ESCOLA POLITÈCNICA SUPERIOR

UNIVERSITAT DE LES ILLES BALEARS

PROYECTE FINAL DE CARRERA

Estudi:

Enginyeria en Informàtica

Títol:

Modelado mediante Stochastic Activity Networks (SANs) de la Fiabilidad de un Sistema Distribuido en el que los nodos se comunican a través de una red con Topología de Bus Replicado basada en el protocolo Controller Area Newtwork (CAN)

Alumne:

Francisco Manuel Pozo Pérez

Director:

Manuel Barranco
Julián Proenza

Data:
3/3/2014

Índice general

Parte I

Introducción

1. Contexto y motivación

Un bus de campo (*fieldbus*) es un protocolo de comunicación digital que se utiliza para interconectar los diferentes dispositivos ubicados en los niveles más bajos de la arquitectura de un sistema de control distribuido, p.e. sensores, actuadores, PLCs (*Programmable Logic Controllers*), microcontroladores, etc.

Los buses de campo se usan ampliamente en sistemas de control distribuido debido principalmente a su bajo coste, así como a la flexibilidad y facilidad de su puesta en marcha y mantenimiento. Además, debido a su robustez (principalmente frente a interferencias electromagnéticas), los buses de campo también se utilizan en multitud de aplicaciones que requieren una elevada *garantía de funcionamiento* (*dependability*).

Un bus de campo normalmente comprende el *Nivel Físico*, el *Nivel de Enlace de Datos* y, en ocasiones, el *Nivel de Aplicación* del modelo de referencia OSI. Desde el punto de vista físico, un bus de campo consta de un conjunto de conductores (cables), conectores y otros elementos discretos (como resistencias) que constituyen un medio de transmisión común que los diferentes dispositivos comparten para comunicarse entre ellos. Esta es la razón por la cuál se dice que la topología subyacente a esta tecnología es un *bus simple*.

Sin embargo, una topología de bus simple presenta inconvenientes desde el punto de vista de la dependability. Algunos de estos inconvenientes se deben al hecho de que esta topología dificulta la contención de errores e, intrínsecamente, no provee mecanismos de tolerancia a fallos, p.e. si se produce un cortocircuito en una sección del bus, éste puede quedar totalmente inutilizado.

La dependability en general, y la fiabilidad (*reliability*) en particular, que se pueden conseguir mediante diferentes topologías de bus de campo es una cuestión que ha despertado un gran interés en los últimos años. Concretamente, en los últimos años los buses de campo destinados a aplicaciones de control críticas, p.e. en el sector de la automoción, la aviación, las aplicaciones aeroespaciales y la automatización industrial, han evolucionado para ofrecer la posibilidad de utilizar topologías alternativas; principalmente buses replicados y estrellas simples y replicadas. Estas alternativas mejoran la contención de errores y, cuando usan redundancia, también proveen tolerancia a fallos.

Desgraciadamente, a pesar de los esfuerzos que se están llevando a cabo para mejorar la dependability mediante estas topologías, la literatura carece de análisis apropiados que cuantifiquen el grado de fiabilidad que realmente se consigue con ellas. Este hecho ha motivado a algunos miembros del grupo de *Sistemas, Robótica y Visión* (SRV) de la *Universitat de les Illes Balears* (UIB) a modelar y cuantificar la fiabilidad que un sistema de control distribuido basado en un bus de campo puede alcanzar mediante diferentes topologías. Concretamente, sus estudios se centran en el protocolo *Controller Area Network* [ISO03a] (CAN), ya que éste es uno de los buses de campo más maduros, baratos y usados en la actualidad. Además, por otro lado, se espera que en los próximos años el uso de CAN siga creciendo aún más en todos los ámbitos incluyendo las aplicaciones críticas [MCACGC+11]

El primer análisis realizado por el SRV en este sentido tuvo como objetivo cuantificar en qué medida se puede mejorar la fiabilidad de sistemas basados en CAN a través de una mejor contención de errores. Para ello en [BPA11] se comparó cuantitativamente la fiabilidad de sistemas CAN dependiendo de si utilizan una topología de bus simple, o una topología de estrella simple llamada CANcentrate. Ésta última es la estrella simple que se ha propuesto para CAN que ofrece la mayor capacidad de contención de errores. Concretamente CANcentrate consta de un hub que permite detectar errores y diagnosticar su procedencia con gran precisión. Cuando este hub diagnostica que uno de sus puertos es el origen de los errores, lo aísla para evitar que éstos se propaguen al resto del sistema.

Los resultados obtenidos en [BPA11] son prometedores porque muestran como la mayor capacidad de contención de errores de una estrella simple permite incrementar la fiabilidad en comparación con un bus simple.

Una vez cuantificado el grado de mejora de la fiabilidad que se puede conseguir aumentando la contención de errores, el SRV se propuso cuantificar cuál es el incremento de fiabilidad que se puede conseguir si, además, se añade la capacidad de tolerar fallos que afecten al medio de transmisión en sí o a las interfaces con éste, p.e. fallos que afecten a los cables, conectores, transceptores, controladores de comunicaciones, etc. Para ello, el siguiente paso que el SRV estableció en su hoja de ruta consiste en cuantificar la fiabilidad que se puede conseguir mediante el uso de topologías redundantes; concretamente mediante una estrella y un bus replicados.

Así pues, en [BPA] el SRV modeló y cuantificó la fiabilidad que puede alcanzar un sistema de control basado en CAN al utilizar la estrella replicada ReCANcentrate. Esta topología básicamente consiste en una red CANcentrate replicada, y constituye la única estrella replicada que se ha propuesto para CAN hasta la fecha. Los resultados en [BPA] corroboran que una estrella replicada puede mejorar la fiabilidad cuando se la compara con ambos, el bus y la estrella simples.

Cabe destacar que el SRV decidió modelar en primer término la estrella replicada y no el bus replicado, porque la primera provee mejor contención de errores que el segundo y, por tanto, intuitivamente provee mayor fiabilidad que éste. Sin embargo, una estrella replicada incluye más componentes que un bus replicado y, por consiguiente, aún no está claro cuál de los dos es más fiable. Es más, comparar ambas topologías es fundamental porque los buses replicados normalmente tienen un menor coste económico y, por tanto, pueden ser más adecuados para ciertas aplicaciones incluso si se demuestra que son menos fiables.

Es en este punto donde se **propuso la realización del presente proyecto**, cuyo **objetivo general** es el de modelar apropiadamente la fiabilidad que pueden alcanzar los sistemas basados en CAN cuando utilizan una topología de bus replicado.

Por una parte, el presente proyecto contribuirá a rellenar el vacío existente en la literatura, en la que el único trabajo que cuantifica la fiabilidad de un

bus replicado en el contexto de los buses de campo, [AvDF11], adolece de ciertas limitaciones que se observan en muchos modelos de fiabilidad de redes de comunicaciones [BPA]. Estas limitaciones consisten básicamente en no modelar detalles que pueden tener una influencia muy importante en la fiabilidad, p.e. la probabilidad con la que los mecanismos de tolerancia a fallos realizan su cometido con éxito.

Por otro lado, los resultados de este proyecto permitirán comparar cuantitativamente la fiabilidad que se puede alcanzar en CAN mediante el uso de estrellas y buses replicados y, por tanto, contribuirá a arrojar luz sobre cuál de las dos topologías es más adecuada para mejorar la fiabilidad de los sistemas de control distribuidos basados en buses de campo.

2. Objetivos

Como se ha dicho más arriba, el objetivo general de este proyecto es el de construir un modelo que permita cuantificar la fiabilidad de un sistema de control distribuido basado en CAN con una topología de bus replicado.

Más concretamente, se debe modelar la fiabilidad de un sistema que utilice el bus replicado propuesto en el contexto de CANELy (*CAN Enhanced Layer*) [**?**]. Como se explicará más adelante, CANEly consiste en una serie de servicios encaminados a mejorar la dependability y las garantías de comunicación en tiempo real de CAN.

La razón por la que se ha escogido este bus estriba en el hecho de que es el bus replicado para CAN que incluye la menor cantidad de hardware y que, al mismo tiempo, permite gestionar el tráfico replicado de la forma más sencilla [RVA99]. En este sentido, como se verá más adelante, tanto CANcentrate como ReCANcentrate están en parte inspirados en los mecanismos de este bus replicado.

Por último, a continuación se listan los requisitos que debe cumplir el modelo desarrollado en este proyecto.

- El modelo debe representar el sistema con el mismo grado de detalle que los modelos del bus CAN, CANcentrate y ReCANcentrate.
- El modelo debe incluir parámetros que permitan caracterizar cuantitativamente aspectos que puedan tener una influencia relevante sobre la fiabilidad del sistema. De esta forma, el modelo debe permitir realizar análisis de sensibilidad con respecto al valor de dichos parámetros.
- El modelo debe incluir parámetros para caracterizar al menos los mismos aspectos que se parametrizaron en los modelos del bus CAN, CANcentrate y ReCANcentrate, y que son comunes en el bus replicado y las estrellas. En cualquier caso, los parámetros deben permitir analizar la influencia que sobre la fiabilidad tienen aspectos exclusivos del bus replicado.

- El modelo debe incluir parámetros que permitan, al menos, caracterizar los siguientes aspectos.
  - Número de buses. Si bien el objetivo de este proyecto es modelar exclusivamente la fiabilidad para el caso en que se tienen dos replicas del bus.
  - Número de nodos.
  - Ratio de fallo de cada uno de los componentes hardware que constituyen el sistema.
  - Proporciones con las que cada componente hardware que forma el sistema falla exhibiendo diferentes modos de fallo.
  - Cobertura (probabilidad de éxito) de los diferentes mecanismos de tolerancia a fallos. Notar que los mecanismos de contención de errores también se consideran como mecanismos de tolerancia a fallos.
- El modelo debe construirse mediante el formalismo conocido como *Stochastic Activity Network* (SAN) [AM02], el cuál es un tipo de *Petri Net*.
- La estrategia de modelado debe basarse en la estrategia que se empleó para construir los modelos de CAN, CANcentrate y ReCANcentrate. De esta forma, el modelo resultante debe satisfacer los siguientes requisitos.
  - Debe permitir comparar los resultados de fiabilidad obtenidos para un bus replicado con los resultados de fiabilidad que se pueden obtener con los modelos ya existentes del bus CAN, CANcentrate y ReCANcentrate.
  - Debe permitir identificar las limitaciones de esta estrategia de modelado para abordar arquitecturas redundantes en las que, como el bus replicado y a diferencia de las estrellas, los mecanismos de tolerancia a fallos tienden a estar distribuidos.
- El modelo debe poder resolverse de forma analítica.
- El modelo debe poder resolverse en una cantidad de tiempo razonable, en la medida de lo posible, y ateniéndose a las limitaciones ya conocidas de las SAN para modelar sistemas complejos [AM02].

3. Hitos del proyecto

El presente proyecto se ha dividido en una serie de etapas o hitos. A continuación se resume cuál ha sido el trabajo llevado a cabo en cada uno de ellos.

Documentación

Antes de iniciar el proyecto y durante la realización del mismo, ha sido necesario estudiar y asimilar una cantidad significativa de literatura sobre diferentes aspectos. Los conocimientos más relevantes que se han estudiado en esta fase son los siguientes.

- Conceptos generales sobre dependability.
- Las características del protocolo CAN que son relevantes para la realización del proyecto.
- Las arquitecturas de CANcentrate y ReCANcentrate.
- Arquitecturas de buses replicados en general.
- La arquitectura de bus replicado para CAN de CANELy.
- Conceptos básicos sobre técnicas de modelado de la dependability.
- Formalismo SAN, patrones de modelado con el mismo.
- Los modelos SANs ya existentes del bus CAN, CANcentrate y ReCANcentrate.
- La herramienta de modelado Möbius [StBoT04].
- El lenguaje de scripting Perl [SdFP11].
- Aspectos básicos sobre la herramienta de control de versiones GIT [Cha09].

Diseño del modelo de fiabilidad

Durante la fase de diseño se ha ido construyendo progresivamente el modelo en *lapiz y papel*. Como se explicará más adelante, el modelo se ha construido de forma modular y jerárquica. Por tanto, en esta fase se han ido proponiendo diferentes versiones tanto del modelo general, como de cada una de sus partes, con el objetivo de ir refinándolo. En este proceso de refinamiento se ha tenido especial cuidado en los siguientes aspectos: (1) incluir tantos detalles del sistema como sea posible, (2) realizar las abstracciones pertinentes sin perder generalidad, y (3) modelar de forma precisa y exhaustiva las capacidades de contención de errores y tolerancia a fallos del sistema.

Aprendizaje de las herramienta de desarrollo

El trabajo realizado en la fase de aprendizaje de las herramientas de desarrollo ha consistido en familiarizarse con el software Möbius, el lenguaje de scripting Perl y el software de control de versiones GIT. Por una parte en esta fase se ha utilizado Möbius para realizar y resolver modelos sencillos de dependability y, de esta forma, conocer los principales aspectos técnicos del software. Por otra parte, se ha utilizado Möbius para realizar algunas experiencias con los modelos ya existentes de CAN, CANcentrate y ReCANcentrate; de esta manera se ha conseguido profundizar más en los detalles tanto de la herramienta como de los modelos. Finalmente, Perl se ha utilizado como lenguaje de scripting para procesar los resultados obtenidos con Möbius y visualizarlos en forma de gráficas.

Implementación del modelo de fiabilidad

La fase de implementación consiste básicamente en implementar mediante Möbius el modelo que se construye en la fase de diseñado.

Cabe destacar, sin embargo, que durante la fase de implementación, se han encontrado algunos problemas relacionados con la explosión de estados. Por tanto, una parte importante del tiempo invertido en esta fase se ha centrado en la necesidad de extender y adaptar la estrategia de modelado inicial para solventar estos problemas.

Es importante notar, además, que realmente ha existido un periodo del proyecto en cuál las fases de diseño e implementación se han ido alternando a fin de obtener el modelo definitivo.

Verificación del modelo de fiabilidad

La fase de verificación se ha dividido en dos tipos de actividades. En primer lugar, se han ido realizando pruebas con cada uno de los submodelos que componen el modelo total. Estas pruebas han servido para depurar el modelo, así como para identificar los problemas relacionados con la explosión de estados que se han comentados antes.

En segundo lugar, se han realizado varios análisis de fiabilidad con el modelo total, tanto para comprobar que el modelo funcionaba correctamente, como para realizar comparaciones cuantitativas con los resultados de fiabilidad obtenidos con ReCANcentrate.

4. Organización de la memoria

El resto de esta memoria se divide en diferentes partes. La Parte II está formada por una serie de capítulos en los que se exponen los fundamentos teóricos y prácticos necesarios para entender el trabajo realizado en este proyecto. Concretamente, la Parte II aborda aspectos teóricos básicos sobre dependability, el protocolo CAN, las estrellas CANcentrate y ReCANcentrate, así como el bus replicado de CANEly. Además, los dos últimos capítulos de esta parte respectivamente resumen los tipos de técnicas de modelado que se utilizan para analizar la dependability, poniendo énfasis en las SAN, y describen el software de modelado Möbius.

Posteriormente en la Parte III se explican la estrategia de modelado de partida, así como la forma en que en este proyecto se ha extendido y modificado esta estrategia para modelar el bus replicado. La última sección de esta parte presenta el modelo del bus replicado realizado en el proyecto. Con el objetivo de no extender esta memoria en exceso y ayudar al lector a entender los aspectos claves del modelo, dicha sección no describe en detalle todo el modelo, sino una traza representativa del mismo. Es decir, esta sección muestra cómo se ha modelado mediante SANs la ocurrencia de un determinado tipo de fallo, las acciones que el sistema lleva a cabo para tratarlo y tolerarlo, y el impacto del fallo en el servicio que presta el sistema.

En la Parte IV se muestran una serie de resultados que muestran la utilidad del modelo realizado en este proyecto, tanto para cuantificar la fiabilidad de un sistema basado en CAN con topología de bus replicado, como para comparar

dicha fiabilidad con la que se obtiene al utilizar otras topologías, p.e. ReCAN-centrate.

Finalmente, en la Parte V se muestran las conclusiones del proyecto y se mencionan algunas posibles extensiones al trabajo presentado aquí.

Parte II

Conceptos básicos

5. Garantía de funcionamiento (*dependability*)

5.1. Introducción

El objetivo de la presente sección es introducir los conceptos básicos relacionados con la Garantía de funcionamiento. En este sentido, y en primer lugar, es necesario aclarar que el término Garantía de funcionamiento es una traducción del término *Dependability* en inglés. Existen varias formas de definir la *dependability*, pero una buena definición es la siguiente [Car82]: *Dependability* es la certeza sobre el correcto funcionamiento de un sistema de cómputo de forma que se puede justificar la confianza depositada en el servicio que éste presta. De ahora en adelante utilizaremos el término *dependability* para ser más escuetos.

La dependability comprende diferentes atributos. Los más importantes son los conocidos como fiabilidad (*reliability*), disponibilidad (*availability*), seguridad (*security*), facilidad de mantenimiento (*maintainability*), facilidad de testeo (*testebility*) y rendimiento con garantías de funcionamiento (*performability*).

Tal y como se ha explicado en la sección anterior, este proyecto se centra en la fiabilidad, la cuál se puede definir de la siguiente forma [Sho02]:

Fiabilidad es la probabilidad con la que un sistema proporciona de forma ininterrumpida a lo largo de un intervalo de tiempo determinado el servicio para el cuál fue concebido.

En las siguientes secciones se explican cuáles son los daños que afectan a la dependability (y por tanto a la fiabilidad) de un sistema; cómo se clasifican jerárquicamente esos daños; cuáles son los mecanismos básicos para conseguir aumentar la dependability; cuáles son los aspectos esenciales que se han de tener en cuenta a la hora de diseñar un sistema que alcance una dependability elevada; y cuáles son las métricas clásicas que se utilizan para caracterizar la fiabilidad.

5.2. Daños que afectan a la *dependability*

Existen daños a la hora de alcanzar una *dependability* deseada. Estos daños son conocidos como: *fallo*, *error* y *avería*, y existe una relación de causa y efecto entre ellos. Un *fallo* es un defecto en el comportamiento del sistema o en la forma en que el sistema esta diseñado o construido. Un fallo puede generar uno o más *errores*, que son resultados incorrectos generados por el sistema. Por último, un error puede generar una *avería*, que implica que el comportamiento del sistema se desvía del servicio para el cual fue diseñado. Un sistema que presenta una avería se dice que está *averiado*.

