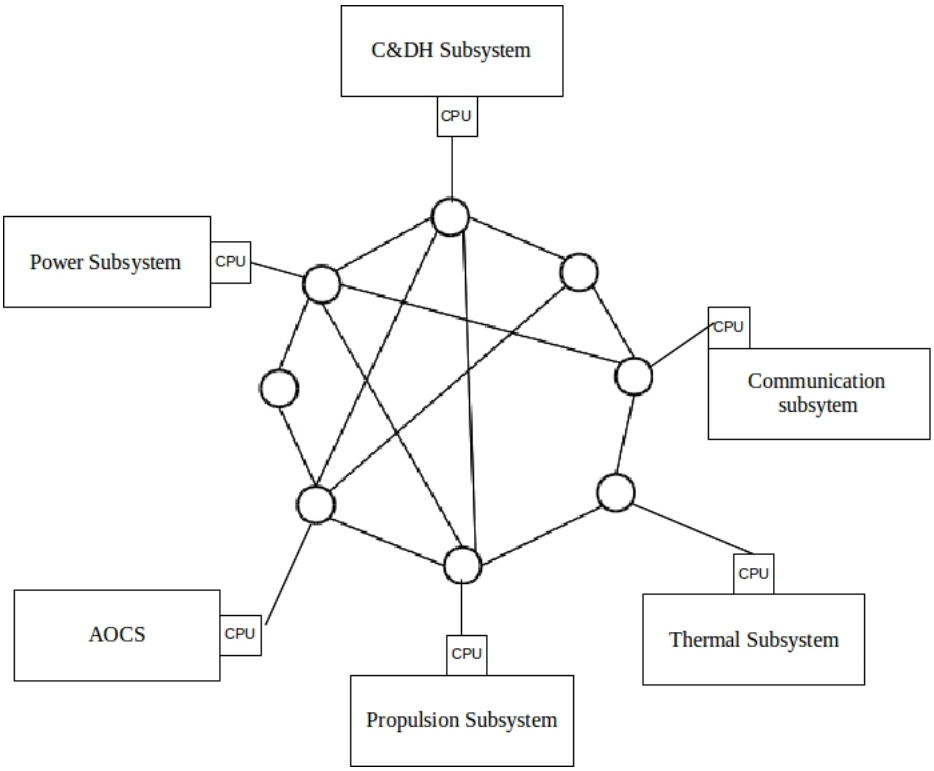

Figura 4.8: Arquitectura propuesta utilizando topología de red distribuida

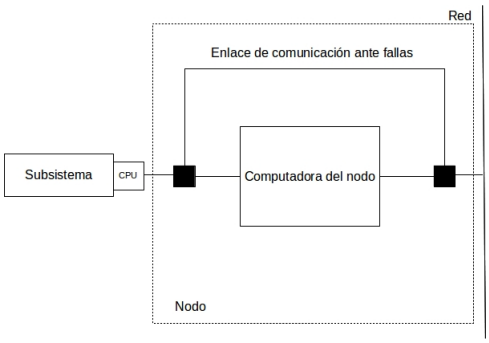

Figura 4.9: Conexión entre la red y el subsistema

Capítulo 5

Arquitectura propuesta

5.1. Introducción

Aquí hago la propuesta de la arquitectura.

El capítulo estará dividido en

- Arbol de requerimientos
- Diseño estructural (Diagramas de bloques, bloques internos)
- Diseño dinámico (Diagramas de secuencia, interacción, maquina de estados,)

Capítulo 6

Análisis y resultados

6.1. Introdución

Si llego a implementar algo (que no creo a esta altura) colocar aquí el código y los resultados.

Capítulo 7

Conclusión

Bibliografía

Mostafa Abd-El-Barr y Fayez Gebali. Reliability analysis and fault tolerance for hypercube multi-computer networks. *Information Sciences*, 276:295 – 318, 2014.

R. Alena, R. Gilstrap, J. Baldwin, T. Stone, y P. Wilson. Fault tolerance in ZigBee wireless sensor networks. En *Aerospace Conference, 2011 IEEE*, páginas 1–15, 2011.

Thomas Anderson y John C. Knight. A Framework for Software Fault Tolerance in Real-Time Systems. *IEEE Trans. Software Eng.*, (3):355–364, 1983.

Emmanuel Arias y Gustavo Wiman. Estudio de la confiabilidad de arquitecturas tolerantes a fallas basada en componentes cots para aviónicas de vehículos espaciales. Asociación Argentina de Tecnología Espacial - Instituto Universitario Aeronautico, Apr 2017.

Klaus Becker y Sebastian Voss. *A Formal Model and Analysis of Feature Degradation in Fault-Tolerant Systems*. Springer International Publishing, 2016.

CAN-CIA. CAN-CIA, 11 de abril del 2017. URL `https://www.can-cia.org`.

Savio. N. Chau, L. Alkalai, Ann T. Tai, y J. B. Burt. Design of a fault-tolerant COTS-based bus architecture. *IEEE Transactions on Reliability*, 48(4):351–359, 1999. ISSN 0018-9529.

Chau, Savio N. and Smith, Joseph and Tai, Ann T. A design-diversity based fault-tolerant cots avionics bus network. En *Dependable Computing, 2001. Proceedings. 2001 Pacific Rim International Symposium on*, páginas 35–42, 2001.

Steve Corrigan. Introduction to the Controller Area Network (CAN). Technical Report SLOA101A, Texas Instruments, 2002.

Steve Corrigan. Controller Area Network Physicalnn Layer Requeriments - ICP, Industrial Interface. Technical Report SLLA270, Texas Instruments, 2008.

Edward Crawley, Olivier de Weck, Steven Eppinger, Christopher Magee, Joel Moses, Warren Seering, Joel Schindall, David Wallace, y Daniel Whitney. Engineering Systems Monograph, The influence of architecture in engineering systems. 2004.

Dr Ralph D. Lorenz (auth.) David M. Harland. *Space Systems Failures: Disasters and Rescues of Satellites, Rockets and Space Probes*. Springer Praxis Books. Praxis, 1 edition, 2005. ISBN 978-0-387-21519-8,978-0-387-27961-9.

Douglas Isbell and Don Savage. Mars Climate Orbiter Failure Report - NASA, 1999.

Elena Dubrova. *Fault-Tolerant Design: an introduce*. Springer-Verlag New York, New York, U.S.A., 1 edition, 2013.

Guillaume Jacques Joseph Ducard. *Fault-Tolerant Flight Control and Guidance Systems for a Small Unmanned Aerial Vehicle*. Tesis de Doctorado, Swiss Federal Institue of Technology Zurich, 2007.

Cristopher Edwards, Thomas Lobaerts, y Hafid Smaili. *Fault Tolerant Flight Control. A benchmarck challenge*. Springer-Verlag, 2010. ISBN 978-3-642-11689-6.

Jens Eickhoff. *Onboard Computers, Onboard Software and Satellite Operations*. Springer-Verlag Berlin Heidelberg, Alemania, 2012. ISBN 978-3-642-25169-6.

ESD Electronics. ESD Electronics, 2017. URL `http://www.esd-electronics-usa.com/Controller-Area-Network-CAN-Introduction.html`.

S. Esposito, C. Albanese, M. Alderighi, F. Casini, Giganti L., M. L. Esposti, C. Monteleone, y M. Violante. Cots-based high-performance computing for space applications. *IEEE Transactions on Nuclear Science*, 62 (6):2687–2694, 2015.

Marie-Aude. Esteve, Joost-Pieter Katoen, VienY. Nguyen, Bart Postma, y Yuri Yushtein. Formal correctness, safety, dependability, and performance analysis of a satellite. páginas 1022–1031, 2012. doi: 10.1109/ICSE.2012.6227118.

J. S. Eterno, J. L. Weiss, D. P. Looze, y A. Willsky. Design issues for fault tolerant-restructurable aircraft control. En *Decision and Control, 1985 24th IEEE Conference on*, páginas 900–905, 1985.

Kevin Forsberg y Mooz Harold. 4 System Engineering for Faster, Cheaper, Better. *INCOSE International Symposium*, 9(1):924–932, 1999. ISSN 2334-5837.

Peter Fortescue, John Stark, y Graham Swinerd. *Space Systems Engineering*. Wiley, West Sussex, England, 3 edition, 2003. ISBN 0171619515.

Peter Fortescue, John Stark, y Graham Swinerd. *Space Systems Engineering*. Springer International Publishing, 2016.

Sanford Friedenthal, Alan Moore, y Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

W. C. Gangloff. Common mode failure analysis. *IEEE Transactions on Power Apparatus and Systems*, 94(1): 27–30, 1975. ISSN 0018-9510.

Hamilton, Deirdre L. and Walker, Ian D. and Bennett, John K. Fault tolerance versus performance metrics for robot systems. *Reliability Engineering & System Safety*, 1999.

Robert S. Hanmer. *Patterns for Fault Tolerant Software*. John Wiley and Sons Ltd, England, 2007. ISBN 978-0-470-31979-6.

John P. Hayes. A graph model for fault-tolerant computer systems. *IEEE Transactions on Computers*, 1976.

Hess, Ronald and Vetter, T. K. and Wells, S. R. Design and evaluation of a damage-tolerant flight control system. En *Institution of Mechanical Engineers Part G Journal of Aerospace Engineering*, 2005.

Ellis F. Hitt y Dennis Mulcare. Fault-Tolerant Avionics. En Cary R. Spitzer, editor, *Avionics Development and Implementation Second Edition*, capítulo 8. CRC Press, Williamsburg, Virginia, U.S.A., 2070.

Jon Holt y Simon Perry. *SysML for systems engineering.* The Institution of Engineering and Technology, London, United Kingdom, 2008. ISBN 978086341825.

K.A. Hoque, O.A. Mohamed, y Y. Savaria. Towards an accurate reliability, availability and maintainability analysis approach for satellite systems based on probabilistic model checking. volumen 2015-April, páginas 1635–1640, 2015.

IEEE. IEEE Standard Glossary of Software Engineering Terminology. Std. 610.12-1990, IEEE, 1990.

INVAP. INVAP Sociedad del Estado, 2016. URL `http://www.invap.com.ar/es/`.

Baback A. Izadi y Füsun Özgüner. An augmented k-ary tree multiprocessor with real-time fault-tolerant capability. *The Journal of Supercomputing*, 27(1):5–17, 2004.

Jim Krodel y George Romanski. Handbook for real-timte operating systems integration and component integration considerations in integrated modular avionics systems. Techn. Rep. DOT/FAA/AR-07/48, U.S. Department of Transportation, 2008.

Kvaser. Kvaser, 2017. URL `https://www.kvaser.com/can-protocol-tutorial/`.

Jens C. Lisner. *A Fault-tolerant Dynamic Time-trigged Protocol*. Tesis de Doctorado, Dependability of Computing Systems - University of Duisburg-Essen, 4 2007.

Yu Liu, Duo Li, y Chao Guo. Software reliability modeling with fault detection data when knowing fault severity. páginas 558–562, 2014. doi: 10.1109/ICRMS.2014.7107257.

Loveless, Andrew T. capítulo On TTEthernet for Integrated Fault-Tolerant Spacecraft Network. AIAA SPACE Forum. American Institute of Aeronautics and Astronautics, Aug 2015. doi: 10.2514/6.2015-4526. 0.

Michael R. Lyu. *Software Fault Tolerance*. John Wiley and Sons Ltd, Chichester, New York, USA, 1995. ISBN 9780471950684.

Pignol M. Dmt and dt2: two fault-tolerant architectures developed by cnes for cots-based spacecraft supercomputers. En *12th IEEE International On-Line Testing Symposium (IOLTS'06)*, 2006.

MARTE. `http://www.omgmarte.org/`, 3 de Junio de 2016.

Jackson R. Mayo, Robert C. Armstrong, y Geoffrey C. Hulette. *Leveraging Abstraction to Establish Out-of-Nominal Safety Properties*. Springer International Publishing, 2016.

Victor P. Nelson. Fault-tolerant computing: fundamental concepts. *Computer*, 23(7):19–25, 1990.

Papyrus. `http://eclipse.org/papyrus/`, 3 de Junio de 2016.

Zhaoguang Peng, Yu Lu, Alice Miller, Chris Johnson, y Tingdi Zhao. A probabilistic model checking approach to analysing reliability, availability, and maintainability of a single satellite system. páginas 611–616, 2013. doi: 10.1109/EMS.2013.102.

Dhiraj K. Pradhan y Sudhakar M. Reddy. A Fault-Tolerant Communication Architecture for Distributed Systems. *IEEE Transactions on Computers*, C-31(9):863–870, Sept 1982. ISSN 0018-9340.

Roger S. Pressman. *Software Engineering, A Practioner's Approach*. McGraw-Hill, fifth edition edition, 2001.

Laura Pullum. *Software fault tolerance techniques and implementation*. Artech House, England, 2001. ISBN 1-58053-137-7.

C. S. Raghavendra, AVIvizienis A., y M. D. Ercegovac. Fault tolerance in binary tree architectures. *IEEE Transactions on Computers*, (6):568–572, 1984.

M. Rausand y A. Hoyland. *System Reliability Theory. Models, statistical methods, and applications*. John Wiley Sons, Inc., Hoboken, New Jersey, USA, 2 edition, 2004.

RTI. *RTI Purchase Order Terms and Conditions v1.12. Spanish Translation*. RTI International, 3040 East Cornwallis Road, Research Triangle Park, USA, 2015.

Norman F. Schneidewind. Reliability modeling for safety-critical software. *IEEE Transactions on Reliability*, 46(1):88–98, 1997.

Norman F. Schneidewind y Allen P. Nikora. Issues and methods for assessing cots reliability, maintainability, and availability. 1998.

Adit D. Singh y Hee Y. Youn. A modular fault-tolerant binary tree architecture with short links. *IEEE Transactions on Computers*, 1991.

Steiner, Willfried. An Introdution to TTEthernet. TTTech Computertechnik AG, Apr 2013.

Constantine Stivaros. A measure of fault-tolerance for distributed networks. En *Computing and Information, 1992. Proceedings. ICCI '92., Fourth International Conference on*, páginas 426–429, 1992.

T. Stone, R. Alena, J. Baldwin, y P. Wilson. A viable cots based wireless architecture for spacecraft avionics. En *Aerospace Conference, 2012 IEEE*, páginas 1–11, 2012.

SysML. `http://sysml.org/`, 2 de Junio de 2016.

A. T. Tai, S. N. Chau, y L. Alkalai. Cots-based fault tolerance in deep space: Qualitative and quantitative analyses of a bus network architecture. En *High-Assurance Systems Engineering, 1999. Proceedings. 4th IEEE International Symposium on*, páginas 97–104, 1999.

Andrew S. Tanenbaum y David J. Wetherall. *Redes de computadora*. Pearson Educación, Mexico, quinta edition, 2012.

A.S. Tanenbaum. *Redes de computadoras*. Editorial Alhambra S. A. (SP), 2003. ISBN 9789702601623.

Wilfredo Torres-Pomales. Software Fault Tolerance: A tutorial. Technical Report NASA/TM-2000-210616, National Aeronautics and Space Administration Langley Research Center, Hampton, Virginia, 2000.

TTTech. Technical report, TTTech Computertechnik AG, Vienna, Austria.

TTTech. `http://tttech.com/`, 20 de Enero del 2017.

Christopher B. Watkins y Randy Walter. Transitioning from federated avionics architectures to integrated modular avionics. En *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, 2007.

Dave Woerner, Les Deutsch, y Chris Salvo. The X2000 Program: An Institutional Approach to Enabling Smaller Spacecraft. 2000.

Cong Zhang, I. M. Jaimoukha, y F. R. S. Sevilla. Fault-tolerant observer design with a tolerance measure for systems with sensor failures. En *2016 American Control Conference (ACC)*, páginas 7523–7528, 2016.

Youmin Zhang y Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252, 2008.

Apéndice A

Protocolo de comunicación: CANae 0.1 Alpha Version

A.1. Introducción

En este apéndice se explica el diseño de la versión Alpha del protocolo CANae. Este protocolo se basa en el la capa de aplicación del protocolo CAN y CANopen. El nombre CANae proviende de la unión de CAN y CONAE.

A.2. Capa de aplicación

La capa de aplicación de CANae está dividida entre la denominada *CANae Application Layer* y la *CANae High Application Layer* FiguraA.1. La funcionalidad que brinda la capa de aplicación está dividad en diferentes elementos de servicios dentro del a capa de aplicación. Un elemento de servicios brinda un determinado servicio.

Una capa de aplicación tiene las siguientes 4 funciones primitivas:

- Request: Una aplicación emite una petición a la capa de aplicación para solicitar un servicio.
- Indication: Emitido por la capa de aplicación con destino una aplicación, para reportar de algún eveno interno detectado por la capa de aplicación, o indicar que un servicio fue solicitado.
- Response: Emitido por la aplicación con destino la capa de aplicación para responder a una Indication previa.
- Confirm: Emitido por la capa de aplicación para reportar el resultado de una peteción previa.

A.3. Tipos de servicios de la capa de aplicación

Un tipo de servicio define las primitivas que intercambian las aplicaciones de usuario y los diferentes elementos de servicio que se encuentran dentro de la capa de aplicación. Los tipos de servicios posibles en CANae son los siguientes:

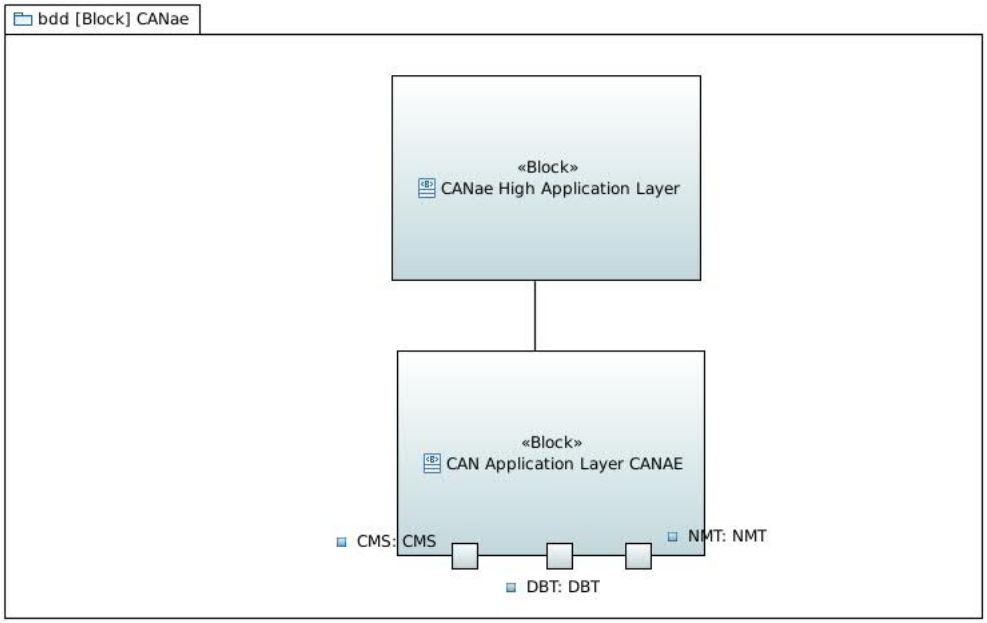

Figura A.1: Estructura de la capa de aplicación de CANae en alto nivel

- Local serivice: Solo envuelve un elemento de servicio local.
- Confirmed service: Implica que uno o más elementos de servicio pares. La aplicación de usuario emite un request a su local service element. Este se transmite a sus elementos de servicios pares, la cual es enviada a la aplicación como una indicación. La otra aplicación emite un response que se transmite a la aplicación originante para confirmar la recepción del request.
- Provide initiated service: Implica solo el elemento de servicio local. El elemento de servicio detecta un evento no solicitado por la aplicación y emite una Indication.

A.4. CANae Application Layer

En CANae se divide la tradicional capa de aplicación del modelo OSI en dos capas. En esta sección se estudia la *CANae Application Layer*. Esta capa está conformada por 3 elementos de servicio:

- CAN based Message Specification (CMS): ofrece un ambiente orientado a objecto para diseñar aplicaciones de usuario. Esta entidad ofrece variables y eventos, y específica como un módulo pude acceder a las interfaces de CAN.
- Network Managment (NMT): ofrece un ambiente orientado a objetos para permitir que un módulo (el NMT Master) se ocupe de la inicialización y posibles fallas de otros módulos (NMT Slaves).
- Distributor: ofrece el servicio para distribuir dinámicamente el identificador de para los diferentes nodos.

Estos están basados en la capa de aplicación de CAN para la industria (CAN in Automation [CiA] International Users and Manufacturers Group e.V. CAN Application Layer for Industrial Applications). Estos elementos de servicios determinan el 'qué' puede hacer la capa. Estos son representados como interfaces(Figura A.2).

Dentro de esta capa existen dos entidades importantes:

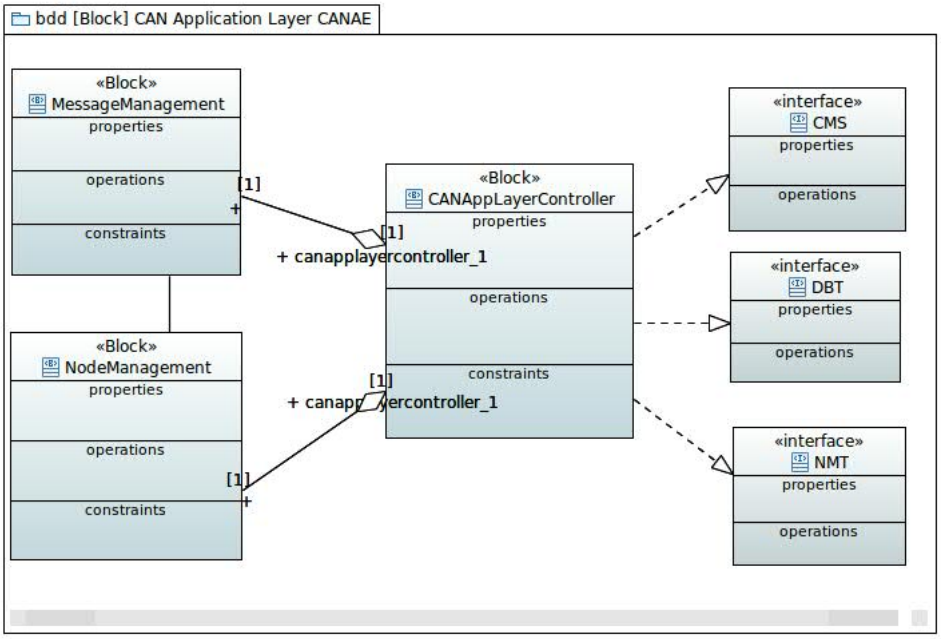

Figura A.2: CANae Application Layer

- Gestor de mensajes: este se encarga de gestionar los mensajes que son enviados y recibidos desde la red. Esta entidad debe ser capaz de determinar si el mensajes contiene datos o eventos. Trabaja en conjunto con el CMS.
- Gestor de nodo: esta entidad se encarga de llevar el control de los nodos existenten en la red. En este nodo se encuentra la tabla de ruteo primario y secundario necesarios para la correcta comunicación.

A.5. CMS

CMS ofrece un ambiente orientado a objeto para diseñar aplicaciones de usuario. CMS ofrece a la aplicación la posibilidad de modelar su comportamiento en la forma de objeto. Este elemento de servicio ofrece variables, eventos que se utilizan para diseñar y especificar como la funcionalidad de un módulo puede acceder a las interfaces CAN.

El servicio asume que no hay fallas provenientes de la capa de enlace de datos y la capa física de la red CAN. Las fallas, si ocurren, son resueltas por el NMSE (Network Management Service Element).

A.5.1. Prioridades de los objetos

La arbitración del protocolo CAN se realiza teniendo en cuenta la prioridad de los identificadores de los frames que se envía. La arbitración es no destructiva, debido a la necesidad de de tener un procesamiento de mensajes de tiempo real. La prioridad se determina teniendo en cuenta cada uno de los bits. El indentificador de los objetos y eventos de CMS está compuesto por 8 bits más 3 bits que los diferenciará entre objetos y eventos. Los eventos tiene mayor prioridad.

000 (ID) Tipo de evento
111 (ID) Prioridad objeto

La prioridad de los objetos es un UInteger de 1 byte. La prioridad más baja es 0 y la prioridad más alta de 1. Por lo tanto la prioridad va desde 00000000b (mayor prioridad) hasta 11111111b (menor prioridad).

A.5.2. Objeto CMS

Un objeto CMS define una estructura de dato que se debe respetar para poder enviar información (variables) a través de la red. El objeto está compuesto por los siguientes campos:

- Name: string de 6 caracteres (6 bytes)
- User_type: client, server (1 bit)
- Priority: UInteger (1 bytes)
- Datatype: UInteger identificador del tipo de datos (1 byte)
- Data: Indefinido.

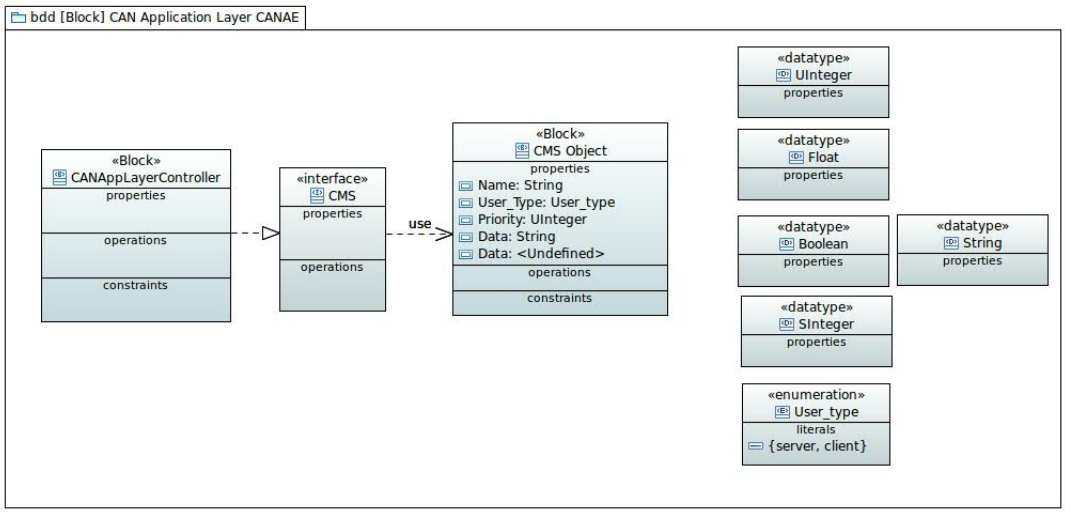

Figura A.3: Definición del Objeto CMS

A.5.3. Servicios de CMS

Con respecto a las variables de CMS existen una serie de servicios brindados por CMS, que pueden ser tanto *local services* como *remote services*. Su clasificación depende de si se está accediendo a variables almacenadas dentro del nodo, o bien se está intentando acceder (escribir o leer) una variable remota. Un evento está definido de la siguiente manera:

- Name: string de 6 caracteres (6 bytes)
- User_type: client, server (1 bit)
- Priority: UInteger (1 bytes)

- Event_Type: UInteger identificador del tipo de datos (1 byte)
- Data: Indefinido.