Si se tiene un sistema compuesto por diferentes subsistemas o componentes, una avería en un subsistema se puede considerar como un fallo desde el punto de vista del sistema global. Es decir, un subsistema averiado puede considerarse como un fallo en el sistema del cuál forma parte, de forma que los errores que este fallo genera pueden llevar a la avería de dicho sistema. Por esta razón no siempre es fácil diferenciar entre fallo y avería; hablar de un fallo o de una avería

dependerá de si el elemento que falla es considerado o no como un subsistema de otro mayor.

Un fallo puede o no generar uno o más errores en un momento determinado. Cuando un fallo genera algún error se dice que está *activo*; cuando no genera ningún error se dice que está *latente* [Lap92] [Lap01]. A lo largo del tiempo un fallo puede cambiar desde el estado *activo* al estado *latente* y viceversa. Según la forma en que un fallo transita de un estado al otro, se puede clasificar como *permanente*, *intermitente* o *transitorio*. Un *fallo permanente* es aquel que se encuentra siempre activo y que, por tanto, genera errores continuamente. Un *fallo intermitente* y un *fallo transitorio* es aquel que oscila entre el estado activo y el estado latente a lo largo del tiempo. La diferencia entre ellos es que el primero cambia de un estado a otro de una forma muy rápida en comparación al segundo; además los fallos transitorios suelen producirse como consecuencia de alguna influencia temporal externa y, en este sentido, suelen activarse de una forma puramente aperiódica.

5.3. Clasificación jerárquica de las averías

Un aspecto esencial que se debe analizar a la hora de incrementar la dependability de un sistema es la forma en que los diferentes fallos se manifiestan, tanto al nivel del sistema en sí, como al nivel de sus subsistemas. En este sentido, se define el concepto de *modo de fallo* (*failure mode*) como la manera en que una avería se manifiesta. Más concretamente y a fin de poder caracterizar exhaustiva y ordenadamente las formas en que un sistema y sus subsistemas fallan, los modos de fallo se clasifican jerárquicamente de la siguiente manera [Pol96]:

- **Averías arbitrarias:** Un sistema que presenta una avería arbitraria tiene un comportamiento incorrecto sin restricciones ni en el dominio de los valores ni en el dominio del tiempo. Este tipo de averías también son denominadas *incontroladas* o *maliciosas*. Un modo de fallo malicioso lo es en el sentido de que tiene un comportamiento inconsistente, de forma que el sistema que lo sufre provee con diferentes resultados a diferentes usuarios u otros sistemas, haciendo que éstos tengan una percepción diferente del servicio administrado por el sistema. También se considera malicioso cuando el sistema falsifica mensajes o resultados que son de otros sistemas.
- **Averías de autenticación detectable:** Un sistema que presenta una avería de autenticación detectable tiene un comportamiento similar a un sistema que sufre una avería incontrolada. Pero con una excepción, el sistema no puede falsificar mensajes o resultados de otros sistemas. Es decir, este modo de fallo equivale al de una avería arbitraria pero en la que el sistema no puede hacerse pasar por otro.
- **Averías de computación incorrecta:** Un sistema que presenta una avería de computación incorrecta provoca que el sistema tenga un comportamiento incorrecto en el dominio de los valores o en dominio del tiempo.
- **Averías de rendimiento:** Un sistema que presenta una avería de rendimiento envía resultados correctos desde el punto de vista del valor de

dichos resultados, pero incorrectos desde el punto de vista temporal. Por ejemplo, un sistema que sufre una avería de rendimiento puede producir resultados demasiado pronto o demasiado tarde.

- **Averías de omisión:** Un sistema que presenta una avería de omisión no responde a una petición de un resultado, o dicho de otra forma, el tiempo de respuesta a la petición es infinito. Hay que tener en cuenta que esto no quiere decir que el sistema pare de realizar su servicio, sino que puede o no responder a las peticiones que recibe.
- **Averías de tipo *crash*:** Un sistema que presenta una avería de tipo crash deja de proveer su servicio permanentemente. Se puede decir que un sistema que presenta un crash nunca responde.
- **Averías de parada:** Es un caso especial del tipo de avería anterior. Cuándo se le realiza una petición a un sistema que presenta una avería de parada, éste siempre responde con el valor que produjo la última vez que funcionó correctamente.

Esta categorización jerárquica de los modos de fallo es muy útil para diseñar un sistema con garantías de funcionamiento, ya que ayuda a especificar de una forma precisa los modos de fallo que el sistema y sus subsistemas pueden exhibir. Por una parte, especificar la forma de que un sistema puede fallar es vital para poder tomar las medidas adecuadas cuando el sistema falla, por ejemplo, parar un vehículo cuando el sistema de control que lo guía se avería. Por otro lado, detallar las formas en que cada subsistema puede fallar es fundamental para diseñar los mecanismos adecuados que el sistema debe incluir para tratar dichos fallos, por ejemplo, para contener los errores generados por un subsistema concreto y evitar que estos se propaguen.

Al conjunto de modos de fallo que un sistema y sus subsistemas pueden exhibir se le denomina generalmente como la *semántica de averías* de dicho sistema. Una definición más formal del concepto de *semántica de averías* es el siguiente [Pol96]: Un sistema exhibe una determinada semántica de averías si la probabilidad de que se den modos de fallo que no están cubiertos por dicha semántica es suficientemente baja.

En la definición anterior aparece el concepto de *cobertura* (*coverage* en inglés). En general, el concepto de *cobertura* en un sistema con garantías de funcionamiento se refiere a la probabilidad de que una propiedad o capacidad concreta de dicho sistema sea cierta. En este sentido existen muchos tipos de coberturas. Por ejemplo, la cobertura de un mecanismo de detección de errores hace referencia a la probabilidad con la que, realmente, dicho mecanismo detecta con éxito los errores que debería ser capaz de detectar. Otro ejemplo es la cobertura citada en el párrafo anterior. Dicha cobertura se conoce como *cobertura de la hipótesis de fallos* (*fault-assumption coverage* en inglés); y se define formalmente como la probabilidad de que un sistema falle realmente de alguna de las formas en que se supone que debería fallar, condicionada por el hecho de que le sistema se averíe [Pow92].

De hecho, uno de los primeros pasos para diseñar un sistema con garantías de funcionamiento consiste en recopilar el conjunto de los modos de fallo que

sus componentes pueden exhibir. A este conjunto se le conoce con el nombre de *modelo de fallos* (*fault model* en inglés). Si se produce una avería que está fuera de este modelo, se dice que dicha avería está *fuera del modelo de fallos*.

Finalmente, es importante notar que se hace un gran esfuerzo para restringir la semántica de averías de los subsistemas que componen un sistema global. Esto es así porque restringir la semántica de averías de los subsistemas permite simplificar los mecanismos que el sistema debe incluir para evitar que el fallo de un subsistema perjudique al resto del sistema. Por ejemplo, es más fácil detectar y contener los errores generados por un subsistema que falla exhibiendo una avería de omisión, que los errores generados por un subsistema que exhibe una avería de computación incorrecta.

5.4. Mecanismos para aumentar la *dependability*

Existen diferentes formas mediante las cuales se puede aumentar la *dependability*. Éstas se conocen por el nombre de *prevención de fallos*, *tolerancia a fallos*, *previsión de fallos* y *eliminación de fallos* [Pro07]. Este proyecto se centra en la *tolerancia a fallos*, que nos permite mantener el servicio aún con la presencia de fallos en el sistema.

La tolerancia a fallos se divide en dos fases: *procesamiento de errores* y *tratamiento de fallos* [AL81]. El objetivo del *procesamiento de errores* es eliminar errores del sistema; mientras que el *tratamiento de fallos* tiene como objetivo prevenir que los fallos vuelvan a generar errores. Existen varias formas de procesar errores, pero las más importantes son la *detección de errores*, que descubre errores en el sistema; y la *recuperación de errores*, que retorna el sistema a un punto en el que no presenta errores. En cuanto al tratamiento de fallos, éste se divide en *diagnostico de fallos*, para identificar cuál es el fallo que está provocando errores en el sistema; y *desactivación de fallos* para impedir que el fallo cause nuevos errores.

El procesamiento de errores y el tratamiento de fallos pueden conseguirse siguiendo dos técnicas diferentes: *tolerancia a fallos a medida* y *tolerancia a fallos sistemática* [Pol96]. La primera consiste en usar el conocimiento que se tiene sobre el sistema específico para detectar fallos y tolerarlos. En cambio, la tolerancia a fallos sistemática utiliza *redundancia* para, en primer lugar, detectar fallos cuando el resultado de diferentes *réplicas* difiere y, en segundo lugar, para continuar proveyendo servicio utilizando únicamente las réplicas que no han fallado.

El sistema modelado en este proyecto utiliza redundancia para tolerar fallos. Por ejemplo, el uso de dos buses permite tolerar el fallo de uno de ellos, ya que los nodos pueden seguir comunicándose a través del bus que no ha fallado. De hecho, la redundancia es la alternativa más habitual para tolerar fallos, debido a su efectividad y la facilidad para incluir replicas.

Existen dos tipos diferentes de redundancia, a saber, *redundancia espacial*, la cual consiste en incluir hardware, software, o incluso código adicional; y *redundancia temporal*, que consiste en usar tiempo de computación extra para

realizar funciones específicas de detección de errores y tolerancia a fallos.

Uno de los aspectos más importantes y delicados a la hora de decidir como introducir redundancia es identificar si es necesario lidiar con *fallos relacionados* o *fallos independientes*. Los fallos relacionados son aquellos que tienen un origen común, mientras que los independientes no. Por ejemplo, si dos subsistemas comparten un mismo recurso, p.e. utilizan la misma fuente de reloj, y dicho recurso se avería, entonces los dos subsistemas sufrirán de un fallo relacionado.

Como se puede deducir, los fallos relacionados pueden provocar que todas o varias réplicas fallen, haciendo que la redundancia sea inútil. Por ejemplo, si se usan varias réplicas para luego votar sobre el resultado que éstas producen, entonces es importante evitar que las réplicas puedan sufrir de fallos relacionados que les lleve a producir el mismo resultado. Por esta razón, las réplicas tienen que estar diseñadas para asegurar *independencia de fallos* entre ellas.

Pero incluso si las réplicas se diseñan de tal forma que se asegura independencia de fallos entre ellas, los errores generados por fallos en otros subsistemas se pueden propagar y afectar a varias réplicas. La *propagación de errores* es uno de los mayores problemas a resolver al diseñar un sistema tolerante a fallos, ya que los errores generados por un único fallo pueden corromper diferentes subsistemas hasta y generar un avería global. De hecho, cuando el fallo de un componente o parte del sistema provoca la avería de todo el sistema, se dice que dicho componente o parte representa un *punto único de avería*.

Así pues, es obligatorio que los sistemas tolerantes a fallos incluyan mecanismos de *contención de errores*. Para ello, lo más común es dividir el sistema en diferentes *regiones de contención de errores* [Kok97], es decir, en regiones que pueden ser aisladas para prevenir la propagación de errores desde ellas hacia el resto del sistema.

Hay que tener en cuenta que los mecanismos para tratar errores y tolerar fallos no son perfectos. Esto se debe a que los errores se manifiestan generando una gran cantidad de escenarios y, por tanto, es imposible detectar todos y cada uno de los errores que pueden afectar al sistema. Por esta razón, al diseñar y evaluar sistemas que requieren una dependability elevada, es necesario tener en cuenta la *cobertura* de los mecanismos que el sistema incluye para tolerar fallos. Como ya se ha dicho anteriormente, cuando hablamos de la cobertura de un mecanismo de tolerancia a fallos nos referimos a la probabilidad de que dicho mecanismo tenga éxito.

5.5. Diseño de sistemas tolerantes a fallos

Para desarrollar un sistema tolerante a fallos es necesario el uso de una estrategia práctica y sistemática debido a su gran complejidad. Una de estas estrategias se basa en el paradigma propuesto en [Avi95], el cual consta de los siguientes tres tipos de actividades: *especificación*, *diseño* y *evaluación* .

Las actividades de especificación consisten en determinar la funcionalidad y el grado de *dependability* requerido por el sistema, .e.g el modelo de fallos, nivel

de fiabilidad, etc.

Las actividades de diseño consisten en definir la arquitectura del sistema, es decir, qué subsistemas constituyen el sistema global, cómo interactúan los unos con los otros, y cómo se integran.

Las actividades de evaluación se realizan durante las otras actividades para guiar la ejecución de las mismas, así como para comprobar que cada subsistema (y el sistema final) cumplen con los requisitos funcionales y de dependability.

Existen dos tipos de evaluación, a saber, *evaluación cualitativa*, que tiene como objetivo verificar que el sistema puede lidiar con todos los fallos incluidos en el modelo de fallos; y *evaluación cuantitativa*, cuyo cometido es corroborar numéricamente que el sistema cumple con los requisitos de *dependability*.

Existen diferentes técnicas para llevar a cabo cada tipo de evaluación. Para la evaluación cualitativa se usan técnicas de *chequeo de modelo* (*model checking*) [CGP99], que permiten comprobar formalmente que el sistema exhibe las propiedades que se le atribuyen. Esta técnica consiste en construir un modelo, típicamente, en forma de autómatas interconectados. Una vez el modelo está terminado, se utiliza una herramienta software para realizar una serie de preguntas sobre las propiedades del sistema. Esta herramienta constituye la parte más importante del *model checking* y su cometido es el de analizar exhaustivamente todos los estados posibles del modelo, a fin de determinar si el sistema cumple o no dichas propiedades.

Las técnicas cuantitativas también se basan en la especificación de un modelo. Existen numerosos formalismos para construir dicho modelo. Algunos ejemplos son las *Cadenas de Markov* y las *Redes de Petri* [TMGT93] [Pet81]. Un modelo cuantitativo normalmente incluye parámetros sobre aspectos del sistema (y del entorno del sistema) que pueden influir en la dependability. De esta forma se pueden realizar *análisis de sensibilidad* para dilucidar qué parámetros (y por ende qué aspectos) tienen una mayor influencia. Por ejemplo, el modelo presentado en este proyecto incluye parámetros para caracterizar la cobertura de diferentes mecanismos de tolerancia a fallos, ya que dichas coberturas suelen tener un gran impacto en la dependability.

5.6. Métricas para cuantificar la fiabilidad

Ya que de entre todos los atributos de la dependability, el proyecto se centra en el estudio de la fiabilidad, en esta sección se introducen las métricas más comunes para cuantificarla.

Una de las métricas más utilizadas para cuantificar la fiabilidad es la denominada *Time To Failure* (TTF). Concretamente, si representamos mediante una variable aleatoria X la TTF de un sistema determinado, entonces podemos definir $F(t)$ como la función de distribución acumulativa de X de la siguiente forma:

$$F(t) = Probability(X \leq t)$$

donde $F(t)$ tiene las siguientes propiedades:

$$F(t) = 0 \text{ for } t < 0$$
$$0 \leq F(t_1) \leq F(t_2) \text{ if } t_1 > t_2$$
$$\lim t \to \infty \text{ F(t)} = 1$$

Si nos centramos en la TTF en un intervalo de tiempo infinitesimal, una forma de escribir *F(t)* en términos de su *función de densidad de probabilidad*, $f(t)$, es la siguiente:

$$F(t) = \int_0^t f(t\prime)dt\prime$$

Otra métrica importante es la denominada Función de Fiabilidad $R(t)'$. $R(t)$ se define como la probabilidad de que un sistema funcione correcta e ininterrumpidamente a lo largo del intervalo de tiempo $[0; t]$. $R(t)$ se puede escribir en términos de $f(t)$ de la siguiente forma:

$$R(t) = \int_t^\infty f(t\prime)dt\prime = 1 - F(t)$$

Las funciones $R(t)$ y $f(t)$ permiten el cálculo de una medida importante de la fiabilidad denominada *failure rate* o *tasa de avería*. La tasa de avería de un sistema $\lambda(t)$ es la probabilidad instantánea de que un sistema falle si éste todavía no ha fallado con anterioridad. Su expresión es la siguiente:

$$\lambda(t) = \frac{f(t)}{R(t)}$$

Es importante aclarar la diferencia entre $f(t)$ y $\lambda(t)$. $f(t)\Delta t$ es la probabilidad incondicional de que un sistema falle en el intervalo $[t; t + \Delta t]$; mientras que $\lambda(t)\Delta t$ es la probabilidad de que un sistema falle en el intervalo $[t; t + \Delta t]$, condicionada por el hecho de que el sistema no haya fallado hasta el instante t.

La función $\lambda(t)$ es una de las más utilizadas para caracterizar la fiabilidad de componentes hardware. Si se analiza cómo varía la tasa de averías de un componente hardware en función del tiempo, se obtiene la *curva de mortalidad* de dicho componente [STP96]. Esta curva se divide en 3 regiones: *periodo de*

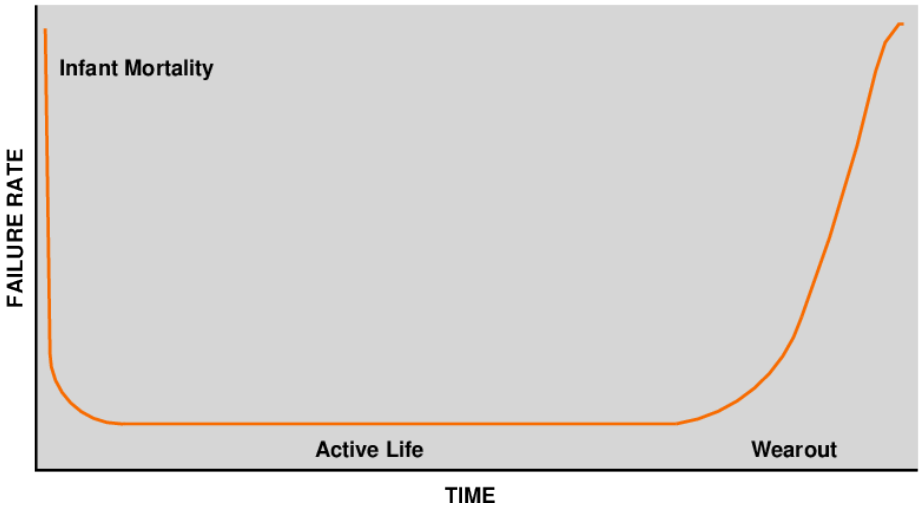

Figura 1: Curva de mortalidad de un componente hardware

mortalidad infantil, *periodo estacionario de operación* y *periodo de desgaste natural*. En el periodo de mortalidad infantil el componente puede exhibir averías relacionadas con imperfecciones o defectos que se deben normalmente a los procesos de manufacturación. Esta región se caracteriza por una tasa de averías alta que decrece rápidamente. El periodo estacionario de operación es el intervalo de tiempo más largo y está caracterizado por presentar la tasa de averías más baja. Si se producen fallos, suelen estar causados por condiciones externas. En el último periodo, el de desgaste natural, la tasa de averías aumenta con el tiempo de forma exponencial. Este último periodo es característico de los componentes mecánicos. Por el contrario, pruebas realizadas sobre componentes electrónicos indican que éstos no exhiben un periodo de desgaste natural, sino que su tasa de averías se mantiene baja y constante durante el resto de su tiempo de vida. Estas regiones se pueden ver en la figura 1 [KNM90].