A.5.3.1. Servicios locales

- Boolean define_variable(String name, DataType data_type)
  - **String name**: Nombre de la variable.
  - **DataType data_type**: Tipo de dato de la variable.
- Boolean set_variable(String name, UInteger destination, Data data, UInteger priority)
  - **String name**: Nombre de la variable.
  - **UInteger destination**: Direción del destinatario. Esta dirección debe ser provista por el NMT.
  - **Data data**: Datos a enviar.
  - **UInteger priority**: Prioridad de la variable.
- Boolean update_variable(String name, data data)
  - **String name**: Nombre de la variable
  - **Data data**: Datos a enviar

A.5.3.2. Servicios remotos

- Boolean write_variable(String name, Data data)
  - **String name**: Nombre de la variable a escribir
  - **Data data**: Dato a escribir
- Data read_variable(String name)
  - **String name**: Nombre de la variable a leer

A.5.4. Eventos

Los eventos se utilizan para modelar comportamientos asíncronos, tales como temperatura excedida en un tiempo determinado. La ocurrencia de los eventos es detectada por el nodo, que en ese momento actuará como server y se notifica a todos los clientes. Los servicios al igual que las variables se dividen los eventos brindados por servicio local o servicio remoto. Los servicios se describen a continuación.

A.5.4.1. Servicios locales

- define_event(String name, UInteger priority, UInteger event_type)
  - **String name**: Nombre del evento
  - **UInteger priority**: Prioridad del evento
  - **UInteger event_type**: Tipo de evento según tabla A.1

A.5.4.2. Setvicios remotos

- Event notify_event()
- Boolean store_event(String name)
    - **String name**: Nombre de evento a almacenar
- Event read_event(String name)
    - **String name**: Nombre de la variable a leer

A.5.4.3. Prioridad de eventos

La prioridad de los eventos se define como una tabla que se define al momento de implementar el protocolo. Al desarrollar la tabla de prioridades se debe cuidar la relevancia de los eventos. La prioridad más baja es 0. El tamaño de la prioridad del evento es de 1 byte. Por lo tanto la prioridad va desde 00000000b (mayor prioridad) hasta 11111111b (menor prioridad).

Tabla A.1: Prioridad eventos

00000000	Nombre evento	Mayor prioridad
00000001	Nombre evento	
00000010	Nombre eevnto	
...	...	...
11111111	Nombre evento	Menor prioridad

Esta tabla de prioridad de eventos, debe ir acompañada de un diccionario de eventos, donde se almacena los datos relevantes al evento. En esta se hace una descripción del evento. Sirve de guía para la implementación y desarrollo de aplicaciones que identifican eventos.

El diccionario debería incluir los siguientes campos:

- Nombre del evento
- Descripción del evento
- Prioridad
- Umbral límite inferior (En el caso de que existiera)
- Umbral límite superior (En el caso de que existiera)

A.6. NMT

Esta es una entidad de la capa de aplicación que permite el correcto funcionamiento de la red. Esta entidad tiene los servicios necesarios para que cada nodo tenga un conocimiento de todos los nodos conectados en la red. Esto permite el armado de una tabla de nodos, que es mantenida en todo los nodos. Esta entidad exige la existencia de un nodo monitor, que es el encargado de monitorear y configurar la red. Luego, una vez en funcionamiento este nodo monitor no será vital para la red. En el NMT se deben implementar los algoritmos de ruteo. El entorno NMT se puede observar en la Figura A.5. Se puede observar también el diagrama de bloques internos en la Figura A.4

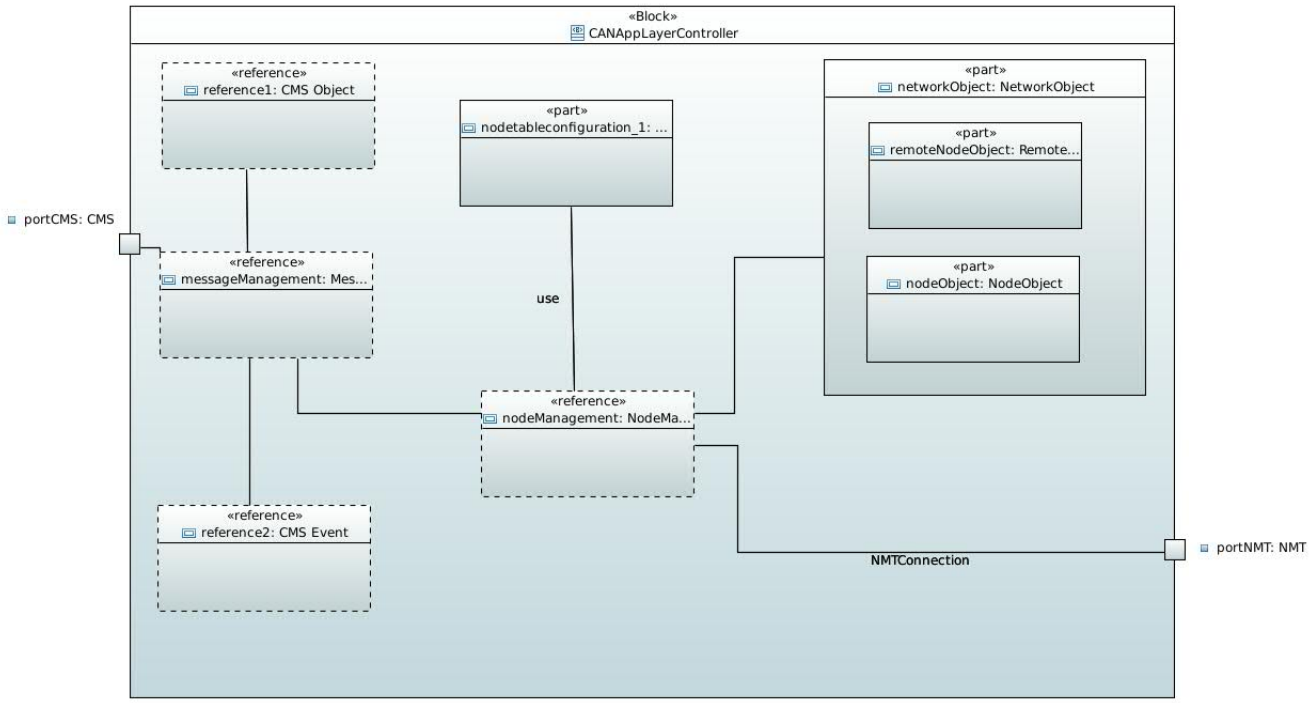

Figura A.4: Definición de la entidad NMT

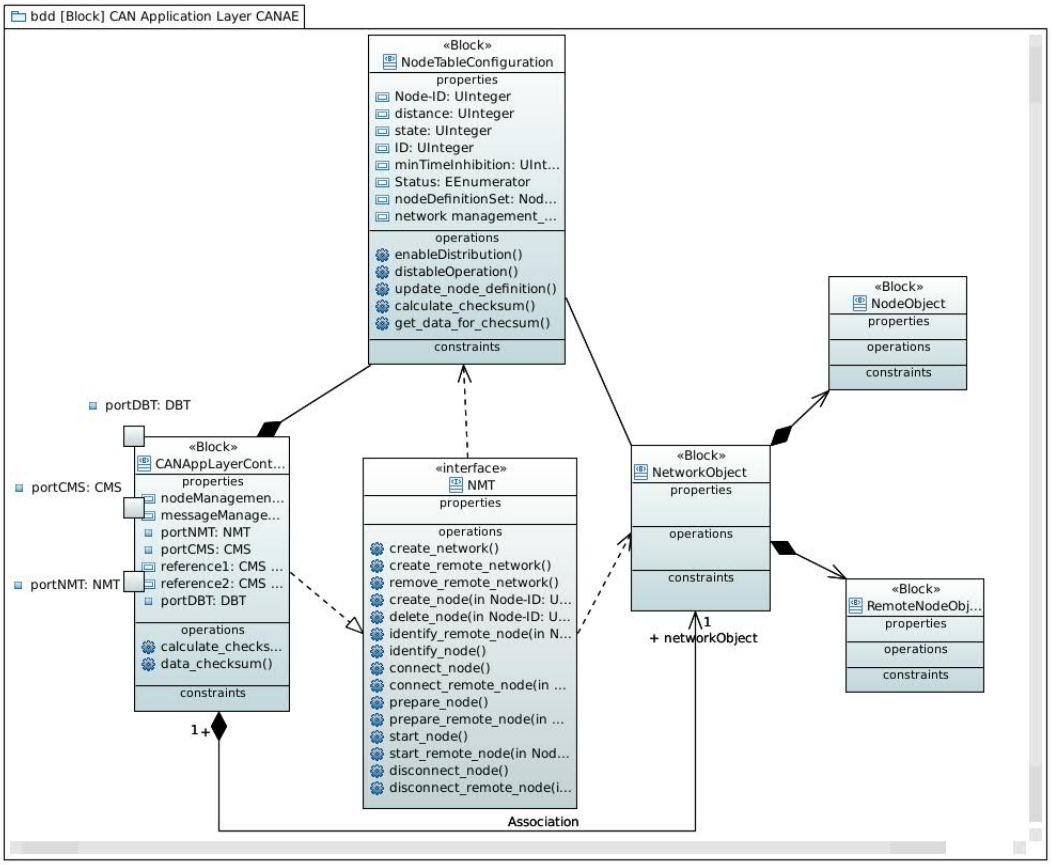

Figura A.5: Definición de la entidad NMT

Por medio de este servicio es posible conocer el estado de la red CAN. Los mensajes de NMT CAN son envíados con máxima prioridad.

A.6.1. Objectos NMT

El Network Management utiliza tres objetos diferentes para modelar una red CAN.

- **Network Object**: Representa todos los modulos conectados en la red CAN. El objeto de red existe en todos los nodos.
- **Remote Node Object**: Cada nodo conectado a la red CAN tiene representado todos los demás nodos.
- **Node Object**: Cada nodo que es gestionado por el servicio NMT está representado por un node object. Este objeto se encuentra modelado en cada uno de los nodos.

Cada nodo y su objeto (remoto y local) es unívocamente identificado en la red por su NMT Address. Esta dirección no se puede cambiar. Esto significa que por cada nodo existe un objeto nodo y un objeto nodo remoto con el mismo NMT Address, replicada en todos los nodos de la red.

A.6.2. Servicios NMT

La entidad NMT ofrece los siguientes servicios:

A.6.2.1. Module Control Service

EL NMT monitor incializa los nodos NMT que forman parte de la red CAN distribuida, a través de este servicio se asegura que todos los nodos se encuentran configurados y funcionando correctamente.

A.6.2.2. Error Control Services

Através de este servicio, el NMT detecta fallas en la red CAN, ya sea fallas producidas en la capa de enlace de datos (fallas locales) y/o fallas producidos en otros nodos.

A.6.2.3. Configuration Control Services

Através de este servicio se realiza la configuarción de la red.

A.6.2.4. Servicios a implementar

La funciones típicas que se deben implementar en el NMT son los siguientes:

- Boolean create_network()
- Boolean create_remote_network()
- Boolean remove_remote_network()

- Boolean create_node(UInteger Node-ID, UInteger address, String description)
  - **UInteger Node-ID**: ID del nodo a crear.
  - **UInteger address**: Dirección del nodo creado. UInteger entre [0,255]
  - **String description**: Descripción del nodo.
- Boolean delete_node(UInteger Node-ID)
  - **UInteger Node-ID**: Dirección del nodo a eliminar. UInteger entre [0,255]
- UInteger identify_remote_node(UInteger Node-ID)
  - **UInteger Node-ID**: Dirección del nodo a idetenficar
- UInteger identify_node()
- Boolean connect_node()
- Boolean connect_remote_node(UInteger Node-ID)
  - **UInteger Node-ID**: ID del nodo remoto a conectar.
- Boolean prepare_node()
- Boolean prepare_remote_node(UInteger Node-ID)
  - **UInteger Node-ID**: ID del nodo a pasar a modo preparado
- Boolean start_node()
- Boolean start_remote_node(UInteger Node-ID)
  - **UIntenger Node-ID**: ID del nodo que comenzará a participar en la red.
- Boolean disconnect_node()
- Boolean disconnect_remote_node(UInteger Node-ID)
  - **UInteger Node-ID**: ID del nodo a desconectar.

A.6.3. Protocolos NMT

A.6.3.1. Crear red CANae

Para conectar correctamente la red CANae se debe respetar el protocolo NMT para crear la red.En primer lugar el nodo monitor debe enviar el mensaje *create_remote_network()* para avisar a todos los nodos que deben crear sus propias instancias de *NetworkObject* a través de *create_network()*. Automáticamente, los nodos deben crear una propia instancia del nodo mediante *create_node()*. Luego el nodo monitor envía el mensaje de *prepare_remote_node()* para prepara todos los nodos a conectarse a la red CANae. Luego el nodo monitor envía la señal de *connect_remote_node(UInteger Node-ID)* con la dirección de todos los nodos que figuran en su *NodeTableConfiguration* preprogramada. Luego cada nodo, correctamente preparado, se conecta a la red a través de *connect_node()*.

Este protocolo puede ser observado en la Figura A.6

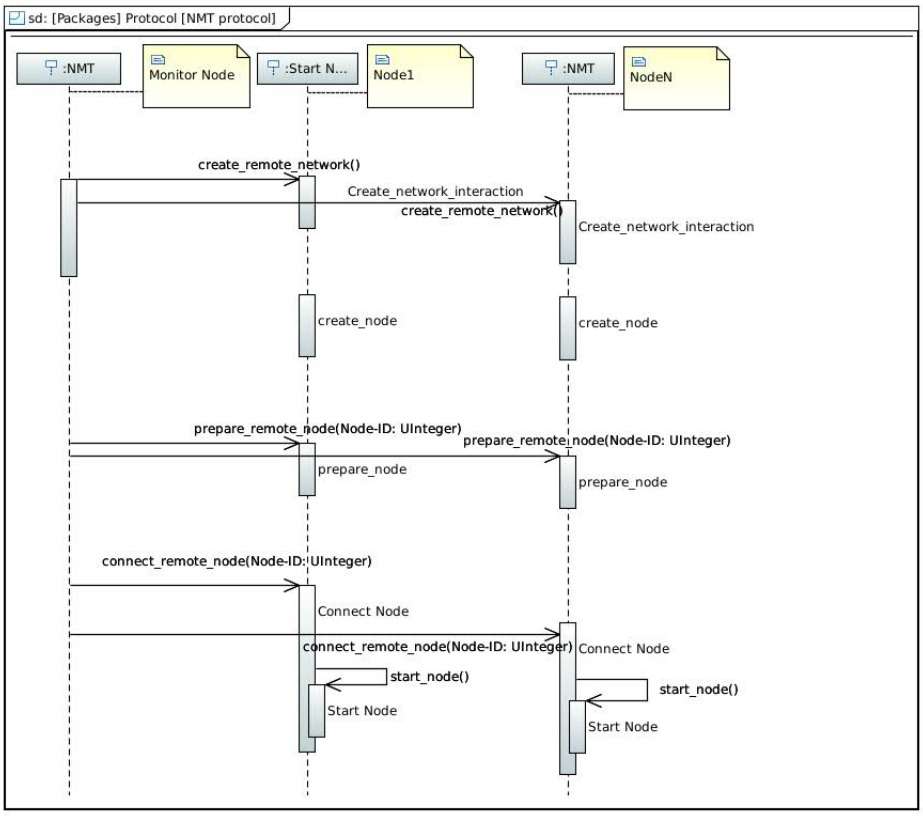

Figura A.6: Protocolo NMT

A.6.3.2. Crear Red

Para lograr la correcta conexión de los nodos a la red, estos deben configurarse. Cuando se alimenta la red, los nodos deben iniciar en estado de *escucha* esperando la orden del nodo monitor de crear la red mediante *create_remote_network()*. Esta se envía en broadcast a todos los nodos conectados. Cuando el controlador de la capa de aplicaciónr recibe el mensaje através del servicio NMT, este manda un mensaje al gesto del nodo interno, el cual crea la *tabla de configuración del nodo* y acutizaliza con la información necesaria.

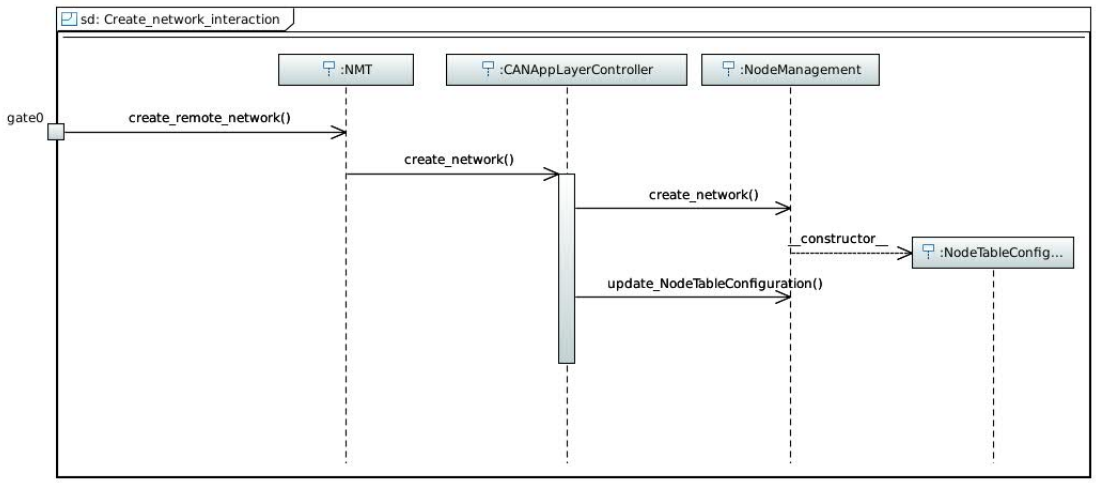

Figura A.7: Crear Red

A.6.3.3. Crear Objecto Red, Objeto Nodo y Objeto Nodo Remoto

Antes de comenzar a funcionar la red, cada nodo debe crear un objeto de red exigidos por el protocolo NMT. Una vez que el proceso Crear Red termina, el Gestor de la capa de Aplicación envía un mensaje *create_node()* al Gestor del Nodo. El Gestor de Nodo crea un *Network Object*. Este a su vez crea un *Node Object* y un *Remote Node Object*. Estos objetos pertenecen a la entidad NMT (Figura A.8).

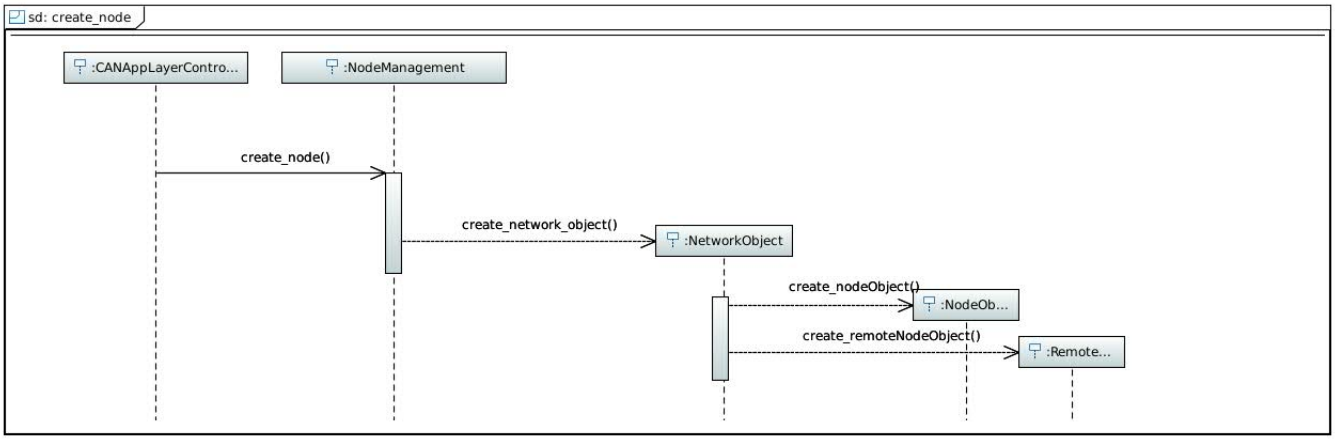

Figura A.8: Crear Objecto Red, Objeto Nodo y Objeto Nodo Remoto

A.6.3.4. Preparar nodos para conexión

Para lograr la conexión de los nodos a la red CAN, es necesario que el nodo monitor envía la señal *_prepare_remote_node(UInteger Nodo-ID)* a cada uno de los nodos conectados a la red siguiendo la propia tabla de configuración de nodos preprogramada. Cada nodo recibe la señal procesa aquella que le pertence, y comienza a preparar el nodo para lograr la correcta conexión. Para ello debe actualizar su propia tabal de configuración de nodos, como así también actualizar su *Objeto Red* (Figura A.9).

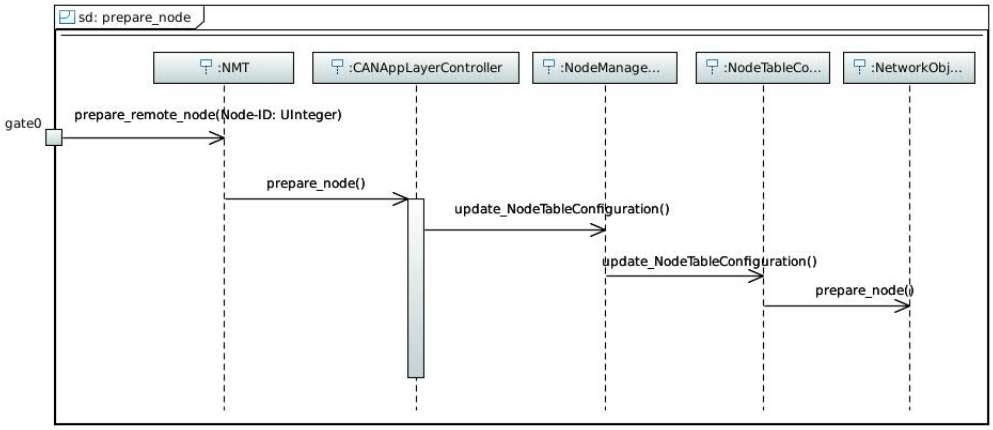

Figura A.9: Preparar nodo para la conexión a la red CAN

A.6.3.5. Conectar nodos en la red

Para conectar los nodos a al red se tiene que modificar al información de la *Node Table Configuration* y en el *Objeto de Red*. Una vez realizado esto se puede comenzar con la comunicación de los nodos (Figura A.10).

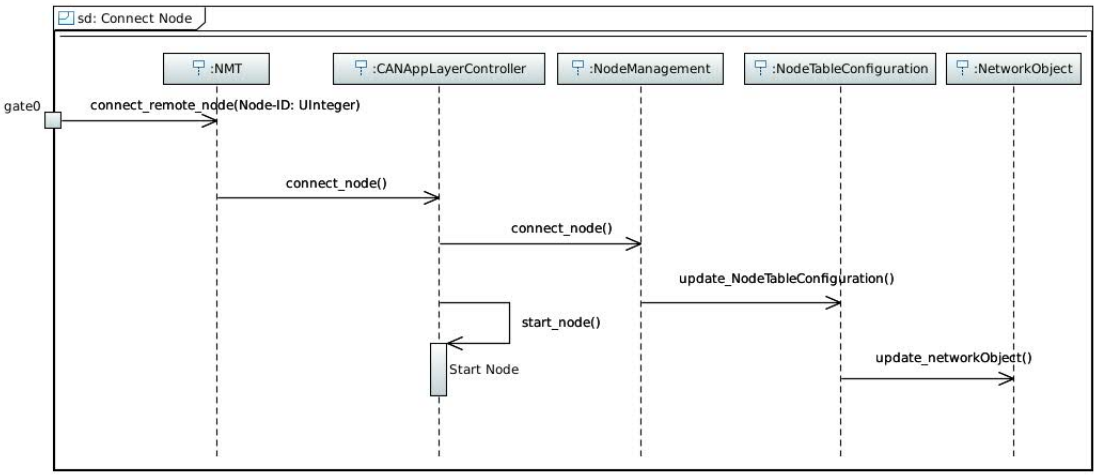

Figura A.10: Conectar el nodo a la red CAN

A.6.3.6. Comenzar la comunicación

Para que el nodo pueda comenzar a comunicar debe encontrarse en un estado *Start*. Esto representa una actualización de la *Node Table Configuration* y del *Objeto de Red* (Figura A.11).

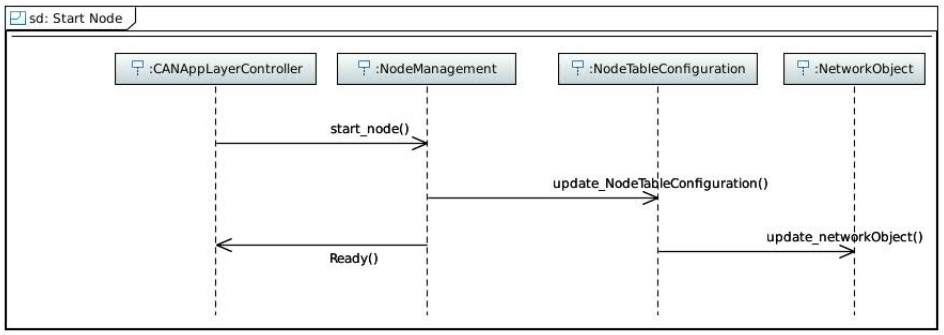

Figura A.11: Comenzar la comunicación de nodos

A.7. DBT

La principal complicación del desarrollo de una red distribuída basada en el BUS CAN, es que puedan comunicarse correctamente entre sí, cómo se deben asignar los identificadores (Node-ID) correctamente a cada uno de los nodos y cómo se asignan los tiempos de inhibición. Los Node-ID y los tiempos de inhibición se deben distribuir entre los nodos de tal manera de asegurar que:

- Se preveean los conflictos entre nodos. Por ejemplo diferentes funciones que usan el mismo identificador.

- Se preveean la falta de coincidencia. Por ejemplo que existan diferentes indentificadores para el mismo nodo.
- ofrecer el control integrado para un comportamiento dinámico del sistema.

El protocolo CAN (CAN Application Layer for Industrial Applications CiA/DS204-1) dispone la existencia de 3 métodos para distribuir identificadores y tiempo de inhibición a un módulo.

- **Distribución estándar**: en este método los identificadores y tiempos de inhibición son estandarizados por el módulo proveedor (nodo monitor. Una distribución estándar require la estandarización de todas las funciones y su correspondiente identificador.
- **Distibución estática**: En este método los identificadores y los tiempos de inhibición son fijos. Todos los identificadores y los tiempos de inhibición son fijados en el momento del desarrollo.
- **Distribución dinámica**: En este método los identificadores y los tiempos de inhibición son distribuidos vía red CAN a través del servicio estándar y un protocolo definido.

En esta instancia de trabajo, para el desarrollo del protocolo CANae, se utilizó el *método de distribución estática*. Esto exige que los nodos ya cuenten con un identificador y tiempos de inhibición por defecto. Esto se asigna en el momento del desarrollo del diseño e ingeniería de la arquitectura de aviónica. Esto reduce considerablemente la complejidad de la entidad DBT. Si bien originalmente, el DBT es un elemento de servicio de la capa de aplicación CAN que ofrece una distribución dinámica de identificadores y tiempos de inhibición, en CANae es utilizado para asegurar la correcta comunicación y consistencia de los nodos y la red en su conjunto.