5.7. Conclusiones

En esta sección se explican algunos conceptos básicos sobre *dependability* que son esenciales para entender el proyecto. Al principio se explica el concepto de dependability y, de entre sus atributos, se define el de fiabilidad, ya que es éste en el que se centra este proyecto.

A continuación se explican los conceptos de fallo, error y avería, y se aclara cuál es su relación de causa-efecto. Posteriormente, se hace una clasificación de las averías y se resalta la importancia de otros conceptos relacionados como son el de modo de fallo, la semántica de avería, el modelo de fallos y la cobertura.

Una vez se han aclarado estos conceptos, se explican brevemente cuáles son los mecanismos mediante los cuales se puede aumentar la dependability. Concretamente, se hace hincapié en los mecanismos de tolerancia a fallos y las dos fases

de operación de dichos mecanismos: el procesamiento de errores y el tratamiento de fallos. Por otra parte, se hace especial énfasis en discutir las características de la tolerancia a fallos sistemática y de los problemas asociados con ella como, por ejemplo, la necesidad de proveer independencia de fallos entre las diferentes réplicas y la importancia de la contención de errores.

Posteriormente, se explica uno de los paradigmas más aceptados para el desarrollo de sistemas tolerantes a fallos, y se explican brevemente cuáles son las actividades de las que consta. De entre ellas, las actividades de evaluación cuantitativa son las que están directamente relacionadas con este proyecto y, por tanto, se resume brevemente cuáles son principales tipos de técnicas utilizadas en este sentido.

Finalmente, se explican cuáles son algunas de las métricas más utilizadas para cuantificar la fiabilidad, y se discuten las propiedades de algunas de ellas.

6. Protocolo CAN

6.1. Introducción

El protocolo CAN (*Controller Area Network*) es una tecnología de comunicación de buses de campo creada a principios de los años 80 por Bosch GmbH en Alemania. Sus puntos fuertes son su bajo coste, configuración simple, mecanismo de arbitraje para acceso priorizado al medio, robustez eléctrica, detección de errores y contención de fallos. Aunque su creación fue pensada para reducir costes de cableado en automoción, se volvió bastante popular en otros sistemas. Hoy en día, se usa para un gran rango de aplicaciones como automatización industrial, robótica, comunicaciones dentro de un edificio, equipamientos médicos, etc.

CAN comprende la capa física y la capa de enlace de datos [ISO03a] del modelo de referencia OSI (*Open Systems Interconnection Basic*). La ISO estandarizó estas dos capas de CAN en el año 1993. Aunque, debido al gran interés en CAN, posteriormente se han especificado otros estándares para CAN.

A continuación se explican los aspectos más importantes de la capa física y de la capa de enlace de datos de CAN. Además, se incluye una especificación de los tipos de fallos que pueden afectar a las redes CAN.

6.2. Capa física

Existen múltiples especificaciones de capas físicas para CAN; pero posiblemente la más popular es la ISO 11898-2 [ISO03b]. Ésta es especificación está pensada para alcanzar velocidades de transmisión de hasta 1 Mbit/s, y es la que se considera en este proyecto al ser una de las más rápidas y utilizadas.

Una red CAN se basa en una topología de bus simple, cuyo medio físico es

una linea de par trenzado que es especialmente resistente a las interferencias electromagnéticas. Para prevenir el reflejo de las señales, ambos extremos de la linea están terminados con impedancias de 120 Ohm, y la sección de cable mediante la cual cada nodo se conecta al bus se configura tan corta como sea posible.

Una de las características más importantes del medio físico de CAN es que dicho medio implementa una función AND cableada de la contribución de todos y cada uno de los nodos. Esta característica es la base de la propiedad de CAN conocida como *transmisión dominante / recesivas* (*dominant / recessive transmission*). Esta propiedad garantiza que si un nodo transmite un valor dominante (es decir, un valor lógico '0') por el bus , todos los nodos conectados al bus recibirán ese valor dominante. En cambio, todos los nodos recibirán un valor recesivo (valor lógico '1') si y solo si todos los nodos transmiten al bus un valor recesivo.

Además, las comunicación en CAN descansa sobre un complejo *mecanismo de sincronización a nivel de bit* que garantiza que todos los nodos leen cuasi-simultáneamente cada bit que se transmite en el bus. A esta propiedad se la conoce con el nombre de (*response en tiempo de bit*) *in-bit response*. Este mecanismo de sincronización utiliza la propiedad de transmisión dominante/recessiva para conseguir que todos los nodos se mantengan sincronizados, bit a bit, con el nodo que transmite, es decir, el denominado como *transmisor líder* (*leading transmitter*).

La *respuesta en tiempo de bit* implica que solo cuando la transmisión del bit actual atraviesa todo el bus y se estabiliza eléctricamente, es posible enviar el siguiente bit. Así pues, la sincronización bit a bit de CAN fuerza una relación inversamente proporcional entre la velocidad de transmisión y la longitud del bus. Por ejemplo, si una red CAN opera a su máxima velocidad de transmisión, 1 Mbit/s, la longitud máxima que puede alcanzar el bus es de 40 metros [CiA]. Si bajamos esta velocidad de transmisión, se pueden alcanzar longitudes más grandes, por ejemplo a velocidades de 125 Kbit/s o 10Kbit/s se alcanzan longitudes de 500 metros y 5 kilómetros respectivamente [CiA].

Aunque la sincronización a nivel de bit en CAN limita la longitud del bus y la velocidad de transmisión, dicha sincronización permite la utilización de algunos mecanismos que se describirán en la capa de enlace de datos.

6.3. Capa de enlace de datos

La capa de enlace de datos de CAN ofrece una serie de mecanismos que permiten a los nodos del sistema enviar y recibir datos aún cuando el bus está afectado por errores transitorios e, incluso, cuando los los nodos están afectados por fallos permanentes. En esta sección se describen algunos de estos mecanismos.

Formato de la trama

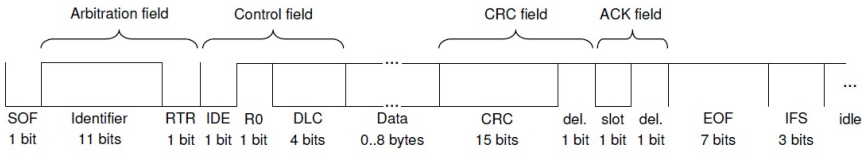

Figura 2: Formato de *trama de datos* de CAN 2.0 A

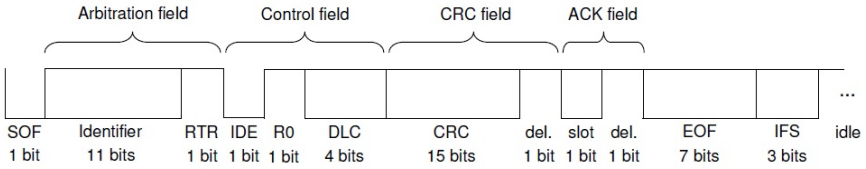

Figura 3: Formato de *trama remota* de CAN 2.0 A

CAN incluye cuatro formatos de trama diferentes: *tramas de datos*, *tramas remotas*, *tramas de errores* y *tramas de sobrecarga*. Las dos primeras tramas se explicarán a continuación, mientras que el resto de tramas se abordarán en las secciones siguientes.

Un nodo usa una *trama de datos* para transmitir datos, y usa una *trama remota* para solicitar datos a otro nodo. El formato de las tramas de datos y remotas es bastante parecido entre sí, tal y como se puede ver en las figuras 2 y 3. El formato de trama que se muestra en estas figuras es el conocido como CAN 2.0 A, aunque existe otro tipo de formato llamado CAN 2.0 B, el cual especifica lo que se conoce como trama extendida. Básicamente, la diferencia reside en el hecho de que las tramas CAN 2.0 A tienen un identificador de 11 bits, mientras que las tramas CAN 2.0 B tienen un identificador de 23 bits. Sólo se describirá el formato de tramas CAN 2.0 A por ser el más común.

Cuando el canal no está ocupado cualquier nodo puede empezar a transmitir enviando un bit dominante llamado SOF (*Start Of Frame* o *comienzo de trama*). Todos los nodos que escuchan el bit SOF se sincronizan con dicho bit y, por tanto, con el nodo transmisor. Sin embargo, se puede dar el caso que varios nodos envíen el bit SOF simultáneamente. En esta situación, todos los nodos que quieren transmitir continúan con su propia trama y tiene lugar un arbitraje. Además, un nodo puede unirse a la transmisión en el instante de bit después de leer un SOF (sin enviar su bit SOF de la trama), y así, unirse al arbitraje.

El arbitraje se resuelve mediante el conocido como *mecanismo de arbitraje bit a bit* (*bit-wise arbitration mechanism*), el cuál se basa en el *arbitatrion field* o *campo de arbitraje* de la trama. Este campo contiene el *identifier* o *identificador* y el RTR (*Remote Transmission Request* o *petición remota de transmisión*). El mecanismo asegura que después de la transmisión del *campo de arbitraje*, solo un nodo podrá transmitir. Este mecanismo se explica más detalladamente en el siguiente apartado.

A continuación, se transmite el campo *control field* o *campo de control*, el cuál está constituido por el IDE (*IDentifier Extension* o *extensión de identificación*), un bit reservado llamado RO y el *Data Lenght Code* o *código de longitud de datos*. El bit IDE indica si el formato de la trama es el CAN 2.0 A (IDE = '0'), o el CAN 2.0 B (IDE = '1'). El bit R0 está reservado para futuras extensiones del protocolo. Los 4 bits del DLC indican el número de bits que contendrá el *campo de datos* en caso de que sea una *trama de datos*, o el número de bits que se piden si se trata de una *trama remota*.

Para comprobar la integridad de los datos que se han transmitido, el transmisor envía un campo de 16 bits. Los 15 primeros bits de este campo contienen un CRC (*Cyclic Redundancy Code* o *código de redundancia cíclica*), mientras que el último bit es un valor recesivo que delimita dicho CRC.

El *ACK field* o *campo de ACK*, se usa por parte de los receptores para indicar si consideran que la parte de la trama que se ha recibido hasta ese momento es correcta o no. Durante el primer bit de este campo el nodo transmisor envía un bit recesivo. Por su parte, cada nodo receptor envía un bit dominante si no ha detectado ningún error hasta el momento, o un bit recesivo en caso contrario. Gracias a la propiedad de transmisión recesiva/dominante, el nodo transmisor sabe que si observa un bit recesivo en el primer bit del campo de ACK, entonces eso quiere decir que ningún receptor ha podido leer la trama. En cambio, si recibe un valor dominante, el transmisor sabe que al menos algún nodo ha podido leerla. El segundo bit ACK es simplemente un bit que delimita el final del campo de ACK.

Por último, la trama finaliza con los bits del campo EOF (*End Of Frame* o *final de trama*), que consisten en 11 bits recesivos. A continuación siguen otros 3 bits recesivos llamados IFS (*Intermission Frame Space* o *espacio entre tramas*). Tras esos tres bits, todos los nodos consideran que el canal está libre para iniciar una nueva transmisión.

Mecanismo de arbitraje bit a bit

Como se ha apuntado anteriormente, cuando más de un nodo quiere transmitir al mismo tiempo, se decide cuál es el nodo que gana el acceso al medio mediante el *mecanismo de arbitraje bit a bit*. Concretamente, cada nodo transmisor chequea cuál es el valor de bit que se observa realmente en el bus durante el campo de arbitraje. Si un nodo transmisor monitoriza un bit dominante, '0', cuando él está transmitiendo un bit recesivo, '1', entonces asume que ha perdido el arbitraje, adquiere el rol de nodo receptor, y pospone la transmisión de la trama que quería enviar para después de la trama actual.

Al final de la fase de arbitraje, el nodo que transmitía la trama con el identificador más bajo es el único que continua transmitiendo. También es importante notar que el valor del bit RTR del campo de arbitraje vale '0' en una trama de datos y '1' en una trama remota. Por tanto, si un nodo intenta transmitir una trama de datos y otro nodo intenta enviar una trama remota con el mismo identificador, la trama de datos tiene preferencia sobre la remota.

Codificación de la trama

CAN usa una codificación de tipo NRZ (*non-return-to-zero*), de tal forma que el nivel de la señal se mantiene en el mismo valor durante todo el tiempo del bit. Sin embargo, como se ha mencionado anteriormente, los nodos usan las transiciones de recesivo a dominante en la señal que se observa en el medio para mantenerse sincronizados a nivel de bit con el transmisor lider. Por ello, para mantener una correcta sincronización, es necesario limitar el tiempo que la señal se mantiene en el mismo valor.

Para conseguir limitar este tiempo adecuadamente, la trama se codifica utilizando la denominada *regla de relleno* (*stuff rule*). Esta regla especifica que el transmisor debe transmitir un bit complementario, llamado *bit de relleno* (*stuff bit*), cada vez que transmita 5 bits consecutivos con la misma polaridad (incluyendo cualquier bit de relleno anterior). Toda la trama se codifica con esta regla, excepto el delimitador del CRC, el campo de ACK y el campo EOF.

Detección y señalización de errores

Cada nodo CAN incluye una serie de mecanismos que permiten detectar errores, bit a bit, en el flujo de bits que se observa en el bus. Todo nodo CAN puede detectar 5 tipos de errores diferentes, los cuales son [ISO93]: *error de relleno*, *error de formato*, *error de bit*, *error de CRC* y *error de ACK*.

Para detectar los errores anteriores se usan una serie de *mecanismos de detección de errores* que comprueban que la trama que el nodo transmite o recibe sea correcta. Estos mecanismos son los siguientes respectivamente [ISO93]: *comprobación de la regla de relleno*, *comprobación de la trama*, *monitorización*, *comprobación del CRC* y *comprobación del ACK*. Cada nodo utiliza estos mecanismos como se indica a continuación:

- **Error de relleno:** Tanto el transmisor como los receptores llevan a cabo una *comprobación de la regla de relleno* para cerciorarse de que la trama que observan en el bus cumple la *regla de relleno*; es decir que no se observan en la trama más de 5 bits consecutivos con la misma polaridad.

- **Error de trama:** Tanto el transmisor como los receptores realizan una *comprobación de la trama* para chequear si ésta obedece las reglas de formato. Estas reglas definen las características de cada campo de la trama: orden dentro de la trama, su longitud y los valores de bit permitidos.

- **Error de bit:** Cuando un nodo envía un bit dominante, hace una *monitorización* del bit que se observa en el bus para comprobar que, efectivamente, en el canal se observa dicho bit dominante. Además, cuando el nodo juega el papel de transmisor (excepto en el bit de ACK), también comprueba que todo bit recesivo que envíe se observe en el canal efectivamente como un bit recesivo.

- **Error de CRC:** Como se ha explicado antes, el nodo transmisor calcula un CRC de 15 bits basado en los bits del frame que ha transmitido y, a continuación, transmite dicho CRC en el antepenúltimo campo de la trama. Los nodos receptores también calculan el CRC a medida que van recibiendo los bits de la trama. De esta forma, cuando un nodo receptor recibe el CRC del transmisor, realiza una *comprobación de dicho CRC* para chequear que éste es idéntico al CRC que él mismo ha calculado.

- **Error de ACK:** Como se ha explicado anteriormente, todo nodo receptor transmite un valor dominante en el bit de ACK, si quiere confirmar que ha recibiendo la trama correctamente hasta ese momento. Por su parte, el nodo transmisor hace una *comprobación del ACK* para determinar si el valor observado en el bus durante dicho bit es dominante o no y, por lo tanto, para detectar si al menos un nodo está recibiendo correctamente la trama.

Dependiendo del número de errores que haya detectado hasta un instante de tiempo determinado, un nodo puede estar en tres *estados de error* diferentes, a saber: estado de *error activo*, estado de *error pasivo* o estado de *fuera del bus*.

Cuando un nodo detecta un error inicia un *mecanismo de señalización de errores* para notificarle, a todos y cada uno de los otros nodos, que ha encontrado un error. Para ello, el nodo empieza a transmitir una trama especial conocida como *trama de error*. Concretamente, el nodo comienza a transmitir la trama de error en el bit que sucede a aquel en el que detectó el error; excepto si detectó un error de CRC, en cuyo caso empieza a transmitir la *trama de error* en el primer bit del campo EOF.

Una *trama de error* esta formada por un *señalizador de error* (*error flag*) seguido de un *delimitador de error* (*error delimiter*). El contenido del señalizador de error depende del estado de error del nodo. Un nodo en error activo transmite lo que se denomina un *señalizador de error activo*, el cuál consta de 6 bits dominantes consecutivos. Este tipo de señalizador viola la regla de relleno, forzando así a los otros nodos a detectar un error y, en consecuencia, a rechazar la trama que se estaba transmitiendo. Es decir, el señalizador activo de error provoca una *globalización del error* o *error global*.

Por contra, si el nodo que detecta el error está en estado de error pasivo, transmite un *señalizador de error pasivo*, el cuál está formado por 6 bits recesivos consecutivos. En este caso, el nodo no siempre puede forzar a que los otros nodos detecten el error y, por consiguiente, tampoco puede forzar la globalización del error.

Por último, un nodo cuyo estado de error es el de fuera del bus, no envía ninguna trama de error cuando detecta un error y, por tanto, no puede informar al resto de nodos de los errores que observa en el bus.

Si el *señalizador de error* produce un *error global*, todos los nodos transmiten su propio *señalizador de error* y, a continuación, transmiten conjuntamente el *delimitador de error*. Por lo tanto, la apariencia final de una *trama de error* es la superposición de los *señalizadores de error* enviados por los diferentes nodos,

seguido de un *delimitador de error* conjunto.