La arquitectura del DBT se puede observar en la Figura A.12

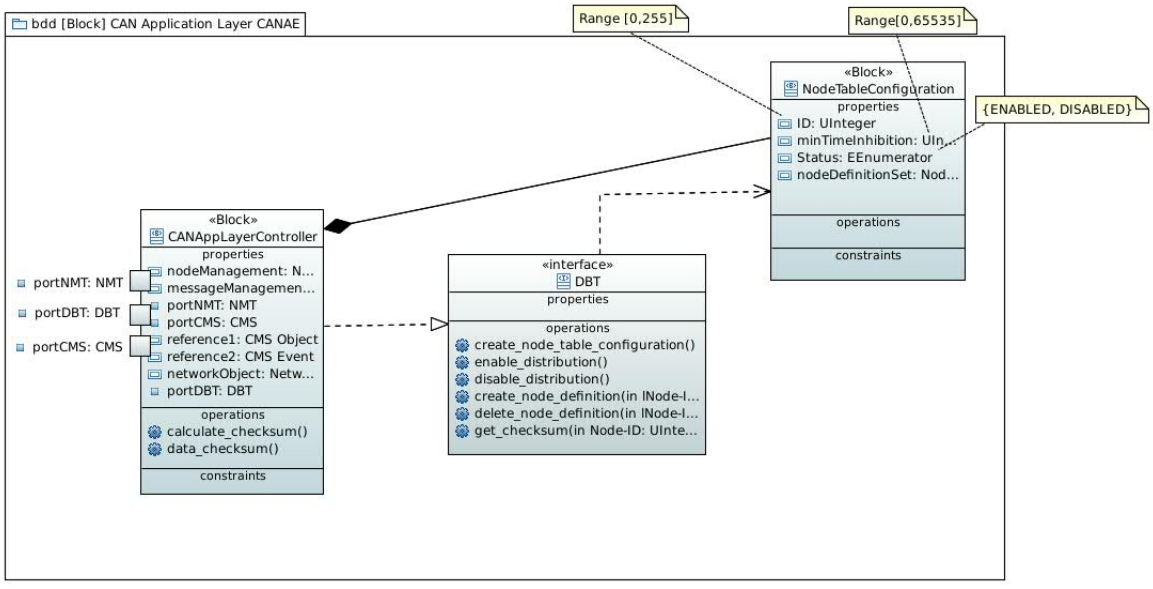

Figura A.12: Arquitectura de la entidad DBT

A.7.1. Objetos y servicios de DBT

El DBT de CANae utiliza 2 objetos para modelar su funcionamiento:

- **Node-ID Database (NodeTableConfiguration)**: Esta tabla contiene la definición de todos los nodos conectados a la red. Esta tabla está compuesta por Node-ID Definitions
- **Node-ID Definition**: define todos los atributos de un Node-ID. Contiene una definición de usuario creado por el usuario.

El DBT de CANae ofrece las siguientes categorias de servicio:

- **Distribution Control Services**: esta es tarea del nodo monitor de enviar el identificador y los tiempos de inhibición a todos los nodos conectados a la red. Los datos que son enviados por el nodo monitor son corroborados por los nodos, comprobando que el ID enviado coincida con su propio Node-ID.
- **Consistency Control Services**: através de este servicio cada nodo puede detectar inconsistencia en el NodeTableConfiguration e inconsistencia entre nodos.

A.7.2. Descripción de los servicios DBT

Los servicios se describen en forma de tabla que contiene los parámetros de cada función que se define para ese servicio. Los parámetros determinan el tipo de servicio. Todos los servicios asumen que no ocurrieron ningún tipo de error en la capa de física ni en la capa de enlace de datos.

A.7.3. Objetos DBT

Los objetos que utiliza la entidad DBT para modelar el servicio son los siguientes:

- **Node Table Configuration (Node-ID Database)**
  - **Atributos**
    - Estado: Uno de los valores ENABLED, DISABLED. Este atributo indica si el DBT Monitor es capaz de distribuir los Node-ID y tiempos de inhibición. En esta instancia de trabajo el valor del estado en el DBT del Nodo Monitor será DSIABLED
    - Node Definition Set set de todas las defincciones.
- **Node-ID Defintion**: Las defincciones contiene los siguientes datos:
  - ID: Valor en el rango de [0,255]
  - Mínimo tiempo de inhibición: en el rango [0,65535] indicando el valor mínimo en unidades de 100μsec para el tiempo de inhibición.

A.7.4. Servicios DBT

A.7.4.1. Distribution Control Services

- create_node_table_configuration()
- enable_distribution()
- disable_distribution()

- create_node_definition(UInteger lNode-ID, UInteger hNode-ID, UInteger minInhibitTime)
  - **UInteger lNode-ID**: Límite inferior del rango de IDs.
  - **UInteger hNode-ID**: Límite superior del rango de IDs.
  - **UInteger minInhibitTime**: Tiempo mínimo de inhibición de cada nodo.
- delete_node_definition(UInteger lNode-ID, UInteger hNode-ID)

A.7.4.2. Consistency Control Service

- get_checksum(UInteger Node-ID, Float checksum)
  - **UInteger Node-ID**: ID del nodo que desea controlar la consistencia.
  - **UInteger checksum**: checksum de la DBT Definition.

A.7.5. Protocolos DBT

A.7.5.1. Crear NodeTableConfiguration

El La tabla de configuración del nodo juega un papel similar al que lo haría el Node-ID DATABASE del estándar CAN. La principal diferencia, es que esta tabla se encuentra presente en todos los nodo y no en el "Master", ya que este protocolo está pensado para ser utilizado en redes distribuidas, donde no existe un nodo central o master.

El nodo monitor envía la señal a los nodos de crear la tabla de configuración del nodo através de *create_node_table_configuration()*. Este es recibido através de la interfaz DBT del nodo, y crea el objeto *NodeTableConfiguration*. Esto se observa en la Figura A.13.

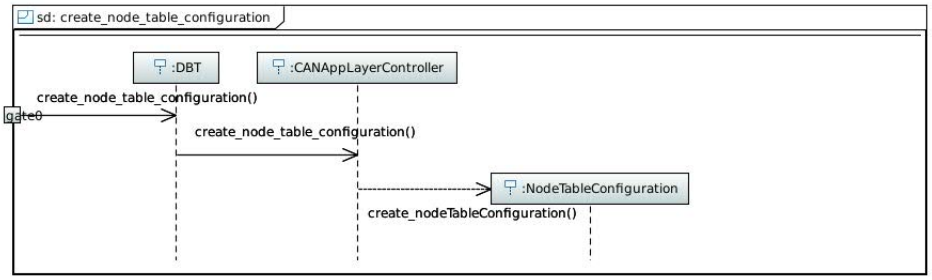

Figura A.13: Protocolo create_node_table_configuration

A.7.5.2. Habilitar distribución

La distribución de Node-ID es utilizada cuando se utiliza el método de asignación dinámica. En esta versión del protocolo la distribución del ID se encuentra desactivada. En la Figura A.14 se puede obseevar el protocolo.

A.7.5.3. Deshabilitar distribución

Esta opción se encuentra por defecto en esta versión del protocolo CANae. El ID se distribuye estáticamente. Son preestablecidos durante el desarrollo. En la Figura A.15

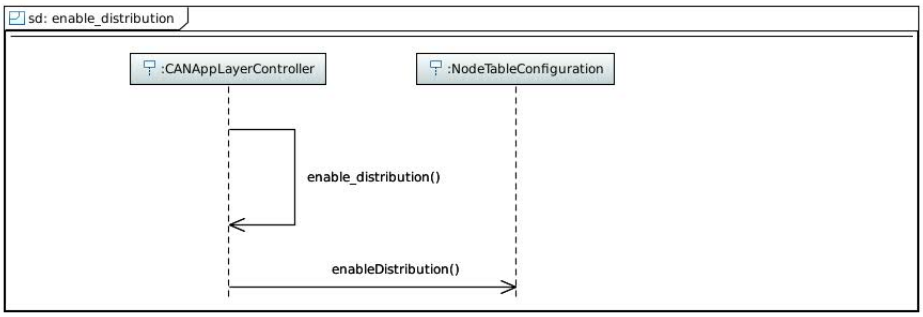

Figura A.14: Protocolo enable_distribution

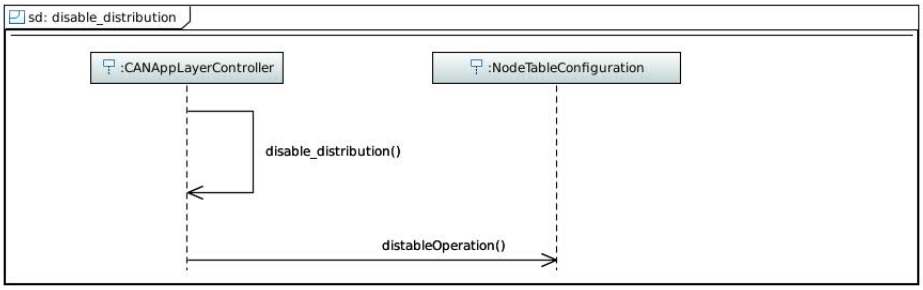

Figura A.15: Protocolo disable_distribution

A.7.5.4. Crear definición del nodo

El nodo monitor envía la señal para que sea creen las definiciones de nodo. Para ello es necesario que el nodo monitor envíe los rangos permitidos para la definición de los nodos. Esto se define a través de las variables *lNode-ID* y *hNode-ID*. En esta versión del protocolo estas variables no son utilizadas ya que los Node-ID ya se encuentran definidas estáticamente. También el nodo monitor envía el tiempo mínimo de inhibición explicado anteriormente. El nodo monitor hace este proceso con todos los nodos de la red a través del mensaje *create_node_definition*. Este se lo puede observar en la Figura A.16.

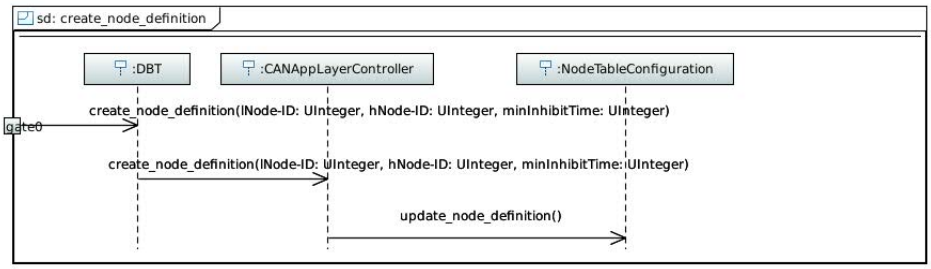

Figura A.16: Protocolo create_node_definition

A.7.5.5. Eliminar la definición del nodo

Esta función es utilizada para eliminar la defnición del nodo. Se lleva a cabo para eliminar un nodo de la red. Se hace através de la función *delete_node_definition()*. Esta se la puede observar en la Figura A.17

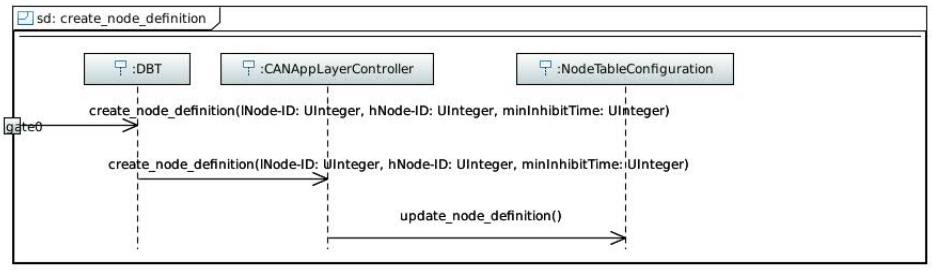

Figura A.17: Protocolo delete_node_definition

A.7.5.6. Checksum

Para poder asegurar la integridad de los nodos conectados a la red, se puede utilizar este mecanismo para comprobar de que la información, referente a la definición de los nodos, se encuentra correctamente en todos los nodos. Para ello, cada nodo lleva a cabo un checksum de su información y la envía por la red. Como esta está basada en el Bus-CAN, el mensaje llega correctamente a todos los nodos. Los nodos realizan sus comprobaciones y responden. Si la información es correcta deben responder con un mensaje que contegan bits NO dominantes. En el caso de que se produzca alguna diferencia en los checksum deberán responder con bits dominantes. Cuando esto se produzca deberán llevarse a cabo medidas de recuperación o reconfiguración de la red. Este protocolo se lo puede observar en la Figura A.18

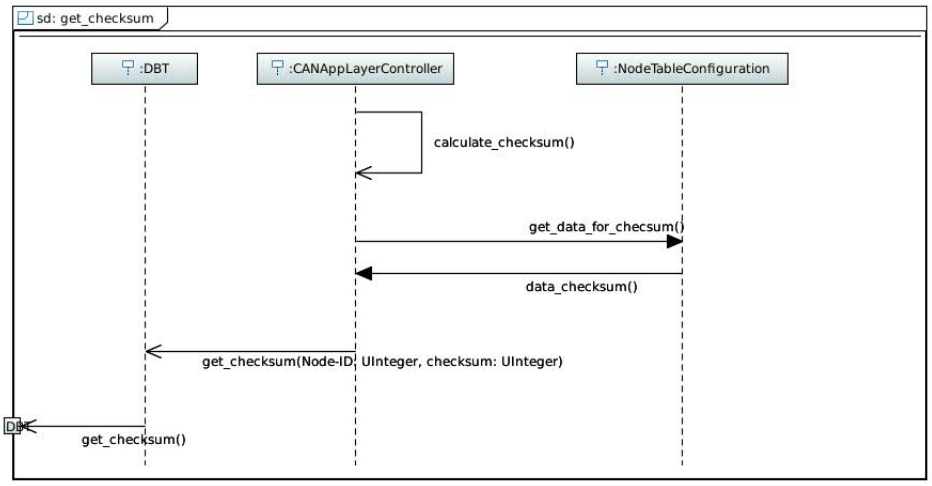

Figura A.18: Protocolo get_checksum

A.8. Gestor de mensajes

El **Message Management** se encarga de gestionar los mensajes que son enviados y recibidos desde la red CAN. Esta entidad debe ser capaz de determinar si el mensaje (enviado o recibido) contiene datos o eventos. El gestor de mensajes tiene cuatro entidades:

- *Receiver Manager*: es el encargado de recibir los mansajes de la capa inferior a traves de la interfaz NMT. Esta entidad recibe y despaquetiza los mensajes.
- *Prepared Message*: es el encargado de recibir los mensajes desde las capas superiores para ser enviado a la red.

- *Sorter Message*: es el encargado de clasificar los mensajes, ya sea aquellos que envía a la red CAN comos los que recibe de esta. Su principal función es verificar el tipo de mensaje que está recibiendo o enviando (datos o eventos). Los eventos tiene mayor prioridad que los datos y deben atenderse con urgencia.

- *Buffer Message*: este sirve de buffer para almacenar mensajes tanto que se envían como lo que se reciben. Este buffer es utilizado por el *Sorter Message*.

En la Figura A.19 se muestra la composición del gestor de mensajes. Por otra parte en la Figura A.20 muestra el diagrama interno del gestor de mensajes.

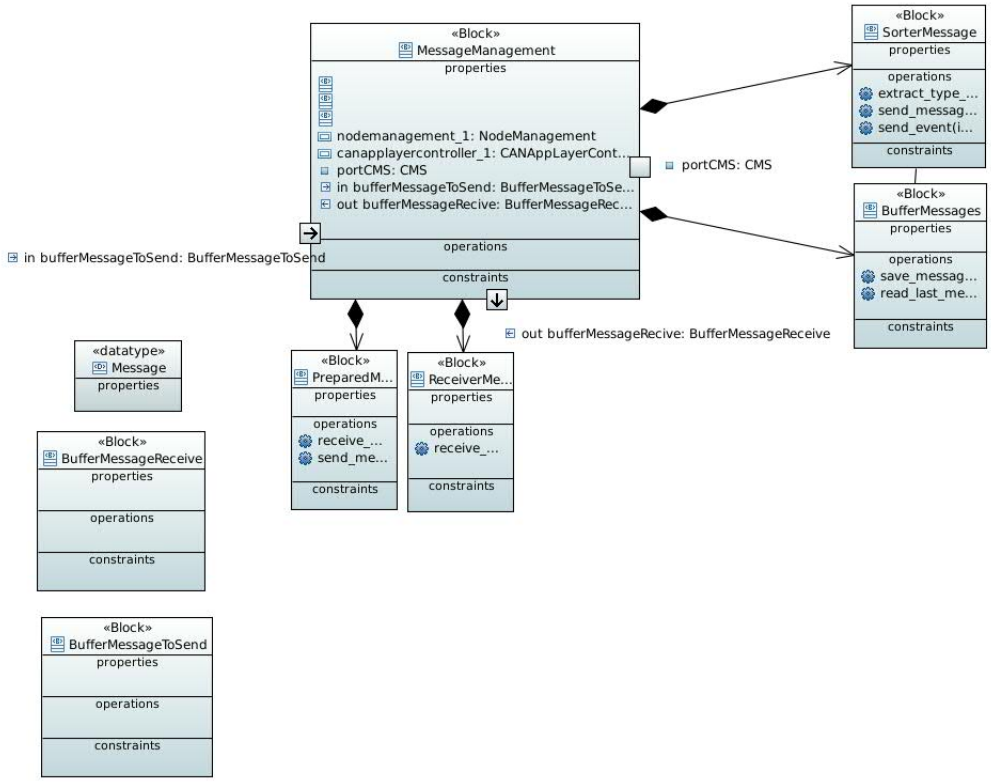

Figura A.19: Arquitectura de *Message Management*

A.8.1. Receiver Manager

Esta entidad perteneciente al *Message Management* se encarga de recibir los mensajes de la capa inmediatamente inferior a través de la interfaz NMT (y sus protocolos). Esta entidad recibe y despaquetiza los paquetes. Se supone que los mensajes llegan sin ningún tipo de error.

Las principales funciones de esta entidad son las siguientes:

- receive_message(Message message): esta es la operación que se debe llamar para recibir el mensaje proveniente de la capa inferior.

    - **Data message**: Es el mensaje que recibe de las capas inferiores de la red CAN.

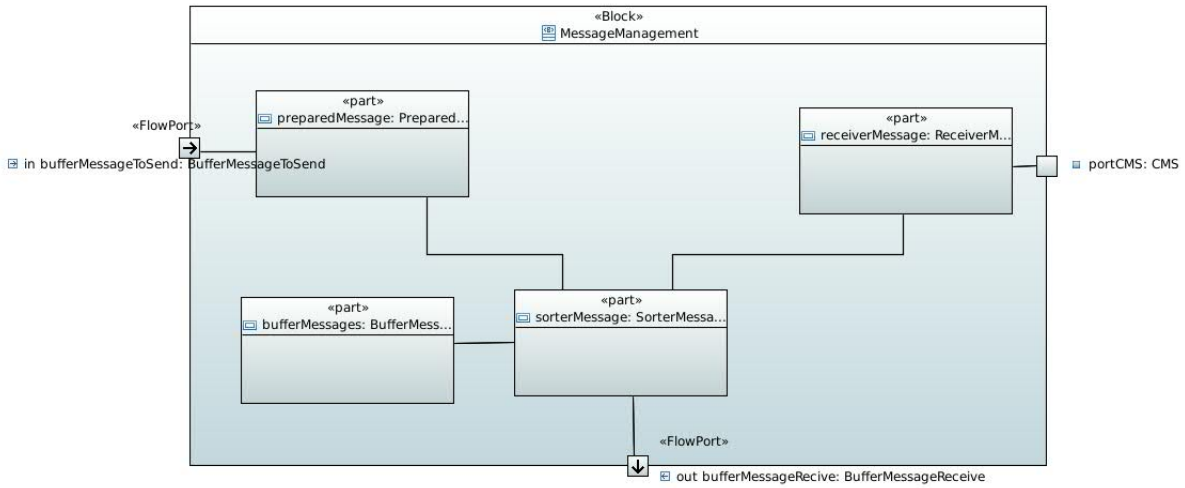

Figura A.20: Diagrama interno del *Message Management*

A.8.2. Prepared Message

Esta entidad es la que recibe los mensajes desde las capas superiores y lo envía al *Sorter Message*.

- receive_message(Message message): esta es la operación que se debe llamar para recibir el mensaje proveniente de la capa superior.
  - **Message message**: Es el mensaje que recibe de las capas inferiores de la red CAN.
- send_message(Message message): esta es la operación que se debe llamar para enviar el mensaje al *Sorter Message*.
  - **Message message**: Es el mensaje que se envía al *Sorter Message*

A.8.3. Sorter Message

Esta entidad es el encargado de clasificar los mensajes que provienen tanto del *Receiver Message* y *Prepared Message*. Este clasifica los mensajes recibidos o por enviar, de modo tal de conocer aquellos que sean datos o eventos. Debe tenerse en cuenta que los eventos tiene mayor prioridad que los datos, debido a que (en su mayoría) informan de algún problema a la red.

Las principales funciones con las que cuenta esta entidad son las siguiente:

- Boolean extract_type_message(Message message): esta operación permite extraer el tipo de información que contiene el mensaje: datos o evento.
  - **Message message**: es el mensaje que recibe de las capas inferiores de la red CAN.
  - **Return Boolean**: retorna 1 se se trata de un evento, 0 si se trata de un dato.
- Boolean send_message(Message message): esta función envía los mensajes a las capas inferiores a través del protocol NMT.
  - **Message message**: es el mensaje formateado listo para enviar.

- **Return Boolean**: retorna 1 si la operación se llevó correctamente.

- Boolean send_event(Message message): esta función envía el evento en forma de mensajes formateado.
  - **Message message**: es el mensaje formateado listo para enviar.
  - **Return Boolean**: retorna 1 si la operación se llevó correctamente.

A.8.4. Buffer Messages

Esta es la entidad que permite almacenar los mensajes que son recibidos y enviados. Esta entidad debe controlar la integridad de los mensajes que se encuentran almacenados. Además se encarga de escribir y leer los mensajes almacenados en el buffer. El buffer es FIFO[1].

Las principales funciones de esta entidad son las siguientes:

- save_message(Message message): esta función almacena el mensaje en el buffer de mensajes
  - **Message message**: es el mensaje formateado listo para enviar.
- Message read_last_message(): esta función lee el último mensaje.
  - **Return Message**: retorna el mensaje extraído del buffer.

A.8.5. Buffer Message Receive y Buffer Message To Send

La comunicación entre el Gestor de Mensajes y las capas, tanto superior como inferior, se realizan por medio de buffers. Se pueden dar las siguientes situaciones:

- Llegada de un mensaje de la capa superior para ser enviada a través de la red. En este caso la aplicación de usuario necesita enviar un mensaje, para ello debe colocar el mensaje en el *Buffer Message To Send*. El *Gestor de Mensaje* debe detectar (por medio de algún mecanismo) que se ha escrito en el buffer, para tomar el mensaje.
- Enviar un mensaje a la red. Cuando el gestor necesita enviar un mensaje lo tiene que almacenar en el *Buffer Message To Send*. Esto "alertaría" por medio de algún mecanismo a las capas inferiores, de que existe un nuevo mensaje a enviar

A.9. Gestor de Nodo

El *Node Management* es una entidad que se encuentra en el NMT, y es la encargada de llevar a cabo el control de los nodos existentes en la red. En esta entidad se encuentra almacenada la tabla de ruteo primario y secundario, utilizado para lograr la correcta comunicación y configuración de la red.

El objetivo del gestor del nodo, es conocer todos los nodos que se encuentran conectados en la red, y la distancia que existe entre el nodo actual y los demás.

Por medio de las tablas de ruteo (primaria o secundaria en caso de fallas) el nodo conoce a qué nodos puede enviar mensajes y a cuáles no. Debe aclararse que nada impide enviar un mensaje a un nodo "desconectado",

[1] First In First Out

pero esta desición es ineficiente y altamente peligrosa, ya que se puede estar enviando mensajes críticos a un nodo “caído”.

El gestor de nodo de cada dispositivo conectado a al red, es el encargado de llevar a cabo el protocolo *heartbeat*, el cual le indica a todos los nodos conectados a la red que este está “vivo”

En la Figura A.21 se puede observar la arquitectura del Gestor de Nodos.

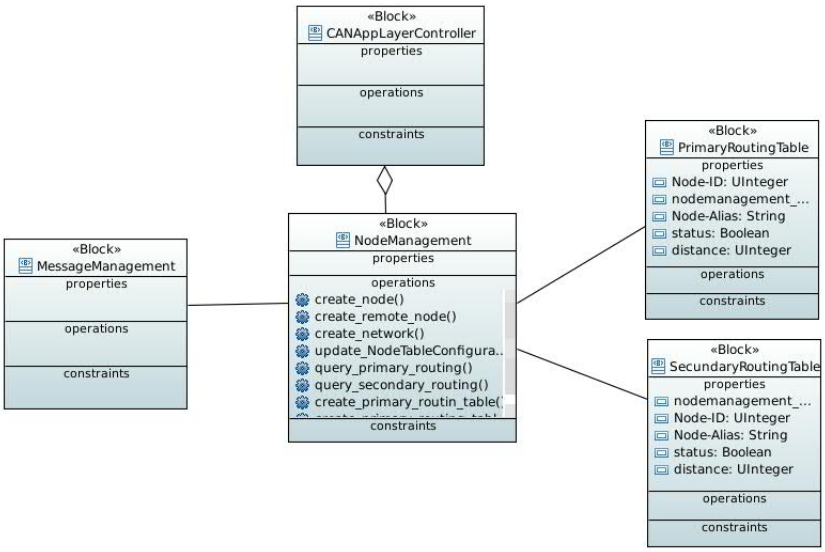

Figura A.21: Arquitectura del *Node Management*

A.9.1. Tabla primara y secundaria de ruteo

Los principales objetos del *Node Management* son las tablas de ruteo. Estas se las puede representar como una lista, en la cual se incluye información de importancia para mantener actualizada la estructura de la red. Los nodos se basarán sobre esta lista para tener conocimiento de los nodos conectados a la red y con esto, llevar a cabo una correcta distribución de las tareas.

Los algoritmos de ruteos se deben basar en estas tablas, ya que contienen toda la información necesarias para cumplir con sus objetivos.

A continuación se detalla la tabla primara y tabla secundaria de ruteo.

A.9.1.1. Tabla primaria

La Tabla Primaria de Ruteo es la tabla principal estática utilizada por el *Node Management* para la gestión de los nodos de la red. Esta tabla mantiene la a los n-iésimos nodos conectados a la red, desde el punto de vista del nodo “actual”. Es decir, para cada nodo existe una y sola una tabla, y esa tabla es única en la red, ya que es creada desde el punto de vista del nodo “host”.

La estructura de la lista es simple. Cada renglón de la lista es un nodo, siendo el primer renglón obligatoriamente el nodo de la tabla, llamado nodo “host”. Los siguientes renglones de la lista representan los demás nodos conectados a la red. Por convención, los nodos se listan comenzando por los vecinos de derecha a izquieda. Luego se continúa la lista tratando de seguir la convención de derecha a izquierda.

La tabla tiene la siguiente estructura:

- **UIntegerNode-ID**: Es el ID único para cada nodo.
- **String Node-Alias**: Es el alias para cada nodo. Este alias puede ser utilizado para comodidad por parte del usuario para referenciar los nodos.
- **UInteger Address**: Es la dirección del nodo.
- **Boolean Status**: Es el estado del nodo. Pude ser { CONNECTED, DISCONNECTED }.
- **UInteger Distance**: Es la distancia (cantidad de nodos) que hay entre el nodo "host" y los demás nodos de la red.

A.9.1.2. Tabla secundaria

La Tabla Secundaria de Ruteo es una tabla estática similar que la Tabla Primara de Ruteo, con la diferencia sustancial que es utilizada cuand hay fallas en ruteo. Esta tabla se genera cuando se producen fallas en uno o más nodos, y se necesita conocer la estructura de la red, para luego tomar desiciones sobre la configuración de la misma.

La estructura de la Tabla Secundaria de Ruteo es similar a la Tabla Primara de Ruteo. La diferencia sutancial es que no se encuentran todos los nodos de la red, sino aquellos que se encuentran "funcionales".

La estructura de la tabla se detalla a continuación:

- **UIntegerNode-ID**: Es el ID único para cada nodo.
- **String Node-Alias**: Es el alias para cada nodo. Este alias puede ser utilizado para comodidad por parte del usuario para referenciar los nodos.
- **UInteger Address**: Es la dirección del nodo.
- **Boolean Status**: Es el estado del nodo. Pude ser { CONNECTED, DISCONNECTED }.
- **UInteger Distance**: Es la distancia (cantidad de nodos) que hay entre el nodo "host" y los demás nodos de la red.