Este *delimitador de error* conjunto está formado por un mínimo de 8 bits recesivos consecutivos y se forma como se explica a continuación. Después de transmitir su propio señalizador de error, cada nodo envía su delimitador de error, el cual está constituido por bits recesivos consecutivos, hasta que consigue monitorizar en el bus un patrón de 8 bits recesivos consecutivos. Este patron indica el final del error delimiter y, por tanto, del frame de error. Es importante notar que como los bits dominantes sobreescriben a los recesivos, todos los nodos monitorizarán bits dominantes mientras transmiten sus propios delimitadores de error mientras exista un nodo que todavía esté enviando su señalizador de error. Por tanto, todos los nodos observarán el inicio y el final del patrón de 8 bits recesivos consecutivos al mismo tiempo y, por consiguiente, todos detectarn cuasi-simultáneamente el final del frame de error. Esto quiere decir que tras la globalización de un error, todos los nodos quedan sincronizados entre sí, de forma que todos consideran al mismo tiempo que el canal está disponible para comunicar.

Notar que el funcionamiento del mecanismo de señalización y globalización de errores está diseñado para garantizar la propiedad de *consistencia de datos* en CAN. Según esta propiedad, en CAN se garantiza que cada trama que se transmite es aceptada por todos los nodos o por ninguno de ellos [ISO93]. Esta propiedad es muy importante en los sistemas distribuidos que requieren una dependability alta. Por esta razón, el hecho de que CAN provea consistencia de datos se consideró durante mucho tiempo como una de las principales ventajas de este protocolo. De todas formas, posteriormente se ha demostrado que CAN no garantiza consitencia de datos siempre, ni siquiera para los nodos que estén en el estado de error activo [RVAR98] y [JJJ00].

Tratamiento de fallos

El estándar CAN especifica algunos mecanismos de tratamiento de fallos para diagnosticar y desactivar los fallos que ocurran en los nodos del sistema. Específicamente cada nodo CAN incluye dos contadores de error [ISO93]: el TEC (*Transmission Error Counter* o *contador de error de transmisión*) y el REC (*Reception Error Counter* o *contador de error de recepción*). Dependiendo de los valores de estos contadores, el nodo cambiará su estado de error y, por tanto, sus restricciones a la hora de transmitir y recibir, con el fin de minimizar el impacto negativo que él pueda ejercer en la comunicación.

En general, un nodo CAN incrementa su TEC/REC cada vez que detecta un error en el canal. Existen reglas especificas [ISO93] para decidir cómo se incrementan estos contadores. Básicamente, estas reglas penalizan más a los errores detectados durante la transmisión que a los detectados en la recepción; ya que el impacto negativo que un nodo averiado puede ejercer en la comunicación es generalmente mayor cuándo transmite. Concretamente, cuando se detecta un fallo en la transmisión, el contador TEC incrementa su valor en 8 unidades; mientras que si el fallo se detecta en la recepción, el valor del contador REC se incrementa en 1 unidad.

Además, el nodo incrementa su contador TEC o REC en 8 unidades si sospecha que él mismo es el responsable del error. Esto sucede si detecta un *error primario* [ISO93], es decir, si detecta cierto patrón durante la señalización y la globalización del error que le hace sospechar que él fue el que inició la señalización (lo cuál es un indicativo de que el error que detectó fue local).

Por otro lado, también existen reglas para disminuir los contadores TEC y REC. Normalmente los contadores TEC y REC se disminuyen en una unidad cada vez que se ha transmitido o recibido correctamente una trama respectivamente (hasta llegar a un valor mínimo de 0). La única excepción a esta regla se da en el caso del REC cuándo éste es mayor que 127. En tal caso, no se disminuye el REC en una unidad, sino que su decrementa estableciéndolo en un valor comprendido entre 119 y 127.

Cada nodo tiene en cuenta el valor de sus contadores TEC y REC para decidir el estado de error en el que se encuentra (activo, pasivo o fuera del bus) y, así, reducir el impacto negativo que él mismo puede provocar en la comunicación. Inicialmente un nodo CAN empieza en el estado de error activo, lo cual implica que puede señalizar y globalizar errores sin ninguna restricción. Si alguno de sus contadores de error alcanza un valor mayor a 127 unidades, el nodo cambia automáticamente al estado de error pasivo. La diferencia entre este estado y el anterior consiste en el formato del *señalizador de error* que el nodo puede enviar. Como se ha explicado antes, el formato del señalizador de error de un nodo en el estado de error pasivo es tal que no puede forzar al resto de nodos a detectar un error y, por tanto, no garantiza que el nodo globalize el error que ha detectado. De esta forma se reduce significativamente el impacto negativo que un nodo puede ejercer en las comunicaciones entre los otros nodos, cuando dicho nodo detecta errores localmente. Por último, si el contador TEC supera el valor 255, el nodo entra en el estado de fuera del bus. En este estado, el nodo se diagnostica a sí mismo en fallo permanente y no transmite nada hacia el bus. De esta forma, se consige aislar a dicho nodo del sistema.

Finalmente, para reintegrar a un nodo que realmente no sufre un fallo permanente, éste puede volver al estado de error activo tanto desde el estado de error pasivo, como desde el estado de fuera del bus [ISO93]. Específicamente, un nodo en estado de error pasivo retorna al estado activo cuando sus dos contadores, TEC y REC, tienen un valor inferior a 127. Por su parte, un nodo en el estado fuera del bus retorna al estado activo cuando observa en el bus 128 ocurrencias de un patron de bits denominado *bus-free*, el cuál consta de 11 bits recesivos consecutivos.

Señalización de sobrecarga

CAN especifica dos tipos de condiciones de sobrecarga [ISO93]. La primera condición se da cuando un nodo receptor necesita un tiempo extra antes de que otra trama pueda ser transmitida por el bus. Cuando esto ocurre, el nodo lo señaliza transmitiendo una *trama de sobrecarga* en el primer bit del *Intermission Frame Space* (IFS). Esta trama consiste en un *flag de sobrecarga* constituido por 6 bits dominantes consecutivos, seguido de un *delimitador de sobrecarga*

que consta de al menos 8 bits recesivos consecutivos. La segunda condición ocurre cuando un nodo detecta un bit dominante en alguno de los bits del IFS. En este caso, el nodo debe reaccionar transmitiendo una trama de sobrecarga. Esta segunda condición es la que permite a un nodo globalizar una situación de sobrecarga provocada por la primera condición.

Es importante notar que este mecanismo de globalización es idéntico al mecanismo de globalización de una trama de error. De hecho, los nodos transmiten cooperativamente el delimitador de sobrecarga como en el caso de una trama de error; por lo que el formato de la trama de sobrecarga que resulta de la señalización de sobrecarga por parte de todos los nodos es idéntica al formato de la trama de error que se observa en el bus cuando un error se globaliza a partir de flag de error activo.

6.4. Tipos de fallos en redes CAN

Los fallos que ocurren en diferentes componentes de una red CAN pueden manifestarse de muchas maneras. Por desgracia, los mecanismos de detección de errores y tratamiento de fallos en el protocolo de CAN tienen serias limitaciones. Además CAN no incluye ningún mecanismo para tolerar fallos. Esto conlleva a que un solo fallo pueda ser suficiente para causar un fallo global en las comunicaciones. Para entender mejor cómo los fallos pueden influir en las comunicaciones de una red CAN, hemos clasificado los fallos en *fallos sintácticos* y *fallos semánticos*.

Un fallo sintáctico es aquél que generar errores que corrompen los valores de los bits que se están retransmitiendo por el medio, y por lo tanto, hacen que las tramas CAN sean incorrectas desde un tipo de vista sintáctico. Estos tipos de fallos se clasifican en dos categorías:

- **Fallo Stuck-at:** Ocurre cuando un componente falla de forma que transmite constantemente el mismo valor de bit. Existen dos tipos de fallos *stuck-at*: *stuck-at-dominant*, cuando el valor que se transmite constantemente es dominante; y *stuck-at-recesive* cuando este valor es recesivo. Dado que el medio físico de CAN equivale a una AND lógica de lo que se transmite en el bus, un nodo que falle de forma stuck-at sólo puede bloquear el bus si los errores que transmite son de tipo *stuck-at-dominant*. En cambio, si un fallo stuck-at ocurre en algún componente del medio, en sí, éste queda inutilizado para la comunicación independientemente de si el fallo es de tipo *stuck-at-dominant* o *stuck-at-recessive*.
- **Fallo Bit-Flipping:** Ocurre cuando un componente falla transmitiendo bits erróneos cuyo valor es aleatorio. Los bits dominantes de una secuencia de errores de tipo bit-flipping pueden corromper cualquier trama que se esté transmitiendo en el bus. Por tanto, una fallo bit-flipping bloquea las comunicaciones si es permanente.

Por otro lado, un fallo de tipo semántico es aquél que corrompe el contenido de una trama de tal forma que ésta sigue siendo correcta desde el punto de

vista sintáctico, pero incorrecta desde el punto de vista de la aplicación. Por ejemplo, un tipo de fallo de este tipo se da cuando un nodo transmite tramas que son incorrectas desde el punto de vista temporal, por ejemplo, tramas que se transmiten demasiado pronto o demasiado tarde. Otro ejemplo muy conocido en el contexto de los sistemas distribuidos empotrados de tiempo real es el denominado como *babbling-idiot* o *balbuceo idiota*. Un fallo *babbling-idiot* se produce cuando un nodo envía un mensaje con una frecuencia tal que consume más recursos de los necesarios; disminuyendo así el ancho de banda disponible para el resto de nodos [BB03]. Un fallo *babbling-idiot* puede ser causado por diferentes razones. Por ejemplo, un fallo que afecte internamente a un controlador CAN y haga que éste siempre tenga una trama pendiente de transmitir. Otra posibilidad es que se de un fallo software que provoque que un nodo ejecute un bucle infinito de forma que transmita un mensaje en cada iteración.

Finalmente es importante notar que los mecanismos de detección de errores y de tratamiento de fallos que provee el protocolo CAN apenas pueden hacer frente a fallos semánticos y, además, tal y como se explicará en las siguientes secciones, presentan limitaciones importantes para lidiar con fallos sintácticos. Muchas de estas limitaciones se derivan de su topología de bus simple, en la cuál los diferentes componentes del sistema se interconectan sin incluir mecanismos adecuados de contención de errores y tolerancia a fallos. El trabajo realizado en este proyecto y en su contexto se centra en cómo mejorar la capacidad de CAN para tratar y tolerar fallos sintácticos y, de esta forma, mejorar la fiabilidad de sistemas basados en él. Los fallos semánticos quedan excluidos de este trabajo porque para tratarlos y tolerarlos se necesitan mecanismos en niveles de la arquitectura por encima de CAN.

6.5. Conclusiones

El protocolo CAN es un sistema de comunicación basado en un bus de campo que ha demostrado ser ideal para una gran variedad de sistemas de control distribuidos. Parte de su éxito se basa en el equilibrio entre su fiabilidad y su precio.

En este capítulo se han descrito las características principales del protocolo CAN. Se ha puesto especial énfasis en el hecho de que su medio físico es equivalente a una función AND, y a las propiedades que se derivan de ello. Concretamente, se ha explicado que gracias a la transmisión basada en valores recesivos y dominantes, así como a la propiedad de in-bit response, CAN puede proveer una serie de mecanismos que presentan ventajas importantes para la fiabilidad.

Sin embargo, también se ha explicado brevemente que CAN adolece de limitaciones importantes para tratar y tolerar fallos. De entre los diferentes tipos de fallo que un sistema CAN puede sufrir, se ha puesto el foco en los llamados stuck-at y bit-flipping. Estos fallos se pueden tratar independientemente de la aplicación y, por tanto, son los que se abordan en el contexto de este proyecto.

También se ha apuntado el hecho de que muchas de estas limitaciones se

deben a la topología de bus simple de CAN, la cual no permite implementar mecanismos de contención de errores y tolerancia a fallos suficientemente avanzados. Por tanto, a pesar de las buenas características de CAN, sus limitaciones invitan al estudio de otras topologías que permitan mejorar dichos mecanismos.

7. CANcentrate, ReCANcentrate y el bus replicado de CANEly

7.1. Introducción

Tal y como se ha explicado en la Sección 6, la topologia de bus simple del protocolo CAN limita sus mecanismos de tratamiento y tolerancia a fallos. Por esta razón, en los últimos años se han propuesto diferentes topologías y soluciones cuyo objetivo es mejorar la dependability de CAN.

En esta sección se introducen los aspectos esenciales de tres de las soluciones más relevantes que se han propuesto para mejorar la fiabilidad de CAN, a saber: la topología de estrella simple llamada CANcentrate, la topología de estrella replicada llamada ReCANcentrate, y la topología de bus replicado propuesta en una capa de servicios de comunicación basada en CAN llamada CANELy *CAN Enhanced Layer*.

7.2. CANcentrate

CANcentrate es una topología en estrella simple para CAN, cuyo hub provee mecanismos avanzados de detección de errores y tratamiento de fallos. Concretamente, en CANcentrate, cada nodo se conecta al *hub* mediante un *link* dedicado que contiene un *uplink* y un *downlink*, como se puede ver en la figura 4. El hub recibe cada contribución de cada nodo por su correspondiente uplink, acopla todas las contribuciones, y retransmite el resultado a través los downlinks.

La figura 5 muestra la arquitectura hardware interna del hub, que contiene 3 módulos: el *Modulo de acoplamiento*, el *Módulo de entra y salida*, y el *Módulo de tratamiento de fallos*. El Módulo de acoplamiento recibe las contribuciones de los puertos ($B_{1..n}$) y las acopla mediante una puerta lógica AND, generando lo que se conoce como *Señal resultante* B_0. A continuación, transmite la señal B_0 de vuelta a los nodos. Es importante notar que la puerta lógica AND reemplaza la función de AND cableada característica del medio físico del bus CAN. Esto quiere decir que el valor que se observa en la Señal resultante es equivalente al valor que se observaría en un bus CAN. Por ello, las tramas que se observan en B0 se denominan *Tramas resultantes*.

El Módulo de acoplamiento incluye también una puerta OR por cada una de las contribuciones de los puertos. Como se explicará más adelante, cada una de estas puertas permite aislar a un puerto concreto cuando éste está permanentemente averiado.

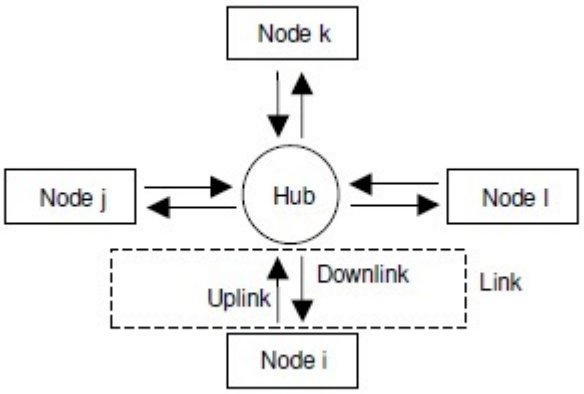

Figura 4: Esquema de conexión CANcentrate

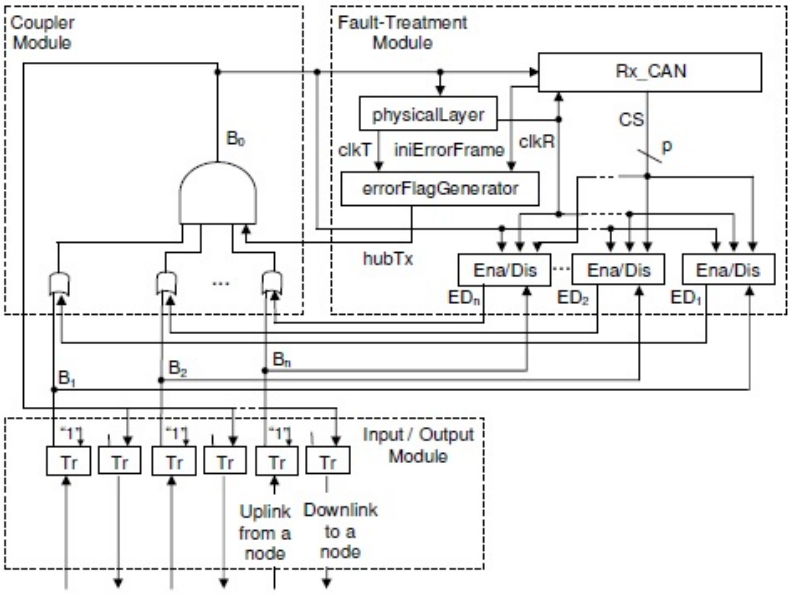

Figura 5: Arquitectura interna del hub de CANcentrate

El Módulo de entrada y salida contiene los *transceivers* que convierten las señales físicas provenientes de los uplinks a valores lógicos que los otros módulos internos del hub pueden procesar. Y viceversa, los transceivers también traducen los valores lógicos de B0 a las señales físicas que se envían a través de los downlinks.

El Módulo de tratamiento de fallos contiene el *Módulo de recepción de CAN* (Rx_CAN) y una serie de *Unidades habilitadoras / deshabilitadoras*. El Módulo de recepción CAN observa la señal B0 para sincronizar al hub, a nivel de bit y de trama, con la Señal resultante. A partir de esta sincronización, el Módulo de recepción CAN genera una serie de señales que indican cuál es el *Estado presente* de la trama resultante. Básicamente, este estado describe el significado de cada bit dentro de la trama que se transmite, por ejemplo, cuál es la posición del bit dentro de la trama, a qué campo pertenece, o si el el bit es un bit de relleno o no.