A.9.2. Servicios

Los servicios brindados por el *Node Management* son los siguientes:

- Boolean create_node(UInteger Node-ID, UInteger address, String description): esta función crear el objeto nodo. Este servicio ya fue descripto en A.6.2.4. Entre otras cosas, este servicio crea la primera entrada en la Tabla Primaria de Ruteo.
  - **UInteger Node-ID**: ID del nodo a crear.
  - **UInteger address**: Dirección del nodo creado. UInteger entre [0,255]
  - **String description**: Descripción del nodo.
- Boolean connect_remote_node(UInteger Node-ID): envía la señal a sus nodos vecinos para que creen una referencia de este nodo en los demás.
  - **UInteger Node-ID**: ID del nodo remoto a conectar.

- Boolean create_network(): esta función fue descrita en A.6.2.4. Este servicio crea una instancia de la Tabla Primara de Ruteo.
- Boolean update_NodeTableConfiguration(): esta función actualiza la tabla de configuración del nodo (Vista en: A.6.2.4).
- Data query_primary_routing(): esta función lleva a cabo una consulta a la Tabla Primaria de Ruteo. Se la utiliza cuando se necesita extraer datos de la configuración de la red.
- Data query_secundary_routing(): esta función lleva a cabo una consulta a la Tabla Secundaria de Ruteo. Se la utiliza cuando se necesita extraer datos de la configuración de la red luego de fallas en los nodos.
- Boolean add_node(UInteger Node-ID, String Node-Alias, Boolean status, UInteger distance, UInteger address): este servicio se lleva a cabo cuando llega el mensaje connect_remote_node(UInteger Node-ID). Este servicio agrega un node (renglón) en la Tabla Primaria de Ruteo.
  - **UInteger Node-ID**: ID del nodo a agregar.
  - **String Node-Alias**: Alias del nodo a agregar.
  - **Boolean status**: Estado del nodo. CONNECTED=1, DISCONNECTED=0. Por defecto los nodos están en estado 1 (CONNECTED).
  - **UInteger distance**: Distancia que existe entre el nodo "host" y el nodo a agregar.
  - **UInteger address**: Dirección del nodo a agragar.

A.10. Formato de mensajes

Los mensajes enviados entre los nodos deben tener un formato compatible con las redes CAN convencionales. Por ello el mensaje tiene un tamaño que varía desde los 45 bits hasta los 109 bits (5 bytes hasta los 13 bytes), dependiendo de la cantidad de datos en el *Data Field*.

En la Figura A.22 se observa la el formato de Frame de datos de los mensajes CANae.

SOF	Priority	Node-ID	RTR	Type of Message	Reserved bit	Data length	Data field	CRC	CRC delimiter	ACK	ACK Delimiter	EOF
1	4	7	1	1	1	0-8	0-64	1	1	1	1	1

Figura A.22: Frame de datos CANae

Los campos del mensaje son los siguientes:

- Start of Frame (SOF) - 1 bit: Este bit marca el comienzo de un frame. Suele ser 0.
- Priority - 4 bits: Este campo declara la prioridad del frame.
- Node-ID - 7 bits: El ID del Nodo origen.
- Remote transmission request (RTR) - 1 bit: Bit dominante (0) para frames de datos o eventos, y bit recesivo (1) para frames remotos. Un frame remoto es un mensaje que es enviado en forma automática por algun sensor. Los frames remotos se utilizan para enviar telemetría por parte de los sensores y componentes.

- Type of Message (TOM) - 1 bit: Bit dominante (0) si es un evento, y bit recesivo (1) para frames remotos.
- Bit reservado - 1 bit: Este bit está reservado para futuras aplicaciones. En esta versión debe ser un bit dominante (0).
- Data length - 6 bits: Cantidad de bytes del campo de datos.
- Data field - 0-64 bits (0-8 bytes): Datos.
- CRC - 15 bits: Cyclic redundancy check.
- CRC delimier - 1 bit: Debe ser recesivo (1).
- ACK - 1 bit: El transmisor debe poner este bit en recesivo (1), y el receptor puede responder este bit con un bit dominante (0).
- ACK delimiter - 1 bit: Debe ser recesivo (1).
- End-of-Frame (EOF) - 7 bits: Final del frame de datos.

El *Data Field* tiene el siguiente formato: el primer byte siempre es el ID de la función, evento y/o comando. Esto significa que es posible definir un número de $2^8 = 256$ funciones, eventos y comandos diferentes.

Los siguientes bytes (del 1 al 63) son argumentos de las funciones o datos.

Debe entenderse que esto no es mandatorio, ya que se pueden definir ID de dos o más bytes reduciendo la cantidad de datos.

A.10.1. Start of Frame

Este bit marca el comienzo de un mensaje CANae. Este es simplemente un bit dominante (0). Para que un dispositivo pueda enviar mensajes, el bus debe estar en estado IDLE.

A.10.2. Priority

Este campo indica la prioridad del mensaje. La prioridad está compuesta por 4 bits, por lo tanto acepta una cantidad de 2^4 niveles de prioridad. La prioridad máxima es 0000, mientras que la mínima es 1111.

Tabla A.2: Prioridades de mensajes

Priority Field	Prioridad	
0000	0	Alta prioridad
0001	1	...
...	...	...
1111	15	Baja prioridad

A.10.3. Node-ID

Este indica el ID del nodo que envía el mensaje. Este campo está compuesto por 7 bits, lo cual posibilita la existencia de hasta $2^7 = 128$ nodos conectados a la red.

A.10.4. Remote transmission request (RTR)

Este permite diferenciar un frame que sea remoto o un mensaje “normal”. Un frame remoto es aquel que se utiliza para enviar telemetría. Comúnmente es enviado por sensores y componentes. El bit dominante (0) indica que es un frame “normal” (contiene un mensaje o un evento), un bit recesivo (1) indica que es un frame remoto.

A.10.5. Type of Message (TOM)

Este campo es de 1 bit. Diferencia frames de mensajes con frames de eventos. Si el bit es dominante (0) significa que se trata de un frame de evento. Si el bit es recesivo (1) significa que se trata de un frame de mensaje. Debe tenerse en cuenta que si el bit RTR es recesivo (1), el bit TOM obligatoriamente debe ser recesivo (1) indicando que el frame trae consigo un mensaje con los datos de telemetría.

A.10.6. Bit reservado

Bit reservado.

A.10.7. Data length

Indica el tamaño del campo de datos (*Data Field*). Este campo esta compuesto por 6 bits.

A.10.8. Data field

En este campo se agregan los datos del mensaje. El primer byte del *Data Field* es el código de la función y/o evento. Los bytes subsiguientes pueden ser los argumentos y/o datos de la función o evento que se está enviando en el frame. Se pueden enviar solo una función a la vez, y hasta 7 datos o argumentos.

A.10.9. CRC

En este campo se almacena el CRC del frame. Para calcular el CRC se utiliza el polinomio de CAN.

$$x^{15}+x^{14}+x^{10}+x^{8}+x^{7}+x^{4}+x^{3}+1$$

El algoritmo para resolver el CRC se muestra en A.1.

Listing A.1: Función para resolver CRC

```
uint16_t crc_c(uint8_t data, uint16_t crc){
uint8_t i;

crc ^= (uint16_t)data << 7;
for (i = 0; i < 8; i++){
  crc <<= 1;
  if(crc & 0x8000){
    crc ^= 0xc599;
  }
```

```
    }
    return crc & 0x7fff;
}
```

A.10.10. CRC delimiter

Este es un bit recesivo (1) para indicar que el CRC Field finalizó.

A.10.11. ACK

El transmisor debe poner este bit en recesivo (1), y el receptor puede responder el mensaje colocando este bit en dominante(0).

A.10.12. ACK delimiter

Este bit indica el final del ACK. Este es un bit recesivo (1).

A.10.13. End-of-Frame (EOF)

Indica el final del frame de datos. Debe ser recesivo, es decir, los 7 bits en 1.

A.11. High Application Layer CANae

La denominada *High Application Layer* tiene una función más del lado de la gestión de nodos y tareas. En esta capa se desarrollan los algoritmos necesarios para realizar el ruteo y la reconfiguración de la red ante fallas, por lo tanto en la *High Application Layer* se debe implementar la Detección, Aislación y Recuperación de Fallas (FDIR). El protocolo no define ninún algoritmo de ruteo, por lo que queda para el usuario la definición de los algoritmos. Se recomienda desarrollar 2 algoritmos, el primario y el secundario. Para aumentar la tolerancia a fallas se pueden desarrollar más algoritmos secundarios, y el switch entre algoritmos, puede depender de medidas de perfomance.

En las Figuras [A.23] y [A.24] se pueden observar los diagramas de bloques y diagramas de bloques internos para el *High Application Layer* de CANae.

Esta capa está compuesta por varias entidades, las cuales tienen solo una instancia por entidad.

En la *High Application layer* se encuentra la entidad más importante, la cual es el FDIR. El desarrollo de esta entidad se encuentra fuera del alcance de esta versión (0.1 Alpha) del protocolo.

A.11.1. Network Management

Esta entidad se encarga de la gestión de la red desde el punto de vista del nodo. La entidad utiliza el *NodeTableConfiguration* del *CANae Application Layer*.

Los servicios brindados por el *Network Management* son los siguientes:

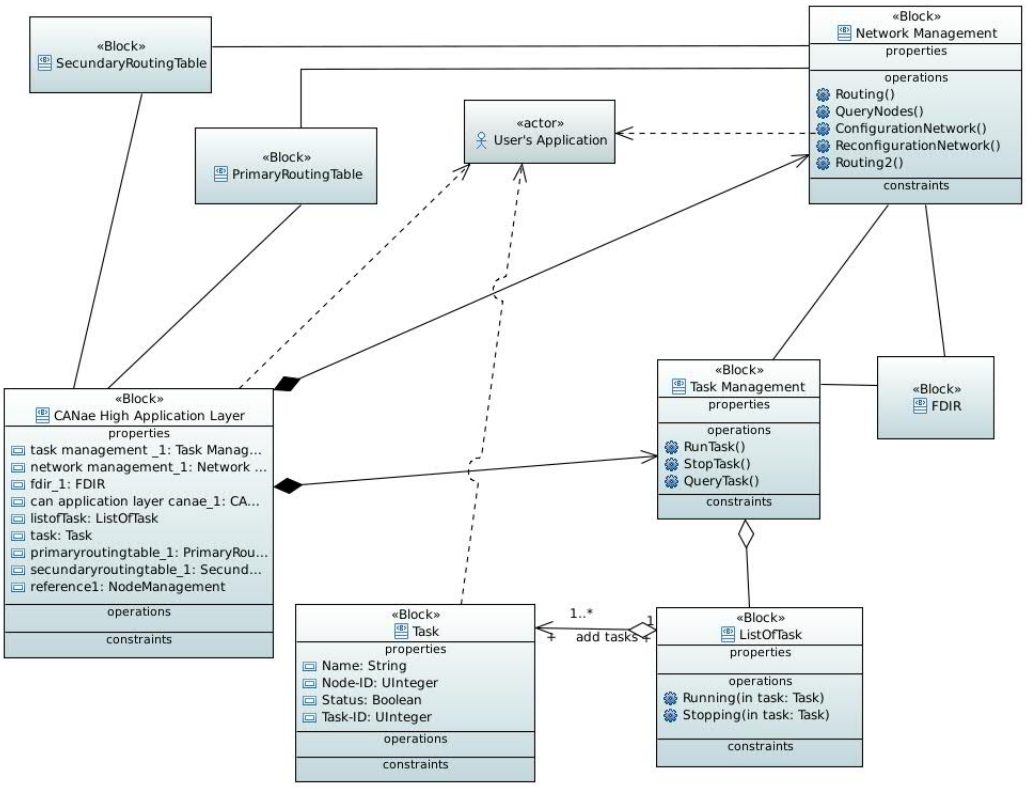

Figura A.23: Diagrama de bloques del High Application Layer de CANae

- **Routing()**: este servicio lleva a cabo el ruteo de los mensajes. Este servicio consulta la tabla *NodeTable-Configuration*.

- **QueryNodes()**: este servicio hace consulta de los nodos conectados. Este puede ser utilizado por el servicio Routing()

- **ConfigurationNetwork()**: este servicio se lleva a cabo solo una vez, durante el encendido y configuración al nodo.

- **ReconfigurationNetwork()**: este servicio lleva a cabo la reconfiguración de la red. Este servicio es activado por el FDIR una vez que detecta algún error.

- **Routing2()**: Este servicio se lleva a cabo cuando se detecta un error en la red. Lo activa el FDIR.

El comportamiento del *Network Management* es sencillo, cuando se enciende por primera vez el nodo, de manera automática, la aplicación de usuario debe llamar el servicio *ConfigurationNetwork()* al *Network Management* esta entidad se comunica con el *Node Management* para que se genere la "Tabla Primaria de Ruteo". Una vez que esta tabla se encuentre generada (tal como se hace referencia en A.9.1), la aplicación de usuario puede enviar señales de *QueryNodes()* y *Routing()*. Esto se puede ver visualmente en la Figura A.25.