Cada una de las Unidades habilitadoras / deshabilitadoras monitoriza la contribución de un puerto concreto con el fin de detectar errores y decidir si el puerto debe ser deshabilitado. Por un lado, la Unidad habilitadora / deshabilitadora es capaz de contar el número de bits consecutivos de la misma polaridad para diagnosticar cuando la contribución de su puerto sufre de un fallo stuck-at-dominant. Por otro lado, cada una de estas unidades usa el estado presente de la trama resultante, las reglas del protocolo CAN (p.e. la regla de relleno), y el rol actual que juega el nodo conectado al puerto que supervisa (transmisor o receptor), para determinar cuáles son los valores de bit que el nodo está autorizado a enviar en el siguiente tiempo de bit. Si la unidad observa que desde su puerto se recibe un bit cuyo valor no está autorizado, incrementa un contador de errores asociado. Uno de estos contadores le permite a la unidad registrar el número de veces que un nodo receptor no ha transmitido el bit de ACK y, de esta forma, detectar cuando la contribución del puerto está stuck-at-recessive. Otro contador, le permite a la unidad registrar cuantos bit-flippings han tenido lugar en el puerto. Cuando alguno de los contadores de error de un puerto llega a un determinado límite, la Unidad habilitadora / deshabilitadora diagnostica que el puerto sufre un fallo permanente (stuck-at-dominant, stuck-at-recesive o bit-flipping) y, acto seguido, utiliza la *Señal de habilitación / deshabilitación* (ED_i) para introducir un valor lógico '1' en la puerta OR correspondiente dentro del Módulo de acoplamiento. Esta acción aisla la contribución de ese puerto de la puerta AND y, por tanto, contiene la propagación de errores desde él hacia el resto del sistema.

La Unidad habilitadora / deshabilitadora también decrementa los valores de los contadores de errores después de no observar ningún error en el puerto correspondiente durante unos periodos predefinidos. Esto permite reintegrar puertos que realmente no están permanente averiados o que han sido reparados.

Es importante notar que usar uplinks y downlinks de forma separada para cada nodo permite separar las contribuciones de cada uno de ellos de la Señal resultante. Esto hace que las Unidades habilitadoras / deshabilitadoras puedan monitorizar individualmente la contribución de cada nodo y detectar errores con una precisión que es imposible en un bus CAN. Es decir, el hub de CANcentrate tiene una visión privilegiada de las comunicaciones y, por tanto, puede

diagnosticar fallos permanentes con más celeridad, así como determinar con más precisión qué nodo o parte del medio presenta un fallo.

Por último, hay que destacar que CANcentrate es totalmente compatible con componentes hardware CAN estándar, y es transparente para cualquier aplicación o protocolo basado en CAN.

7.3. ReCANcentrate

ReCANcentrate se presenta como una extensión a CANcentrate para proveer mecanismos de tolerancia a fallos, conservando la misma compatibilidad con componentes hardware y software CAN estándar.

La tolerancia a fallos de ReCANcentrate se consigue básicamente mediante una topología de estrella replicada. Concretamente, aunque ReCANcentrate no limita el número de hubs, para simplificar la explicación se describirá la arquitectura de una red ReCANcentrate con 2 hubs.

La estrategia de conexión de ReCANcentrate es similar a la de CANcentrate; es decir, cada nodo se conecta a cada uno de los hubs mediante un uplink y un downlink (figura 6). Además, para simplificar la gestión que los nodos tienen que hacer del tráfico (replicado) que reciben de los hubs, en ReCANcentrate se fuerza a los hubs a transmitir en paralelo el mismo tráfico, bit a bit, hacia los nodos. Para conseguir transmitir exactamente el mismo tráfico desde los dos hubs, éstos se intercambian el tráfico que reciben de los nodos conectados directamente a ellos. Para hacer ester intercambio, los hubs utilizan dos o más links dedicados llamados *interlinks*. Cada interlink contiene dos *sublinks* independientes, uno para cada dirección. Usar más de un interlink permite a la red tolerar averías en los interlinks.

Es importante notar que la Señal resultante que ambos hubs transmiten a los nodos es idéntica, y contiene las contribuciones de todos los nodos que al menos tienen una conexión no averiada con al menos un hub, sin importar cuál. Esta propiedad fuerza a que los nodos tengan una visión consistente tanto del tráfico, como de los nodos que participan en la comunicación; aún cuando algunos de los nodos han perdido su conexión con uno de los hubs. De hecho, para reducir costes en el cableado, ReCANcentrate hace posible conectar aquellos nodos que sean menos críticos a uno solo de los hubs.

Internamente, el hub de ReCANcentrate es similar al de CANcentrate, como se muestra en la figura 7. Las mayores diferencias estriban en que el Módulo de acoplamiento en el hub de ReCANcentrate realiza el acoplamiento en dos fases. En la primera fase cada hub acopla las contribuciones de sus propios nodos (los nodos conectados a ese hub en concreto) y obtiene su propia Señal resultante, B_0. En ReCANcentrate a la señal B_0 de cada hub no se le llama realmente Señal resultante, sino que se le denomina como la *Contribución de ese hub*. Cada hub genera dos (o más) replicas de su contribución, p.e. B_{00} y B_{01}, y las transmite al otro hub mediante un sublink de cada interlink. En la segunda fase, cada hub acopla su propia contribución, B_0 con las replicas de la contribución que recibe del otro hub. Así cada hub obtiene la misma Señal resultante, ahora identificada

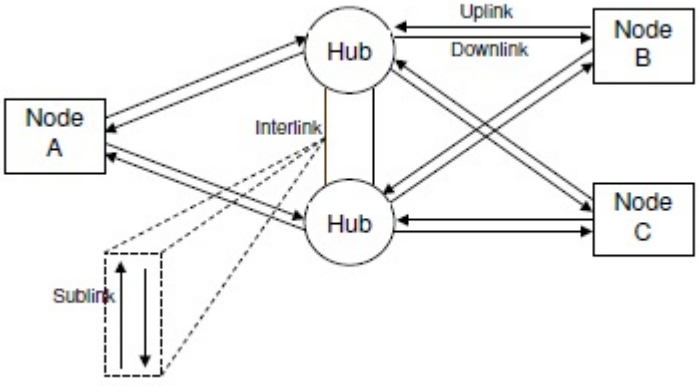

Figura 6: Esquema de conexión ReCANcentrate

mediante la etiqueta *BT* en la figura, y la transmite a través de los donwlinks conectados directamente a él.

Otra de las diferencias con respecto al hub de CANcentrate es que en ReCANcentrate cada hub incluye una serie de *Unidades de habilitadoras / deshabilitadoras* adicionales. Cada una de ellas monitoriza la contribución del otro hub recibida desde uno de los sublinks con el fin de detectar errores y, si es necesario, aislar a dicho sublink. Estas unidades difieren de las Unidades de habilitación / deshabilitación de los puertos esencialmente en los umbrales que aplican a los contadores de errores respectivos.

En cuanto a los nodos de ReCANcentrate, típicamente cada uno se conecta a cada hub mediante un controlador CAN dedicado. De esta forma, para detectar cuándo un hub está averiado, el nodo puede utilizar los mecanismos nativos ya incluidos en el controlador CAN correspondiente. Es más, el uso de dos controladores CAN le permite a cada nodo tolerar el fallo de uno de sus controladores CAN (sin importar cuál).

Es importante notar que ReCANcentrate permite tolerar fallos que afecten a uno de los hubs (no importa a cuál), a los interlinks, a los links (cada nodo se puede comunicar mientras tenga al menos un link correcto conectado a un hub correcto), y a los controladores CAN.

7.4. Bus replicado de CANELy

Otra de las topologías que mayor interés han despertado para aumentar la dependability de los buses de campos en general, y de CAN en particular, es el bus replicado. De hecho, la mayoría de buses replicados que se han propuesto para CAN son anteriores a las estrellas CANcentrate y ReCANcentrate.

Uno de los buses replicados más interesantes es el propuesto en el contexto de CANELy (*CAN Enhanced Layer*) [**?**]; el cuál consiste en una serie de servicios

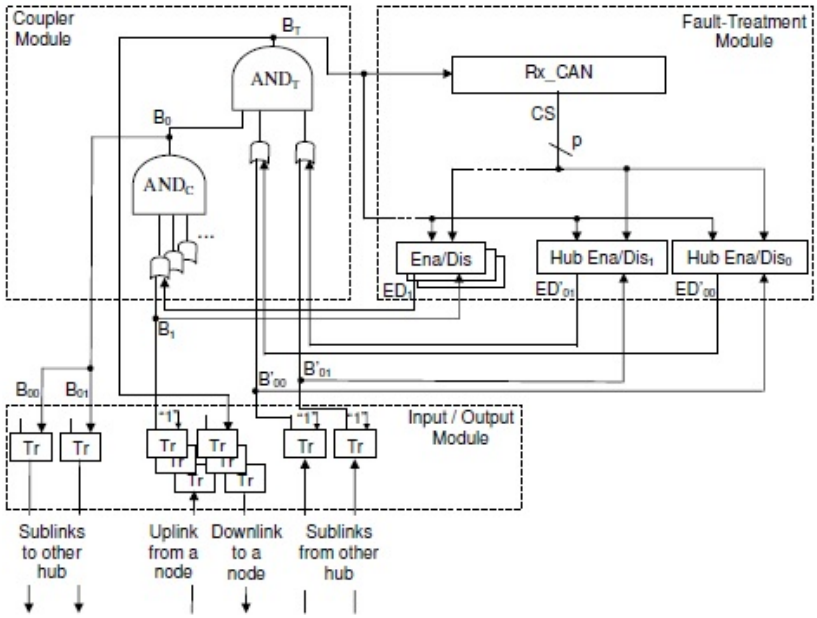

Figura 7: Arquitectura interna del hub de ReCANcentrate

dispuestos en capas que permiten mejorar la dependability y las garantías de comunicación en tiempo real de CAN. Es importante aclarar que CANELy y su bus replicado [RVA99] son anteriores a la existencia de CANcentrate y ReCANcentrate. De hecho, como se verá a continuación, los mecanismos de acoplamiento y contención de errores de los hubs de estas dos estrellas están inspirados en los mecanismos del bus replicado de CANELy.

También es importante destacar que el bus replicado de CANELy es especialmente interesante porque incluye menos hardware que otros buses redundantes (con la consiguiente ganancia en dependability y coste), y permite gestionar el tráfico replicado de una forma muchos más sencilla [RVA99]. Además, de forma análoga a ReCANcentrate, CANELy no restringe el número de buses que se pueden utilizar. Sin embargo, este proyecto se centra en el estudio de este bus cuando incluye dos replicas; ya que por motivos de coste los sistemas tolerantes a fallos normalmente no suelen incluir más de dos replicas de la red.

Para poder gestionar el tráfico replicado de una forma sencilla, cada nodo de CANELy dispone de un único controlador CAN y transmite y recibe de ambos buses en paralelo (figura 8). Para trasmitir, el nodo simplemente envía la salida de transmisión del controlador CAN a ambos buses al mismo tiempo utilizando dos transceivers, uno por cada bus. Para recibir, las señales que provienen de los buses (a través de sus transceivers correspondientes) se acoplan mediante una puerta AND, tal y como muestra la figura 8. El resultado de esta AND constituye la señal que se conecta a la entrada de recepción del controlador. Es importante notar que esta señal es análoga a la señal resultante obtenida por los hubs en CANcentrate y ReCANcentrate. En estas dos estrellas, la señal resultante se

calcula acoplando en los hubs la contribución de todos los nodos, mientras que en el bus de CANELy esta señal se obtiene acoplando la contribución de todos los buses y, por ende, también la de todos los nodos.

En lo que concierne a los mecanismos de tolerancia a fallos, cada nodo necesita asilar y descartar cualquier bus que falle. Para ello, cada nodo incluye una puerta lógica OR entre la contribución de cada bus y la puerta AND, así como una *Unidad de selección del medio*, o *Media Selection Unit* (MSU) en inglés. El papel de estas OR es análoga al de las puertas OR en los módulos de acoplamiento de los hubs de CANcentrate y ReCANcentrate; es decir, cada OR se utiliza para aislar la contribución proveniente de un bus cuando ésta se diagnostica en fallo permanente.

El papel de la MSU es similar al de las Unidades habilitadores / deshabilitadoras. La MSU monitoriza la contribución de cada bus, así como la señal que resulta de acoplar estas contribuciones en la puerta AND. Por una parte, la MSU cuenta el número de bits consecutivos de la misma polaridad y es capaz de detectar cuando una contribución está stuck-at. Por otro lado, la MSU incluye un contador de errores para la contribución de cada bus, y lo incrementa cada vez que detecta que la contribución de dicho bus difiere de la de los otros buses y corrompe la trama que se observa en la señal obtenida por la AND. Cuando la MSU detecta un que la contribución de un bus está stuck-at o que su contador de errores ha sobrepasado un cierto límite, aisla el bus de la puerta AND escribiendo un '1' en la puerta OR correspondiente. Es importante notar que el bus replicado de CANELy es anterior a la existencia de CANcentrate y ReCANcentrate

Finalmente es importante notar que el bus replicado de CANELy tiene una menor capacidad de contención de errores, tratamiento de fallos y tolerancia que ReCANcentrate. En primer lugar, la MSU no puede monitorizar la contribución de cada nodo independientemente, por lo que no puede alcanzar la misma precisión que ReCANcentrate a la hora de detectar y diagnosticar fallos de tipo stuck-at-recessive y bit-flipping. En segundo lugar, si bien en CANELy se pueden aislar y tolerar fallos que vierten errores al canal, p.e. un transceiver que envía bit-flippings, o una sección del medio que falla en modo stuck-at, un fallo en CANELy suele provocar una pérdida mayor de redundancia que en ReCANcentrate. Esto se debe a que un fallo que vierte errores al canal deja inutilizado todo un bus, mientras que en ReCANcentrate simplemente deja inutilizado un puerto. Por último, es importante destacar que los hubs, los nodos y los cables están más separados físicamente en una estrella replicada que en un bus replicado. Por ejemplo, todas las replicas de bus se aproximan físicamente las unas a las otras en cada nodo. Esto hace que la probabilidad de que un fallo, por ejemplo un choque, afecte a más de un bus es mayor que la probabilidad de que afecte a más de una estrella. Dicho de otra forma, los fallos relacionados (ver Sección **??**) son más comunes en un bus replicado que en una estrella replicada, ya que el primero presenta una menor independencia entre fallos.

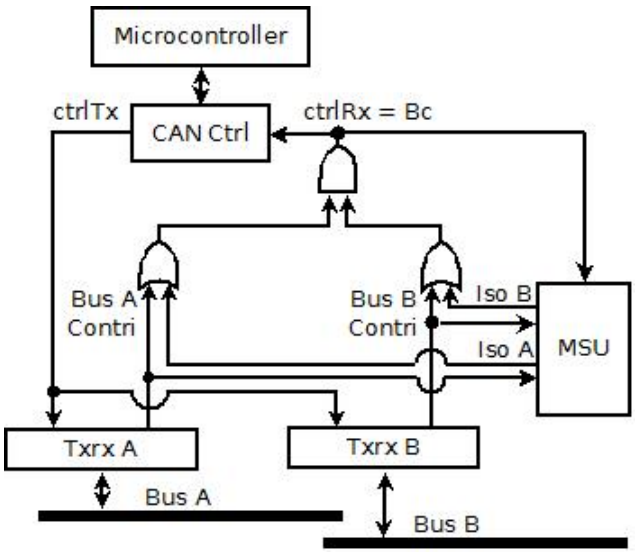

Figura 8: Aspecto de un nodo de la red CANEly

7.5. Conclusiones

En esta sección se presentan tres de las alternativas topológicas más importantes que se han propuesto para CAN hasta la fecha.

La primera de ellas, CANcentrate [Bar10], utiliza una topología de estrella simple cuyo hub es capaz de contener errores provocados por fallos en los nodos o en el medio, evitando así que los errores se propaguen a través del canal de comunicaciones. El principal inconveniente de CANcentrate, sin embargo, es que a pesar de eliminar los múltiples puntos únicos de avería (ver Sección *mecasDependability*) de un bus CAN, todavía presenta uno de estos puntos: el hub.

La segunda topología presentada es ReCANcentrate [Bar10]. Básicamente esta topología es una mejora de CANcentrate y consiste en el uso de estrellas replicadas. ReCANcentrate incluye dos o más estrellas, de tal forma que elimina todos los puntos únicos de avería. Además, los hubs de ReCANcentrate están interconectados entre sí y crean un dominio único de comunicación, transmitiendo exactamente el mismo tráfico, bit a bit, hacia los nodos. Por una parte, esta característica permite simplificar la forma en que los nodos tienen que gestionar el tráfico replicado. Por otro lado, esta característica hace posible que un nodo se pueda comunicar con el resto mientras tenga al menos una conexión correcta hacia un hub que no haya fallado. De hecho, en ReCANcentrate es posible conectar nodos no críticos a sólo uno de los hubs para, así, reducir los costes del cableado. Por último, cada nodo de ReCANcentrate dispone de un controlador CAN por cada estrella a la que está conectado, proveyendo así tolerancia a fallos de los controladores.

La última alternativa topológica que se ha descrito es el bus replicado pro-

puesto en el contexto de CANELy [**?**]. Este bus replicado es anterior a la existencia de CANcentrate y ReCANcentrate e inspiró el diseño de ambas estrellas. La principal ventaja de este bus replicado es que utiliza menos hardware adicional en comparación con las estrellas y otros buses replicados, lo cual presumiblemente tiene un impacto positivo sobre la dependability. Además, en este bus replicado la contribución de todos los buses se acopla en cada nodo, de tal forma que se consigue crear un dominio único de comunicación que facilita a los nodos la gestión del tráfico redundante. El bus replicado de CANELy elimina el punto único de avería que un bus simple representa y permite a los nodos aislarse de buses que consideran averiados. De esta forma, si el tráfico de alguno de los buses se corrompe permanentemente, los nodos se pueden seguir comunicando mediante los buses restantes.

La principal desventaja del bus replicado de CANELy es que no tolera fallos que afecten a los controladores CAN, ya que cada nodo tan sólo incluye un único controlador. Otra desventaja importante es que presenta una menor independencia entre fallos en comparación con las estrellas, lo cual a priori tiene un impacto negativo importante sobre la dependability.

Como ya se ha dicho, este proyecto se centra en modelar la fiabilidad de un sistema distribuido basado en el bus replicado de CANELy. Esto permitirá comparar dicha fiabilidad con la que se puede obtener mediante otras topologías, y arrojará luz sobre las ventajas de un bus replicado con respecto a una estrella replicada y viceversa.

8. Técnicas de modelado para evaluar cuantitativamente las garantías de funcionamiento

8.1. Introducción

Como se explicó en el diseño de sistemas tolerantes a fallos, existen dos tipos de evaluación: cualitativa y cuantitativa. En nuestro caso, y como se ha también explicado anteriormente, es necesario evaluar el sistema según la fiabilidad. Para medir la fiabilidad de un sistema es necesario una evaluación cuantitativa, que utiliza un modelo.

En la siguiente sección se explican algunos de los formalismos de modelado más comunes y se hace una pequeña clasificación de éstos. A continuación se describe con más detalle el formalismo de modelado SAN, que es el utilizado para modelar el sistema en este proyecto.

8.2. Técnicas de modelado

Existen dos tipos de formalismos de modelado para la evaluación cuantitativa: modelos sin espacio de estados y modelos con espacio de estados [MFT00]. Una definición formal sería la siguiente: el espacio de estados es un modelo ma-

temático de un sistema físico descrito mediante un conjunto de entradas, salidas y variables de estado relacionadas por ecuaciones diferenciales de primer orden que se combinan en una ecuación diferencial matricial de primer orden [MFT00].

Algunos de los modelos sin espacio de estados más importantes son: diagrama de bloques de fiabilidad y árboles de fallos. Un diagrama de bloques de fiabilidad modela las dependencias operacionales entre subsistemas que hacen posible al sistema funcionar. Esta forma de modelado tiene un modelo de fallos simple; una dependencia operacional solo puede o funcionar, o tener una avería. Además los fallos son independientes entre sí y no existe un mecanismo de reparación de los fallos. En cambio, un árbol de fallos modela las dependencias operacionales entre los subsistemas que conducen al sistema a fallar. Como el tipo de modelo anterior, también tiene el mismo modelo de fallos simple, independientes entre si y sin mecanismo de reparación.

Los modelos con espacio de estados son más potentes ya que pueden modelar fallos, mecanismos de reparación, coberturas, etc; además tiene dependencia entre esos aspectos. La evolución del estado del sistema se modela mediante un proceso estocástico. Un proceso estocástico es una familia de variables aleatorias definidas en el espacio de estados y que son indexadas por un parámetro de tiempo de tal forma que representa el estado del sistema en un instante de tiempo determinado [MFT00]. Existen 2 diferentes tipos de modelos con espacio de estados: cadenas de Markov y Redes de Petri [MFT00].

Hay una gran cantidad de modelos con cadenas de Markov, dos de los principales son: Cadenas de Markov de Tiempo Continuo (CMTC) y Modelos de Recompensa de Markov (MRM). En un modelo CMTC los estados y transiciones se utilizan para modelar como evoluciona el estado del sistema. Esta evolución no depende de la historia del sistema, si no que evoluciona según unas probabilidades definidas, además la probabilidad tampoco depende del tiempo que el sistema a estado en ese estado, el tiempo es homogéneo en todo el sistema. Los modelo MRM son extensiones de los modelo CMTC, un modelo MRM puede asociar una función de recompensa a un estado especifico del espacio de estados. Así pues, mientras el sistema esté en ese estado con función de recompensa, se acumula un valor dependiente a la función.

Los modelos con Redes de Petri son los más usados en muchas áreas, por ejemplo, en modelos econométricos. Una de sus características principales es que pueden usarse para modelar procesos estocásticos, estas Redes de Petri se denominan Redes de Petri Estocásticas y se caracterizan por tener un retraso de disparo en cada transición definida por una variable aleatoria. Esta característica nos permite modelar procesos estocásticos de una forma muy compacta y flexible. Otra de las ventajas de las Redes de Petri, es la posibilidad de construir automáticamente un modelo CMTC asociando una función exponencial a cada transición. La importancia de poder convertir la red de Petri en un modelo CMTC se basa, como se explicará en el siguiente apartado, en la facilidad de resolver un modelo CMTC mediante herramientas software. Por último, existen muchas variables de las Redes de Petri Estocásticas, cada una con algunas características adicionales. En nuestro caso, se usa las *Stochastic Activity Networks*, que nos permite modelar las transiciones mediante actividades temporales o instantáneas.

8.3. SAN

Como se ha explicado en el apartado anterior, el modelo de este proyecto usa el formalismo *Stochastic Activity Networks* (SAN), que es un extensión de las Redes de Petri [CFJ+91] [TMGT93] [StBoT04].

En lugar de utilizar SANs, se habría podido usar otros formalismos para construir el modelo. Como se verá cuando se explique el modelo detalladamente, se modela el sistema representando con diferentes estados cuando ha ocurrido un fallo y como ha afectado al sistema. Esto implica que el modelo cambia su estado (salvo algunas excepciones que se explicarán en las siguientes secciones) cuando un fallo ocurre o cuando un error se contiene o cuando el mecanismo de tolerancia a fallos actúa.

Por un lado, el TTF de un componente en concreto está distribuido exponencialmente, lo que implica que el tiempo que transcurre hasta que un fallo ocurre en el sistema también está distribuido exponencialmente. Por otro lado, las acciones de contención de errores y tolerancia a fallos están modeladas para ser realizadas instantáneamente, por lo que el sistema puede cambiar de un estado a otro inmediatamente. El hecho de que las transiciones de estado sean distribuciones exponenciales o instantáneas permite modelar el sistema como una Cadena de Markox de Tiempo Continuo (CMTC) [STP96]. Sin embargo, modelar un sistema por medio de CMTC tiene algunas dificultades. El número de estados de una CMTC tiende a aumentar cuando la complejidad del sistema del modelo incrementa. Además, una CMTC no provee demasiadas primitivas para el modelado, por esta razón, un modelo CMTC no suele ser intuitivo.

En contraste, las Redes de Petri ofrecen muchas más primitivas que las CMTC y permiten construir un modelo compacto que es mucho más sencillo de comprender. Los diferentes formalismos de las Redes de Petri que existen difieren en las facilidades que estas proveen. En este caso, SANs es uno de los formalismos de redes más potentes [AM02]. Una SAN incluye *tokens*, *places*, actividades, puertas de entrada y puertas de salida. El número de tokens localizado en cada place, es decir, el marcaje de los places, determina el estado del sistema modelado. Una actividad está conectada con una o más places fuente y tiene uno o más casos, cada uno conectado a una o más places de destino. Cada actividad puede estar habilitada y desactivada. Cuando está habilitada, dispara inmediatamente o en concordancia con una distribución estadística dada para cambiar el marcaje de los places, así modelando las transiciones del sistema mediante diferentes estados. Las actividades que se disparan inmediatamente son denominadas actividades instantáneas, mientras que las otras se denominan actividades temporizadas. Una puerta de entrada define una condición por la cual una actividad se dispara, que depende del marcaje de los places fuente. Esta puerta además especifica de que forma van a cambiar los marcajes de los places fuentes una vez la actividad se dispara. Por último, una puerta de salida está conectada a un caso en concreto de una actividad y especifica los cambios en el marcaje que se realizan dependiendo de algunas condiciones.

El formalismo de las SANs también provee de dos primitivas para construir un modelo como una composición jerárquica de submodelos: la primitiva

Join, que permite interconectar diferentes submodelos compartiendo places; y la primitiva *Rep*, que puede usarse para replicar un modelo dado con la idea de modelar diferentes instancias de un mismo submodelo.

Todos los mecanismos explicados anteriormente se usan para especificar la estructura de un proceso estocástico que modela el comportamiento de un sistema. Pero además de estos mecanismos, el formalismo de las SANs ofrece la posibilidad de especificar un modelo de recompensa. Un modelo de recompensa dado está asociado para especificar estados concretos de la estructura de estados del modelo y poder calcular una métrica especifica o atributos como la fiabilidad, la productividad, etc. Particularmente, el formalismo de las SANs ofrece la posibilidad de especificar un modelo de recompensa mediante las denominadas variables de recompensa. Una variable de recompensa está relacionada con un aspecto específico del proceso estocástico. Existen dos tipos de variables de recompensa: variables de recompensa por impulso, que son las asociadas al número de ocasiones que una actividad en concreto se dispara; y variables de recompensa de estimación, que están asociadas a un marcaje especifico.

Por último, es importante recalcar la importancia de poder asumir distribuciones exponenciales de TTF. Ya que el proceso estocástico correspondiente al modelado SAN puede ser caracterizado por medio de una CMTC que se puede resolver analíticamente. Existen diferentes herramientas software que ofrecen la posibilidad de modelar SANs y traducirlas automáticamente a su correspondiente CMTC y proveer de las facilidades para resolverla [AM02]. En particular se usa la herramienta Möbius software [StBoT04] para construir y resolver analíticamente nuestros modelos.

8.4. Conclusiones

Después de una breve explicación de los diferentes formalismos de modelado, se decide que el modelado mediante SANs es el más adecuado para un sistema de estas características debido a la facilidad de modelar procesos estocásticos. La posibilidad de de modelar el sistema según places; el fallo de los componentes mediante actividades temporizadas; y la tolerancia a fallos y contención de errores mediante actividades instantáneas hacen de las SANs el formalismo de modelado más conveniente.

Otra de las características que ayudan a elegir las SANs, es la posibilidad de transformar automáticamente el modelo SAN en un modelo CMTC, el cual se puede resolver fácilmente mediante herramientas software.

9. Herramientas de modelado

9.1. Introducción

Como se ha mencionado en la sección anterior, la herramienta elegida para modelar el sistema es la herramienta de software Möbius, que soporta múltiples

formalismos de modelado, incluido las SANs. Otra de las razones para elegir esta herramienta, es la posibilidad de transformar el modelo SAN en un modelo CMTC y resolverlo, todo en la misma herramienta. Por lo cual nos permite modelar y extraer los resultados deseados del modelo sin necesidad de usar otros software auxiliares.

Las utilidades que nos concede la herramienta Möbius son muy amplias y explicarlas todas queda fuera de la intención de este documento. Entonces, se explica únicamente de una forma breve las utilidades empleadas para diseñar el modelo. Al igual que el proceso que nos lleva de diseñar el modelo, a obtener resultados.

El proceso completo se basa en 6 partes que se han de realizar en este orden concreto: modelos atómicos, modelo compuesto, formalismo de recompensa, estudio, transformación del modelo y solución del modelo.

9.2. Modelos atómicos

Los modelos atómicos se podrían denominar como submodelos de las SANs. Möbius nos permite dividir un modelo en diferentes submodelos para poder gestionarlo de una forma más cómoda y sencilla. Aunque las utilidades para crear submodelos son numerosas, solo se utilizan 5: *places*, *activites*, *input gates*, *output gates* y *variables*.

Las variables, nos permite definir, como su nombre dice, variables que podremos asignar valores en procesos posteriores para modificar el modelo. Por ejemplo, es nuestro caso, se usa para modificar coberturas, números de nodo, etc.

Los places, por ejemplo la *cpus* en la figura 9, representan el estado del sistema modelado. Cada place contiene un número de tokens (que puede ser una variable) que representa el marcaje del place.

Las activities, *io_port_failure* en la figura 9, son las acciones del modelo y cambian el marcaje del sistema. Hay dos tipos: *instantaneous* y *timed*. Una instantaneous activity se dispara inmediatamente cuando las condiciones de disparo se cumplen, éstas condiciones pueden ser un marcaje positivo de un place destino; o que se cumple la condición de una input gate. Dentro de una instantaneous activity, se puede escoger hasta 30 diferentes casos o caminos que se configurar según una probabilidad. Una timed activity presenta las mismas actividades que una instantaneous, con la única diferencia que se dispara según una función de distribución cuando se cumplen sus condiciones de disparo.

Las input gates, *IG1* en la figura 9, permiten crear condiciones de disparo para las activities, que son más complejas que el marcaje de un solo place. En concreto, nos permite crear condiciones con múltiples marcajes y condiciones, por ejemplo, si un marcaje de un place en concreto esta a 0 y la suma de los otros mayor a 5. Además, las input gates permiten ejecutar acciones si alguna activity se dispara debido a que se ha cumplido su condición. Éstas acciones permiten cambiar el marcaje de los places como se desee.

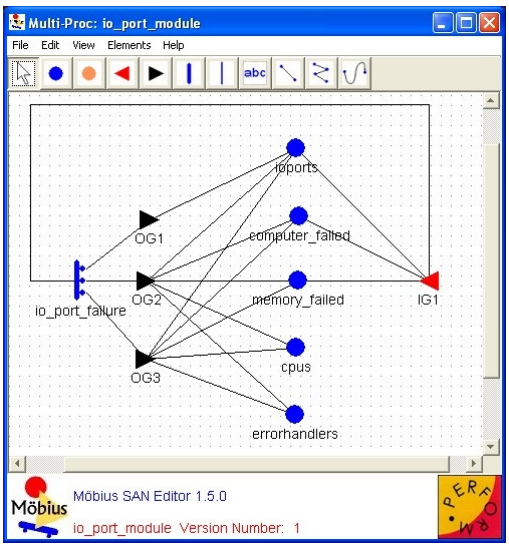

Figura 9: Ejemplo de modelo atómico SAN

Las output gates, *OG1* en la figura 9, permiten a las activities cambiar el marcaje de los places destino de una forma más compleja. Se realiza mediante la acción descrita anteriormente en las input gates y, por ejemplo, se podría sumar 2 tokens a un place en concreto y restar 5 a otro.

9.3. Modelo compuesto

El modelo compuesto tiene como función juntar todos los modelos atómicos en un único modelo SAN. Nótese que si solo existe un único modelo atómico, este paso sería innecesario. Para este proyecto solo se han utilizado dos utilidades: submodelos y joins. Un submodelo identifica a todo un modelo atómico. Un join permite unir varios submodelos y compartir places. Aunque existe una gran variedad de formas de compartir places, solo se usa el modo automático, que realiza una unión con todos los places con el mismo nombre y el mismo número de marcaje (o variable dentro del marcaje).

9.4. Formalismo de recompensa

El formalismo de recompensa nos permite definir funciones que miden información sobre el sistema modelado. Para definir la función, primero se crea una variable de recompensa para identificarlo. A continuación se define la función de recompensa, que es un retorno de una configuración de places en concreto del sistema que se quiere estudiar. Por último, se escoge un intervalo de tiempo en el cual se quiere estudiar la función de recompensa. Aunque existen muchas otras formas de realizar el estudio, éste es el usado en el proyecto.

9.5. Estudio

El estudio esta totalmente ligado a las variables definidas en los modelos atómicos. En esta parte se definen estudios, que no es mas que una configuración del modelo asignando valores a todas las variables definidas anteriormente. Es posible definir más de un estudio y permite generar más de un modelo según las asignaciones de las variables.

9.6. Transformación del modelo

La transformación del modelo simplemente transforma el modelo SAN a un modelo CMTC para poder ser resuelto. La técnica utilizada se denomina *Flat State Space Generator*. Esta transformación consume un gran número de recursos para ejecutarse.

9.7. Solución del modelo

Por último la solución del modelo, existen numerosas posibilidades para solucionar el modelo, pero solo se utiliza la denominada *transient solver*. El transient solver resuelve para instantes de tiempo determinados usando aleatorización. Calcula la media y la varianza de cada variable de recompensa en los tiempos definidos en las variables de recompensa. La aleatorización se basa en la idea de subordinar la cadena de Markov con un proceso de Poisson. Además, la computación de las probabilidades de estado en la CMTC y la computación de las probabilidades de Poisson pueden resolverse de una forma numérica estable. Por último, la media y varianza se muestran en un fichero de salida.

9.8. Conclusiones

En esta sección se muestra el proceso que se utiliza en la herramienta software Möbius desde que se genera el modelo hasta que se resuelve y obtener la información deseada. Se explica las herramientas y utilidades básicas que el software provee utilizadas en cada parte del proceso.

Parte III

Modelado

10. Métricas para cuantificar la fiabilidad

Como se ha ido mencionando a lo largo del documento, entre todos los atributos de la *dependability*, se está interesado en medir la fiabilidad. La fiabilidad se define como la habilidad del sistema de proporcionar continuamente el servicio propuesto en un intervalo de tiempo. Ésta definición implica que la fiabilidad es un atributo que está fuertemente relacionado a las características particulares del sistema ya que se define de forma diferente su servicio.

Se puede diferenciar dos tipos de sistemas. Los primeros son sistemas que solo pueden proporcionar su servicio si todos los nodos se pueden comunicar con el resto de nodos. Este sistema se define como no tolerantes a fallos. Los segundos son sistemas que pueden aceptar o tolerar el fallo o la desconexión de hasta k de los N nodos que forman el sistema. Se define como sistemas tolerantes a fallos. Ejemplos de sistemas tolerantes a fallos pueden ser un hotel que se comunica con el sistema de luz y calefacción de cada habitación, que el hecho de que alguna habitación esté desconectada no implica que no se pueda continuar con el servicio en el resto de habitaciones. Otro ejemplo sería un sistema con redundancia en los nodos, en el que si un nodo se desconecta, el sistema sigue funcionando mediante su nodo replica.

Para cuantificar la fiabilidad del bus replicado se define una métrica llamada fiabilidad de sistemas tolerantes a fallos (FT/AR). Además se añade el parámetro k y se denota como FT/AR_k. Entonces FT/AR_k se define como la probabilidad en la cual al menos $N-k$ de los N nodos del sistema pueden operar y comunicarse correctamente con el resto de nodos en un intervalo de tiempo en concreto. Con la definición anterior, se puede definir la métrica de fiabilidad de sistemas no tolerantes a fallos como una FT/AR_0 o también denominada NFT/AR. Es decir, la probabilidad de que al menos $N-0$ de los N del sistema puedan operar y comunicarse entre si, o lo que es lo mismo, todos los nodos del sistema.

Particularmente, se enfoca en las métricas FT/AR_1 y FT/AR_0. Esto es debido a que la intención del modelo es para comparar con el resto de modelos CAN, CANcentrate y ReCANcentrate. Se podría usar solamente FT/AR_0, pero el modelo de CANcentrate no esta diseñado para ser mejor en este caso, ya que la topología en estrella busca poder aislar nodos con fallos del resto del sistema. Si un solo nodo que falla nos provoca el fallo del sistema, los mecanismos implementados en CANcentrate se vuelven inútiles y además se añade hardware y software extra que aumentaría la probabilidad de surgir errores. Por esta razón se calcula también FT/AR_1 y poder extraer comparaciones para ambos casos.

Otra métrica que se usa en los resultados extraídos es el tiempo de misión. Se define como el intervalo de tiempo en que FT/AR_k no supera un porcentaje definido. Por ejemplo en nuestro caso, se define el tiempo de misión como el intervalo de tiempo de 0 hasta t en el que FT/AR_k es menor a 0,9999.

11. Limitaciones del modelo

Moderar la *dependability* es una disciplina cuya aplicación presenta algunas limitaciones que se tienen que tener en cuenta para entender la utilidad y para interpretar adecuadamente los modelos y los resultados.