En caso de errores, esto debe ser captado por el FDIR del sistema. El comportamiento del mismo se muestra en la Figura A.26

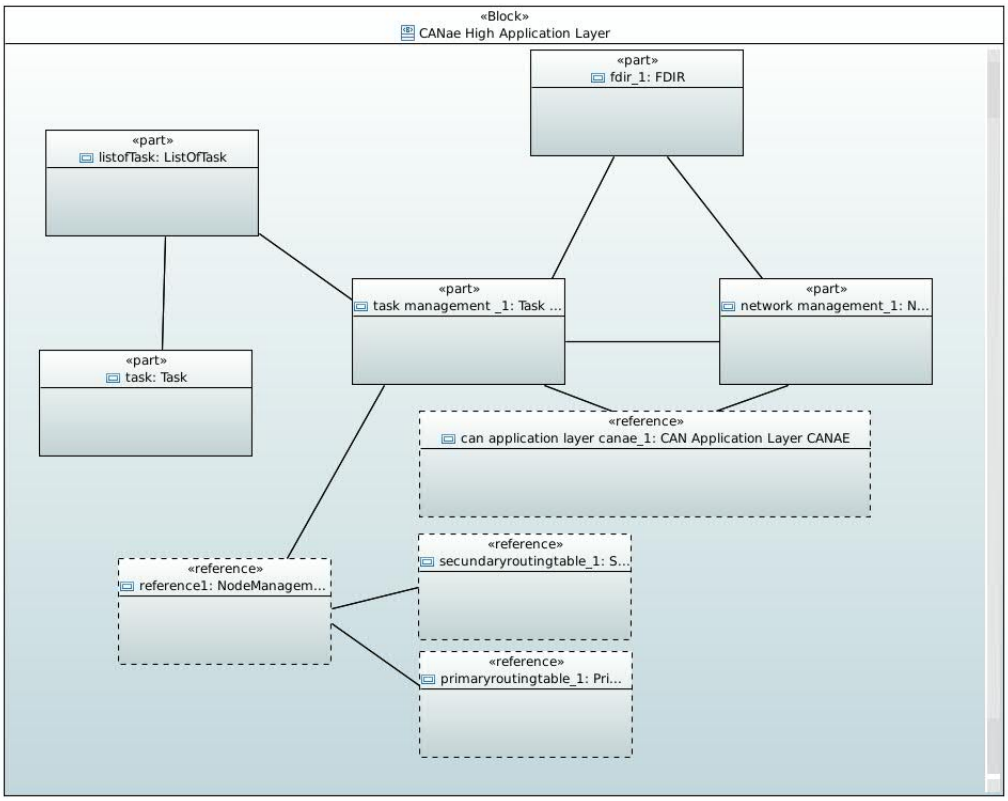

Figura A.24: Diagrama de bloques internos del High Application Layer de CANae

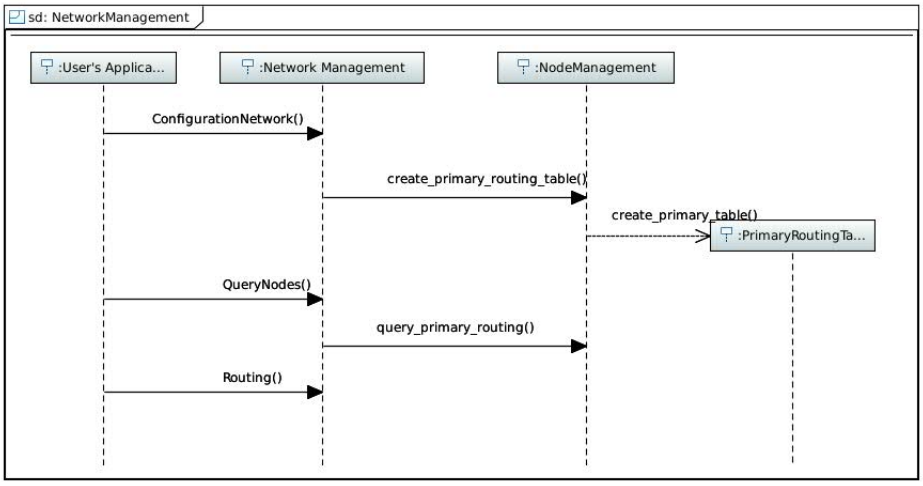

Figura A.25: Diagrama de interacción del comportamiento del *Network Management*

A.11.2. Task Management

Esta entidad se encarga de la gestión de las tareas. Debe aclararse que esta es una abstracción, y no depende del Sistema Operativo que se utilice para la codificación del sistema. El objetivo de esta entidad es mantener la distribución de las tareas entre los nodos de la red. Esto permitiría que cuando deja de funcionar (detectado por el FDIR) uno de los nodos, el *Task Management* gestiona las tareas que se desarrollan en los nodos. Para ellos se hace uso de la "Lista de Tareas" (*ListOfTask*).

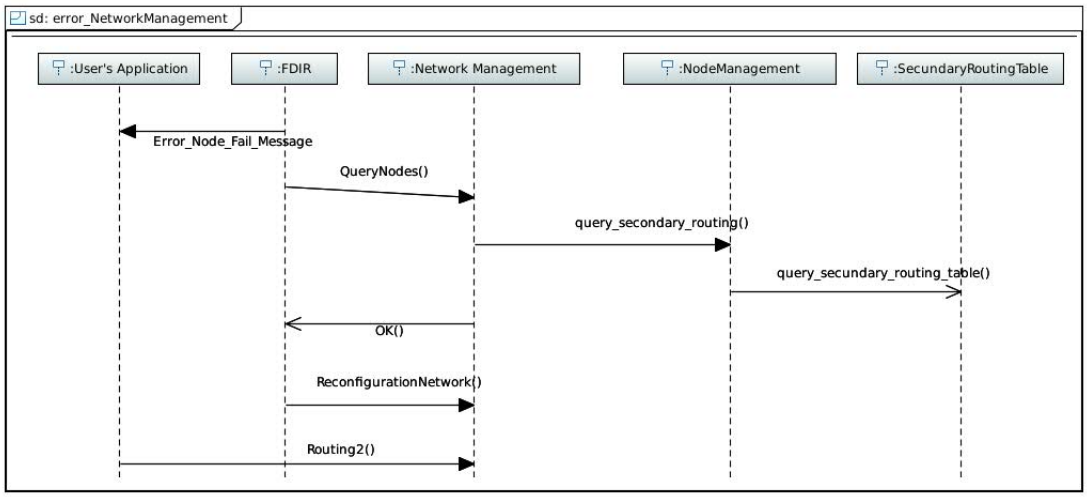

Figura A.26: Diagrama de interacción del comportamiento del *Network Management en caso de errores*

Los servicios que ofrece el *Task Management* son los siguientes:

- **RunTask()**: este servicio activa las tareas para ejecutarse en el nodo.
- **StopTask()**: este servicio detiene las tareas que se ejecutan en el nodo.
- **QueryTask()**; este servicio hace consulta de las tareas que existene la "lista de Tareas".

El comportamiento de esta entidad es sencilla. La aplicación de usuario le envía una señal de *QueryTasks()* con el motivo de conocer todas las tareas que se están ejecutando en la red. El *Task Management* lleva a cabo un *query_tasks_in_network()* al *Network Management* el cual le provee la información. La primera comprobación que debe realizar la aplicación de usuario es que la tarea no existe en otro nodo ejecutandose. Luego de esta comprobación ya tiene permitido llevar a cabo el *RunTask()*. Por último el *Task Management* manda el mensaje *Running(Task task)* al *ListOfTasks*(ver A.11.3). El comportamiento de la entidad se la puede observar en la Figura A.27.

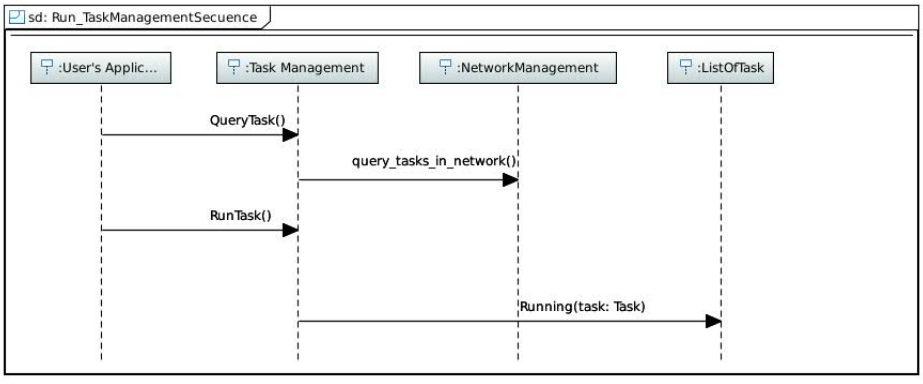

Figura A.27: Diagrama de secuencia de la ejecución de tareas del Task Management

Por otro lado cuando la aplicación de usuario necesita para una tarea, o debido a que el FDIR detecto una falla y necesita detener la ejecución de una tarea, el proceso que debe llevar el *Task Management* es el siguiente, en primer hace un *QueryTasks()* para conocer las tareas que se están ejecutando y sus estados. Luego manda un señal de *StopTask()* al *Task Management*. Esta, envía mensajes de *query_tasks_in_network()*, y luego el *Stopping(Task task)* para detener la ejecución del proceso. En la Figura A.28

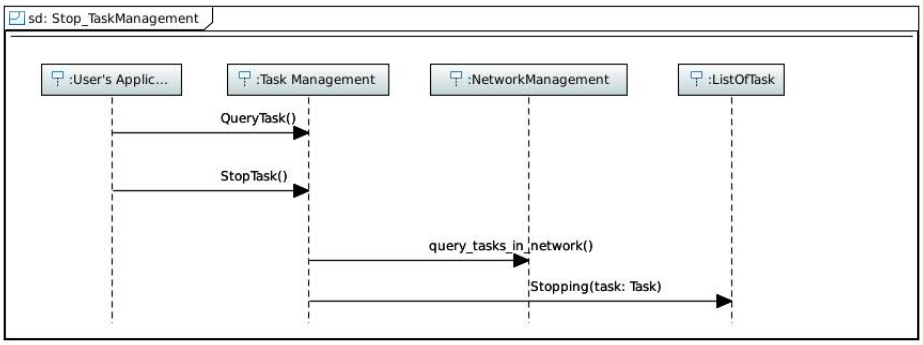

Figura A.28: Diagrama de secuencia para detener procesos por parte del Task Management

A.11.3. List of Tasks

Este es un objeto que mantiene una lista de las tareas que se están definidas en el sistema. Estas tareas pueden o no estar ejecutandose. Esta lista tiene un instancia a todas las tareas.

Los servicios que se ejecutan en esta entidad son los siguientes:

- **Running(Task task)**: este servicio ejecuta una tarea.
  - **Task task**: tarea a ejecutar.
- **Stopping(Task task)**: este servicio detiene una tarea.
  - **Task task**: tarea a ejecutar.

A.11.4. Task

Este es un objecto que representa la abstracción de las tareas. Las propiedades de esta entidad son las siguientes:

- **String Name**: es el nombre de la tarea.
- **UInteger Task-ID**: es el ID de la tarea.
- **UIntenger Node-ID**: es el ID del nodo donde se está ejecutando la tarea.
- **Boolean Status**: es el status de la tarea {RUNNING, STOPED}

A.12. Arquitectura completa del CANae

CANae nace en el principio de la ingeniería en sistemas basada en modelo utilizando el lenguaje de modelado SysML para su diseño y desarrollo. Debido a la complejidad de los sistemas espaciales, a la necesidad de lograr un entendimiento completo por parte de los desarrolladores, y para lograr un correcta interpretación del protocolo se decidió basarse en este principio. En esta sección se pretende mostrar solo con motivo informativo la arquitectura estáttica completa del protocolo, para demostrar la complejidad del mismo, pero a la vez lo sencillo que resulta su interpretación, con conocimientos básicos de SysML.

En esta se puede observar como interactúan las diferentes entidades, objetos y eventos que se desarrollaron a lo largo de este documento.

En la Figura A.29 se observa la arquitectura de la capa de aplicación de CANae denominada: **CAN Application Layer CANae**. La cual como se mencionó con anterioridad se encuentra basada en la capa de aplicación de diferentes protocolos que existen para las redes CAN.

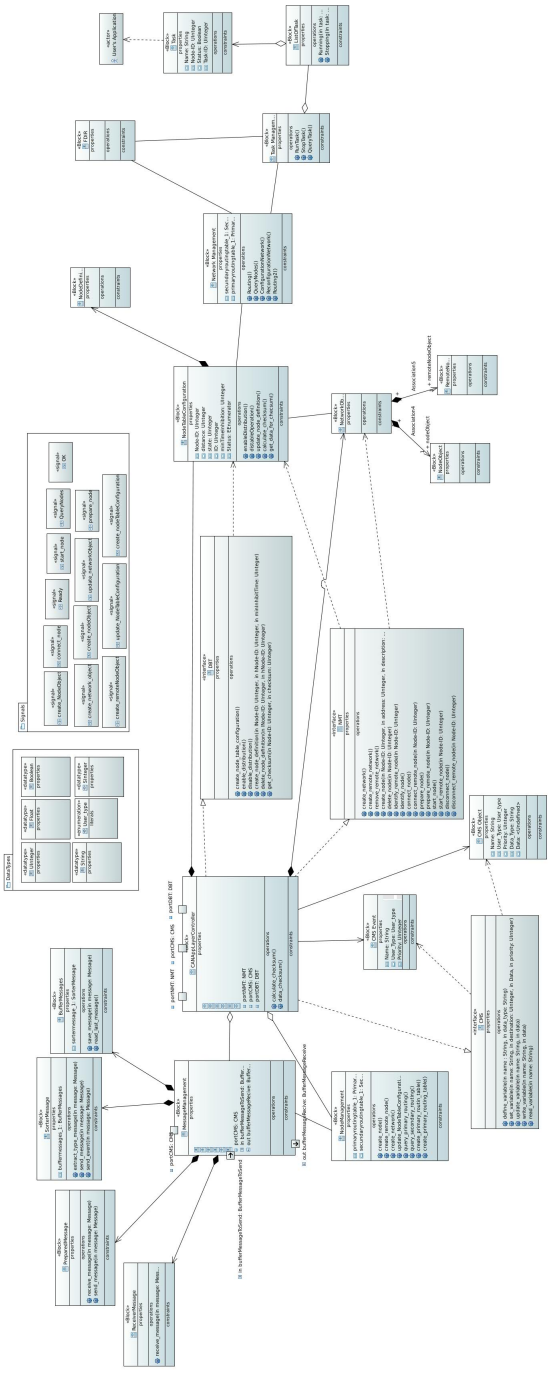

Figura A.29: Arquitectura completa del CANae