Una de las mayores limitaciones de modelar la *dependability* es que es imposible reflectar todas las características reales del sistema, por eso el modelo es una abstracción del sistema real. Ésto es porque computacionalmente se necesita una gran cantidad de recursos para resolver el modelo y obtener resultados numéricos. La necesidad de más recursos incrementa cuando se intenta incluir más detalle al modelo, de forma que éste se vuelve imposible de resolver. Como consecuencia, cada modelo tiene una serie de suposiciones y por tanto es inexacto.

Aunque no hay una solución definitiva para resolver este problema, aún es posible modelar las características esenciales del sistema que nos permita extraer conclusiones sobre como diferentes alternativas de diseño y de suposiciones afecta a atributos de la *dependability* como la fiabilidad.

Otra importante limitación del modelado de la *dependability* es el hecho de que es difícil cuantificar de forma correcta algunas de las características específicas del sistema, como la proporción de los modos de fallo, la cobertura de los mecanismos de contención de errores o del failure rate. Para extraer resultados que se puedan comparar, éstos valores numéricos se extraen de las pruebas realizadas en los modelos de CAN, CANcentrate y ReCANcentrate, que a su vez fueron buscadas dentro de implementaciones reales y aspectos tecnológicos.

Además, en orden de evitar resultados que puedan favorecer a CANcentrate y ReCANcentrate, los modelos con los que se quiere comparar, las suposiciones para las características siempre han sido a favor del bus replicado. Pero además, para poder identificar que características son las que más afectan a la fiabilidad del sistema, todas éstas son parametizables en el modelo. En consecuencia, permite al modelo adaptarse fácilmente para poder estudiar como cambia la fiabilidad según sus diferentes parámetros.

Después de éstas limitaciones y sus soluciones, queda claro que la idea principal del modelo no es cuantificar exactamente como sería la fiabilidad de un sistema de bus replicado. Sino tener una idea aproximada y compararse con otros modelos de topología de CAN, concretamente, CAN, CANcentrate y ReCANcentrate.

12. Suposiciones del modelo

Algunas suposiciones del modelo presentes se basan en determinar su estructura, mientras otras se reflejan en sus parámetros. Éstos parámetros permiten hacer análisis respecto a aspectos fundamentales del sistema relacionados con

la *dependability*. Como ya se ha dicho anteriormente, en el caso de comparar el bus replicado con la estrella replicada se consideran valores por defecto razonables, pero que además se consideren favorables al bus replicado y desfavorable a los los sistemas basados en estrella CANcentrate y ReCANcentrate. Todos los parámetros del modelo del bus replicado se pueden observar en la tabla 1.

Parámetro	Valor por defecto	Significado
Bus Failure Rate	6.1573E-6	Failure rate del conjunto de componentes del bus
Bus Failure Flip Prob	0.66	Probabilidad de fallo de Bit-Flipping en el modelo de fallos del bus
Bus Failure Str Prob	0.33	Probabilidad de fallo de Stuck-at-Recesive en el modelo de fallos del bus
Ctrl Failure Rate	1.25537E-6	Failure rate del controlador CAN
Ctrl Failure Flip Prob	0.33	Probabilidad de fallo de Bit-Flipping en el modelo de fallos del controlador CAN
Ctrl Failure Str Prob	0.66	Probabilidad de fallo de Stuck-at-Recesive en el modelo de fallos del controlador CAN
Ctrl Stops	0.95	Cobertura de detección de Bit-Flipping del controlador CAN
Ctrl Autodetection Prob	0.5	Cobertura de detección de Bit-Flipping provocado por el propio controlador CAN
Txrx Failure Rate	6.7328E-7	Failure rate del transceiver
Txrx Failure Flip Prob	0.66	Probabilidad de fallo de Bit-Flipping en el modelo de fallos del transceiver
Txrx Failure Str Prob	0.33	Probabilidad de fallo de Stuck-at-Recesive en el modelo de fallos del transceiver
MicroC Failure Rate	3.2531E-6	Failure rate del Microcontrolador
MSU Failure Rate	1.225537E-6	Failure rate de los componentes de la MSU
MSU Failure Iso Prob	0.5	Probabilidad de fallo en que la MSU aisla las dos contribuciones del nodo
MSU Failure Not Iso Prob	0.5	Probabilidad de fallo en que la MSU no contiene errores de las dos contribuciones del nodo
MSU Stops	0.95	Cobertura de detección de Bit-Flipping de la MSU

Cuadro 1: Parámetros del modelo

El sistema se considera que está compuesto por los siguientes componentes: microcontrolador, controlador CAN, transceivers, MSU, memorias IC, osciladores, PCBes, segmentos de cable, conectores y terminaciones de la red. Se asume que la longitud del bus es de 100 metros. Los nodos del bus están equidistantes y están conectados siguiendo una configuración *daisy chain* el cuál minimiza el número de conectores [Bar10].

Las averías de los componentes se consideran como permanentes. Cada distribución TTF de los componentes es exponencial y no defectuoso [MTK95], con media $1/\lambda$, donde λ, es la Failure rate expresadas en horas^{-1}.

Desde que no hay un consenso real en la proporción de modo de fallos de los componentes, desde el punto de vista de un componente que falla se asume que manifiesta la misma proporción de fallos de Stuck-at-Recesive o Bit-Flipping. Pero hay dos excepciones: el microcontrolador siempre falla en Stuck-at-Recesive; y el controlador CAN, que envía Stuck-at-Recesive y Bit-Flipping con una proporción de 75 % y 25 %. En el caso de los fallos de los componentes que forman la MSU, la proporción es la misma en los dos tipos de fallos: aislar las contribuciones o no aislar las contribuciones.

Cuando el transceiver falla, tiene dos caminos por los que puede fallar: hacia fuera del nodo afectando al bus; hacia dentro del nodo afectando al propio nodo. En nuestro caso, cuando el transceiver falla lo hace hacia las dos direcciones, afectando tanto al bus como al mismo nodo.

La efectividad de los mecanismos de tolerancia de errores se caracterizaron mediante dos tipos de parámetros. El primer tipo de parámetro se denomina *Max_Nodes* y determina la habilidad del sistema para aceptar o tolerar la avería o desconexión de hasta k de los N nodos. Es decir, *Max_Nodes* representa los k nodos máximos que el sistema puede tolerar con averías. El otro tipo de parámetro son de cobertura y definen la probabilidad de que un mecanismo de contención de errores funcione. Dos de las coberturas son de 95 % para Bit-Flipping y 100 % para Stuck-at-Recesive para la MSU y el controlador CAN. Otra cobertura es la del controlador CAN a fallos propios, y es de 50 % para Bit-Flipping y 100 % para Stuck-at-Recesive.

13. Estrategia de modelado de partida

13.1. Introducción

En esta sección se explicará cuál es la estrategia de modelado de partida. Es importante recordar que esta estrategia es la misma que se ha utilizado en los modelos de CAN, CANcentrate y ReCANcentrate; con el fin de poder comparar los resultados que se pueden obtener con estos modelos y el modelo de bus replicado que se ha desarrollado en este proyecto.

13.2. Error-Containment Regions

Para poder lidiar con la complejidad del sistema, los fallos de componentes individuales no se modelan, sino que se modela el grupo de éstos. Específicamente, se divide el sistema en lo que llamamos *Error-Containment Regions* (ECRs). Una ECR es una parte del sistema que incluye diferentes componentes que pueden ser aislados para prevenir la propagación de errores. Es decir, para prevenir la propagación de errores cuando un componente falla, los mecanismos de con-

tención de errores del sistema no tienen la granularidad suficiente como para aislar únicamente dicho componente; si no que aíslan a una ECR o subconjunto de componentes como un todo. Esta limitación con la que el sistema contiene errores hace que sea irrelevante guardar el estado interno de cada componente; hace irrelevante guardar información sobre si cada componente individual ha fallado o no. Por tanto, es suficiente con guardar información sobre si cada ECR ha fallado o no, y por eso, reducimos la complejidad del sistema modelado.

Todos los modelos, tanto CAN, CANcentrate, ReCANcentrate y buses replicados, están formados por los mismos componentes o entidades: microcontrolador, controlador CAN y transceivers. Sin embargo, existen componentes que solo existen en algunas arquitecturas. Por ejemplo, el Hub en las arquitecturas de estrella, el bus en las arquitecturas de buses o la MSU en la arquitectura de bus replicado.

En el caso del bus replicado, se puede diferenciar cinco diferentes tipos de ECRs (TECR):

1. El Nodo Kernel que incluye el Microcontrolador.
2. El Controlador representado por el Controlador CAN
3. La MSU que incluye la MSU y las puertas AND y OR que usa para unir las contribuciones.
4. Las Entradas y Salidas del nodo correspondiente al transceiver.
5. El Bus que incluye los cables y los conectores de una replica en concreto.

Sin embargo, todos los modelos no agrupan los componentes en las mismas ECRs, cada modelo agrupa las ECRs de forma ligeramente distinta. Esto se debe a los componentes diferenciadores y a la propia definición de una ECR, ya que los componentes que forman una ECR dependen de la arquitectura y de cómo se aplican los mecanismos de contención de errores.

Por ejemplo, en los modelos de CAN y CANcentrate, una de las ERC se denomina *Conexión de Nodo*, la cuál comprende todas las entidades necesarias para conectar el nodo al bus o a un Hub. En el caso del bus incluye un controlador CAN y un transceiver. En CANcentrate, incluye un controlador CAN, dos transceivers, dos conexiones (una para el uplink y otra para el downlink), dos interfaces de entrada y salida con el Hub y cuatro terminaciones.

13.3. Failure rate y modos de fallo

Con la definición de las ECRs y la definición del modelo en base a ECRs, es importante definir el *Time To Failure* (TTF) de cada ECRs en base a sus componentes. Hay que recordar que se representa el tiempo que transcurre hasta que un fallo ocurre en alguna de las ECRs de cada un tipo en concreto. Entonces, la distribución del TTF se modela de la siguiente forma. Primero, suponer

que una ECR esta compuesta de E entidades, cada una de las cuales falla independientemente siguiente una distribución TTF exponencial con un fallo un failure rate λ_i, donde $i \in [1, E]$. Entonces, la distribución TTF de una región en concreto es la siguiente:

$$F_{ECR}(t) = (1 - e^{-\lambda_{ECR} \cdot t}) \tag{1}$$

donde λ_{ECR} es el failure rate de toda la ECR, y se calcula como el sumatorio de los failure rates de cada una de las entidades individuales:

$$\lambda_{ECR} = \sum_{i=1}^{E} \lambda_i \tag{2}$$

Segundo, se puede ver que todas las ECRs supervivientes representadas por su modelo de ECR son iguales una con las otras. Esto implica que cada una de ellas tienen las mismas entidades, y por eso, la misma distribución TTF. Por esta misma razón, la distribución TTF se representada por la actividad se calcula como:

$$F_{TECR}(t) = (1 - e^{-\lambda_{TECR} \cdot t}) \tag{3}$$

donde λ_{TECR} es el periodo por el cual cualquier modelo de un ECR superviviente representa el fallo. Específicamente, si *okECRs→Mark()* representa el numero de ECRs supervivientes, entonces el valor de λ_{TECR} se obtiene como:

$$\begin{aligned} \lambda_{TECR} &= okECRs \rightarrow Mark() \cdot \lambda_{ECR} \\ &= okECRs \rightarrow Mark() \cdot \sum_{i=1}^{E} \lambda_i \end{aligned} \tag{4}$$

Como que cada una de las ECRs que el modelo representa incluye las mismas entidades, es posible calcular las proporciones de los casos de las *ECRsFailure* en términos de la proporción de los modos de fallo de las entidades de una sola región. En este sentido, se considera que una ECR esta constituida por E entidades que exhiben M modos de fallo diferentes. Además, si un modo de fallo dado es denominado fm_j con $j \in [1, M]$, por la que la proporción i de una entidad exhibe el modo de fallo fm_j se denota por $fmp_{i,j}$ con $fmp_{i,j} \in [0, 1]$ $\forall i \in [1, E]$ y $\forall j \in [1, M]$. Entonces, la proporción con la cual la ECR exhibe el modo de fallo fm_j es una función que depende de los failure rates de las entidades constituyentes, λ_i, así como de las proporciones con las cuales cada una de las entidades exhibe el modo de fallo, $fmp_{i,j}$. Ésta función se denomina Fmp_j y se denota como:

$$Fmp_j(\lambda_1, ..., \lambda_E, fmp_{1,j}, ..., fmp_{E,j}) \tag{5}$$

Dada la función anterior, la probabilidad por la cual una ECR falla exhibiendo el modo de fallo fm_j en el instante de tiempo t o anterior se escribe como:

$$F_{ECR_j}(t) = F_{ECR} \cdot Fmp_j = (1 - e^{-\lambda_{ECR} \cdot t}) \cdot Fmp_j \quad \forall j \in [1, M] \tag{6}$$

Además, ya que la ECR no puede exhibir más de un modo de fallo simultáneamente, el TTF de una ECR se expresa en términos de F_{ECR_j} como:

$$\begin{aligned} F_{ECR}(t) &= \sum_{j=1}^{M} F_{ECR_j}(t) = \sum_{j=1}^{M} [(1 - e^{-\lambda_{ECR} \cdot t}) \cdot Fmp_j] \\ &= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \sum_{j=1}^{M} Fmp_j \end{aligned} \tag{7}$$

Teniendo en cuenta la ecuación 1 y la ecuación 7, se corrobora que:

$$\sum_{j=1}^{M} Fmp_j = 1 \tag{8}$$

Después de las expresiones anteriores, se puede expandir la ecuación 1 de una forma que incluya los failure rate y las proporciones de los modos de fallo de las entidades (λ_i y $fmp_{i,j}$) y, entonces, comparar las expresión resultante con la ecuación 7 para obtener la expresión Fmp_j en términos de λ_i y $fmp_{i,j}$. Para poder expandir la ecuación 1, se usa una expresión que especifica λ_{ECR} en términos de las failure rates de las entidades y las proporciones de los modos de fallo.

$$\lambda_{ECR} = \sum_{i=1}^{E} \lambda_i = \sum_{i=1}^{E} \left(\lambda_i \cdot \sum_{j=1}^{M} fmp_{i,j}\right) \tag{9}$$

Más específicamente, la ecuación 1 se puede expandir de la siguiente forma:

$$
\begin{aligned}
F_{ECR}(t) &= (1 - e^{-\lambda_{ECR} \cdot t}) \\
&= (1 - e^{-\lambda_{region} \cdot t}) \cdot 1 \\
&= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \frac{\lambda_{ECR}}{\lambda_{ECR}} \\
&= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \frac{\sum_{i=1}^{E} (\lambda_i \cdot \sum_{j=1}^{M} fmp_{i,j})}{\lambda_{ECR}} \\
&= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \frac{\sum_{\substack{i=1..E \\ j=1..M}} \lambda_i \cdot fmp_{i,j}}{\lambda_{ECR}} \\
&= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \sum_{\substack{i=1..E \\ j=1..M}} \frac{\lambda_i \cdot fmp_{i,j}}{\lambda_{ECR}} \\
&= (1 - e^{-\lambda_{ECR} \cdot t}) \cdot \sum_{j=1}^{M} \Big(\frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\lambda_{ECR}} \Big)
\end{aligned}
\tag{10}
$$

Después de ésta expresión, si se compara con la ecuación 7, se deduce que Fmp_j se escribe como:

$$
Fmp_j = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\lambda_{ECR}} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\sum_{i=1}^{E} \lambda_i} \quad \forall j \in [1, M] \tag{11}
$$

La expresión Fmp_j indica que la proporción por la cual una ECR superviviente exhibe un modo de fallo es la media aritmética de las proporciones por las cuales cada entidad de la ECR manifiesta ese modo de fallo. Por ejemplo, en el caso de una ECR del Controlador del bus replicado, la expresión de un modo de fallo es la siguiente:

$$
Fmp_{flip} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,flip} \cdot (1 - flipCov_i)}{\sum_{i=1}^{E} \lambda_i} \tag{12}
$$

donde Fmp_{flip} es la proporción por la cual el controlador exhibe un fallo Bit-Flipping; $fmp_{i,flip}$ es la proporción por el cual la entidad i manifiesta un Bit-Flipping; y $flipCov_i$ es la cobertura por el cual el controlador CAN del ECR Controlador detecta y aísla la entidad i cuando se manifiesta un Bit-Flipping.

13.4. Estructura del modelo

La estructura del modelo está diseñada como una composición jerárquica de diferentes submodelos SANs, que comparten places específicas entre ellas y se unen con la primitiva Join, tal como se explicó en la sección 9.3. Se clasifican los diferentes submodelos del sistema modelado en 3 grupos dependiendo de su

función en el modelo: SANs que representan la ocurrencia de fallos en diferentes Tipos de ECR (TECR); SANs que modelan las acciones que el sistema lleva a cabo para contener errores y tolerar fallos, es decir, SANs que representan lo que hemos llamado como proceso de cobertura; y SANs que una vez se haya terminado el proceso de cobertura, evalúa si el sistema puede seguir proveyendo el servicio, o por el contrario, falla.

Tipos de Error-Containment Regions

En el primero grupo de modelos SANs, se modela la ocurrencia de fallos en diferentes TECR. Hay que notar que no se modela cada ECR, sino cada Tipo de ECR diferente. Esto nos permite reducir aún más el número de estados de la CMTC generada a partir del modelo SAN conjunto. Concretamente, cada SAN que representa un tipo de ECR guarda la información de cuantas ECRs de dicho tipo siguen funcionando correctamente y todavía no han fallado.

Se puede modelar el sistema en base a TECRs ya que no hace falta distinguir entre las ECRs de un mismo tipo. Esto es debido a que cada ECR de un mismo tipo está formado por los mismos componentes; y porque no se intenta modelar el estados de todos los ECRs. Sino que parte o partes del sistema se ven afectadas por el fallo de este tipo de ECR. Más en concreto, hay que determinar el lugar del sistema en primera instancia donde se observan los errores generados por el ECR. A partir de esta situación, en cuanto un ECR falla, se decide cuál es el lugar del sistema desde cual empieza a evaluarse el proceso de cobertura, que se modela en el siguiente grupo de submodelos.

Proceso de cobertura

Una vez ha fallado un TECRs de una forma en concreto, el sistema tiene que llevar a cabo las acciones para contener los errores y tolerar los fallos. Para decidir cual es el punto de partida de las acciones hay que elegir en que ECR en concreto ha ocurrido el fallo, y se hace estadísticamente. Por ejemplo, en buses replicados, si falla un ECR de tipo Controlador, hay que decidir en que nodo del sistema ha fallado y como es su estado de este nodo. En este caso, las posibilidades son que los dos transceivers estén operativos, o que solo esté operativo el transceiver del bus A, o solo esté operativo el transceiver del bus B, o en cambio no funcione ningún transceiver. Esta posibilidad de afectar a cada tipo de nodo se decide estadísticamente, por ejemplo, en un sistema de 4 nodos con dos nodos con los dos transceivers operativos, la probabilidad que afecte a un nodo de este tipo es 2/4.

Dependiendo del tipo de nodo que haya afectado con las características descritas anteriormente, el nodo variará su estado de una forma u otra, transmitiendo nuevos errores o no por los buses. Dependiendo de si se transmite errores nuevos en los buses y que tipo de modo de fallo está transmitiendo, habrá que seguir mirando la propagación de estos errores por el bus. Por lo tanto, este tipo de SANs pueden llamar a otra SANs del mismo tipo si alguno de los buses se

ve corrompido por un nuevo modelo de fallos.

Aunque es cierto que no hace falta modelar cada ECR de forma individual, se necesita más información sobre cada nodo de la que se ha mencionado anteriormente. Tal y como se explicará más detalladamente en la siguiente sección, nos referiremos a esta información como el *Node Operational State* (NOS) o *Estado Operacional del Nodo.*

Evaluación

Una vez que el proceso de cobertura ha finalizado, las SANs que determinan si el sistema puede proveer su servicio o no se activan, evalúan la información sobre la capacidad de operación y comunicación de los elementos del sistema, y deciden si el sistema como un todo falla o no.

13.5. Conclusiones

En esta sección se explica cuál es la estrategia de modelado de partida para este proyecto, que se corresponde con la estrategia utilizada para modelar la fiabilidad del bus CAN, CANcentrate y ReCANcentrate. En esta estrategia se abordan los principales obstáculos que se presentan al modelar estos sistemas mediante SANs.

Entre los mecanismos propuestos para abordarlos cabe destacar la definición y utilización de las Error-Containment Regions (ECRs), las cuáles permiten reducir significativamente el número de places necesarios para identificar el estado de cada una de las partes del sistema. En este sentido, una parte importante de esta capítulo está dedicada a explicar cómo se definen las propiedades de las ECRs.

Por último, también se muestra como según esta estrategia el modelo global se descompone en tres tipos de submodelos, según su funcionalidad: (1) aquellos que modelan la ocurrencia de fallos, (2) aquellos que modelan el proceso de cobertura que tiene lugar en el sistema para contener los errores y tolerar los fallos, y (3) aquellos modelos que evalúan si el sistema puede continuar ofreciendo su servicio una vez terminado el proceso de cobertura.

14. Estrategia de modelado del bus replicado

14.1. Introducción

Como se ha explicado en la sección anterior, la estrategia de modelado del bus replicado es la misma que la estrategia de los modelos de CAN, CANcentrate y ReCancentrate. Esto es debido a que el objetivo de estos modelos es poder extraer información que se pueda comparar entre ellos.

En este sentido, una de las primeras decisiones que se han tomado para modelar el bus replicado consiste en agrupar de los diferentes componentes del sistema en Error-Containment Regions (ECRs). Además, como en los otros modelos, se ha dividido el conjunto de SANs que constituyen el modelo en tres tipos: SANs que modelan la ocurrencia de fallos en diferentes tipos de ECRs, y que guardan la información sobre el número de ECRs de cada tipo que siguen en funcionamiento; SANs que evalúan el proceso de cobertura cuando una ECR ha fallado; y por último, unas SANs que, una vez el proceso de cobertura ha terminado, determinan si el sistema puede seguir proveyendo su servicio, o por el contrario, el sistema falla.

Sin embargo, se han tenido que resolver varios problemas a la hora de aplicar dicha estrategia en el caso del bus replicado. En primer lugar, se ha tenido que decidir cuáles son las ECRs que constituyen un sistema basado en el bus replicado CANely. Como se explicará más adelante, la división de este sistema en ECRs es más complicado que en los sistemas basado en una topología de estrella. Esto se debe a que los mecanismos de contención de errores y tolerancia a fallos se encuentran más distribuidos en el caso del bus replicado, mientras que en las estrellas éstos mecanismos tienden a estar centralizados. En segundo lugar, ha habido que identificar cuál es la información adicional que el modelo debe incluir sobre la capacidad de operar y comunicar de ciertas partes del sistema. De nuevo, esta tarea ha sido más complicada que en los modelos anteriores debido a que los mecanismos de contención de errores y tolerancia a fallos se encuentran más distribuidos en el bus replicado y se necesita más información para identificar la información adicional necesaria.

En ésta sección se explicarán éstos y otros detalles relacionados con el modelado del bus replicado.

14.2. Tipos de ECRs

Como ya se ha mencionado en la definición de la estrategia de modelado, se especifica un grupo de componentes llamados Error-Containment Regions (ECRs) para reducir la complejidad del sistema. Un ECR es una parte del sistema que incluye diferentes componentes y que pueden ser aislados para prevenir la propagación de errores. En el caso del bus replicado, la identificación de los diferentes tipos ECRs es bastante más complicada que los modelos que siguen una topología en estrella, ya que la contención de errores y la tolerancia a fallos es distribuida a lo largo del sistema y están definidas principalmente en el interior del nodo.

En el caso del bus replicado, se puede distinguir cinco tipos diferentes de ECRs: el Nodo Kernel que incluye el microcontrolador; el Controlador que representa el controlador CAN; el Modulo de Selección de la Unidad (MSU) que incluye el Modelo de Selección de la Unidad (MSU) y las diferentes puertas lógicas AND y OR; la IO del nodo que corresponde a un transceiver; y el Bus que incluyo los cables y conectares de un bus replica concreto. Los diferentes tipos de ECRs se pueden observar en el sistema de buses replicado de 2 buses y nodos de la figura 10.

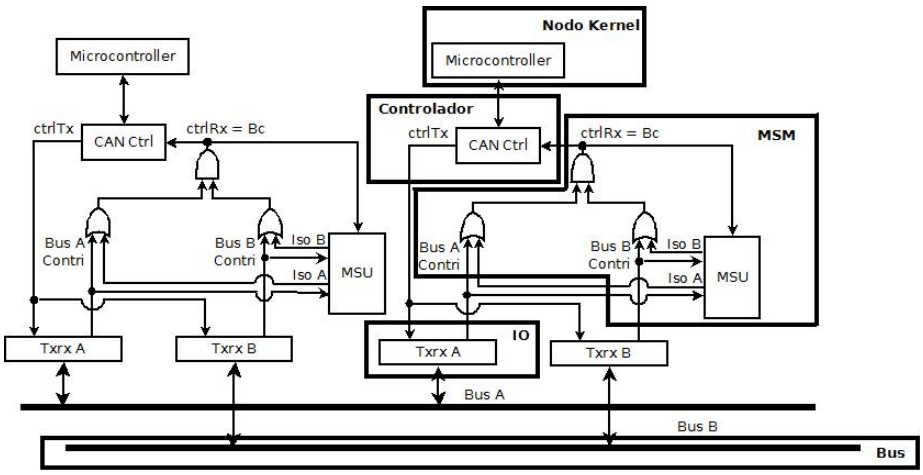

Figura 10: Identificación de tipos de ECR en un sistema de buses replicado con 2 nodos

La elección del grupo de componentes que forma cada ECR depende de la contención de errores y la tolerancia a fallos del sistema de buses replicado que está distribuido por todo el sistema. Así pues, el tipo de ECR del Nodo Kernel que engloba el microcontrolador y todos los componentes que conlleva es un tipo de ECR ya que si falla, propaga los errores hacia el controlador CAN que corta la comunicación al no recibir ordenes correctas.

El siguiente tipo de ECR es el Controlador que contiene el controlador CAN. Es un tipo de ECR ya que el controlador CAN contiene mecanismos para detectar errores que provienen de los transceivers y pasan por la MSU; y errores que el mismo provoca. Una vez ha detectado estos errores, el controlador puede aislarse de la red y no propagar los errores que recibe o que él mismo está provocando.

Otro tipo de ECR es el MSU que contiene la MSU y las puertas lógicas AND y OR. Es un tipo de ECR ya que su función es detectar errores provenientes del transceiver y aislar su contribución hacia el controlador.

La IO del nodo contiene un transceiver, y se puede ver que cada nodo tiene dos ECRs de este tipo. Este tipo de ECR es un poco más confuso, ya que de por si no tiene ningún mecanismo de detección de errores. Sin embargo, un modo de fallo del transceiver (Stuck-At-Recesive) permite tolerar fallos provenientes del bus o del controlador CAN aislando cualquier error hacia el exterior del nodo y hacia el interior.

Por último, el Bus es otro tipo de ECR y contiene como su nombre indica, el bus CAN, con sus terminaciones y sus conectores. También es un tipo de ECR por la misma razón que la IO del nodo. Aunque no tenga mecanismos de detección de errores ya que es un simple bus, uno de sus modos de fallo (Stuck-At-Recesive) permite parar la propagación de errores que los nodos transmiten por este bus, y por consecuente, al resto de nodos conectados a este mismo bus.

14.3. Modelado de las capacidades de operación y comunicación

Como ya se ha mencionado anteriormente, la elección de modelar el sistema mediante ECRs tiene el propósito de reducir el espacio de estados del CMTC que se crea con la transformación de la SAN global del sistema modelado. Sin embargo, con el modelo que contiene únicamente los places para las ECRs sólo se tiene el número de ECRs de cada tipo que aún están operativas. Con esta información del sistema, cuando una ECR falla, no se puede localizar en que parte concreta del sistema se produce ese tipo de fallo y mucho menos evaluar de forma apropiada con el proceso de cobertura si el error producido por ese fallo se propaga o se contiene. Por ejemplo, si se produce un fallo de Bit-Flipping en un transceiver, el proceso de cobertura es diferente dependiendo del estado del nodo en el cuál se ha producido el fallo. Si ese nodo con un nuevo fallo ocurre en un nodo con la MSU que ha fallado aislando todas las contribuciones, su proceso de cobertura será muy distinto que si el fallo del transceiver ocurre en un nodo que tiene la MSU operativa.

Además, es importante identificar como se realiza el proceso de cobertura porque el estado del sistema y de sus componentes puede cambiar dependiendo de como se realiza. Si se recoge el ejemplo anterior, que un transceiver falle en un nodo con la MSU que falla aislando; el nodo propagará ese error por el bus al que está conectado si éste está operativo. En cambio, si la MSU está funcionando sin ningún fallo, pueden ocurrir dos casos: primero, la MSU detecta el error y corta la contribución de ese transceiver, por tanto, el sistema cambiará igual que si la MSU del nodo ya haya fallado aislando. El segundo caso, que la MSU no detecte el error, hará que el error se propague al controlador CAN, el proceso de cobertura en el caso de que el error llegue al controlador CAN es bastante amplio, pero las diferentes consecuencias que conlleva pueden ser desde que el nodo falle completamente, hasta incluso que todo el sistema falle por consecuencia de ese error. Por lo tanto, las posibilidades en las que el sistema cambia son bastante amplias para un mismo modo de fallo en un ECR.

Para poder modelar los aspectos comentados en el párrafo anterior, se incluyen una serie de places auxiliares que permiten representar cuáles son los diferentes estados en los que los nodos y los buses se encuentran desde el punto de vista de su capacidad de operación y comunicación. El marcaje (marking) de éstos places auxiliares representa el número de nodos o buses que se encuentran en ese estado en concreto. En el caso de los buses es sencillo, los places auxiliares son cuatro, dos para cada bus que identifican si el bus está en un modo de fallo o en el otro. En cambio, para los nodos es un poco más complicado. El nombre que se le da a los estados auxiliares de los nodos ya se ha mencionado en la sección 13; es el de Node Operational States (NOS). En principio, para caracterizar el estado operacional de cada nodo, es necesario guardar la información sobre el estado de todos los ERCs de cada uno de ellos. Pero ésta solución conlleva una explosión del espacio de estados del modelo cuando el número de nodos aumenta.

Para mitigar el problema de la explosión de estados se usa una estrategia similar a la de ReCANcentrate [BPA]. En esta estrategia cada nodo y su estado

no se modelan explícitamente, sino implícitamente. Para esto, el modelo incluye un place para cada posible NOS y su marcaje representa el número de nodos que están en ese estado operacional en concreto.

La información necesaria para identificar el funcionamiento del nodo es que transmite a los dos buses. Sin embargo, es necesario clasificar de donde proviene esa transmisión, ya que un fallo en un ECR del nodo puede provocar la transmisión del error y cortar la transmisión correcta, o incluso, de otro error que se propagaba con anterioridad. Para esto, se define unos niveles de transmisión, en el que la transmisión del nivel más alto bloque una transmisión de nivel más bajo. La clasificación es la siguiente:

Nivel 0 (Ok): Transmite información correcta.

Nivel 1 (Ind): Transmite un error provocado por el Nodo Kernel.

Nivel 2 (Obl): Transmite un error provocado por un mecanismo de tolerancia a fallos del Controlador.

Nivel 3 (Ctrl): Transmite un error provocado por el Controlador.

Nivel 4 (Txrx): Transmite un error provocado por el IO del Nodo.

Sin embargo, la utilización de esta estrategia en los buses replicados conlleva problemas. Ya que los ECRs están distribuidos y la mayoría se encuentran en el nodo, la cantidad de diferentes NOS es mucho más grande. Específicamente, se generó un árbol que incluye todas las combinaciones posibles de los ECRs, con el cual se obtuvo un número enorme de NOSs diferentes. Afortunadamente, existen ramas que conducen al mismo NOS, por lo tanto el número de NOS se reduce significantemente.

Aún así, con solo la información que transmiten los nodos y su nivel no es suficiente. Para llevar a cabo todos los procesos de cobertura es necesario tener información de si el nodo puede recibir por cualquier de los dos buses, y también necesita el estado de la MSU si está ha fallado. Por suerte, la primera información viene ya implícita. Si el transceiver transmite un error, que al ser de máximo nivel, siempre se sabrá, también implica que el error se propaga hacia dentro, y por tanto no puede recibir nada. El caso de la MSU es diferente, es necesario tener información del estado de la MSU dentro del nodo, por lo tanto se añade si la MSU ha fallado y de que forma a los NOS. Pero la MSU al provocar propagación de errores de Nivel 2, no es necesario que el estado de la MSU aparezca en todos los NOS extraídos del árbol. Esta información extra es solo necesaria en los NOS que transmitan por alguno de los dos buses errores de Nivel 0 o 1. Ya que no es necesario tener la información de la MSU si está no puede provocar un cambio en el NOS.

Un ejemplo de un NOS sería *Node_OkA_TxrxStrB_NotIsoMSU*. Este NOS representa un nodo que su transceiver del bus B ha fallado con Stuck-At-Recesive, y por tanto transmite ese error por el bus B; sigue funcionando correctamente por el bus A; y los mecanismos de detección de errores y tolerancia a fallos de la MSU han fallado. Gracias a esta estrategia se ha podido reducir el número de NOS a 53.

Una vez definidos todos los places auxiliares necesarios, el proceso de cobertura se puede aplicar ya que se tiene la información necesaria de todo el sistema. Así pues, cuando un tipo de ECR falla, se decide estadísticamente (dividiendo casos posibles por totales) cuál es el NOS del nodo o el estado del bus afectado. Una vez ya se ha sabe en que tipo de nodo o bus ha afectado el fallo, el proceso de cobertura modifica el marking de los places auxiliares necesarios. Finalmente, teniendo en cuenta cómo ha quedado el marking de los places auxiliares, se evalúa si el sistema ha fallado o no.

14.4. Proceso de cobertura

Como ya se ha mencionado a lo largo de la sección, el proceso de cobertura comienza cuando un tipo de ECR falla, en ese momento se realiza un proceso que se puede resumir en la figura 11. Cuando un fallo ocurre, el tipo de ECR activa una SAN del proceso de cobertura que determina que NOS son afectados por el fallo. Específicamente, si el fallo ocurre en una replica del bus, entonces el proceso de cobertura necesita determinar cuales son los NOSs afectados por el error que el fallo genera. Por ejemplo, si falla un bus, los NOSs afectados son aquellos que todavía consideran los datos del bus fallado como válido. En cambio, si el fallo ocurre en una ECR de un nodo, entonces una SAN del proceso de cobertura determina cuáles son los NOSs candidatos para incluir esa ECR. Básicamente, la probabilidad de escoger un NOS en concreto es calculada dividiendo el número de nodos en ese NOS por el número de nodos en los NOSs candidatos a incluir ese ECR con fallos.

Una vez que se ha determinado que NOSs son los afectados, evalúa si cada nodo que están en ese tipo de NOSs aisla el error. Esta decisión se basa en la cobertura de los mecanismos de contención de errores que el nodo correspondiente a la NOS puede usar. Como resultado de la evaluación, el proceso de cobertura cambia el NOS de cada nodo. Entonces, si el nodo cambia a un NOS que representa una situación en la cual se contiene el error, el proceso de cobertura finaliza. En cambio, si el nodo cambia a un NOS en el cual el error aún se propaga, el proceso de cobertura continúa. Un nodo que no contiene el error puede contaminar todos los buses replica por los cuáles es capaz de transmitir. Por esta razón, el proceso de cobertura evalúa primero si este caso se ha dado y no existe un bus operativo para comunicar, si se llegase a esa situación, el proceso de cobertura acaba ya que ningún nodo podría comunicarse. Sino, el proceso de cobertura evalúa que modo de fallo presenta el bus contaminado y el continúa desde el principio.

Una vez definido el esquema básico, se puede observar el esquema de SANs del modelo en la figura 12. Las SANs encargadas del proceso de cobertura son las que tienen el fondo blanco, las grises son las SANs de evaluación y la SAN con el fondo negro es la SAN de fallos en los tipos de ECR. Como se puede observar, el proceso de cobertura es el que más SANs ocupa y además el más complejo. Para poner un ejemplo del recorrido de las SANs en el proceso de cobertura del modelo, supongamos un fallo de Bit-Flipping en un ECR de tipo Controlador. En este caso se ejecuta la SAN denominada *Ctrl_Failure_BitFlip* evalúa en que NOS se encuentra el nodo afectado y recrea el proceso de cober-

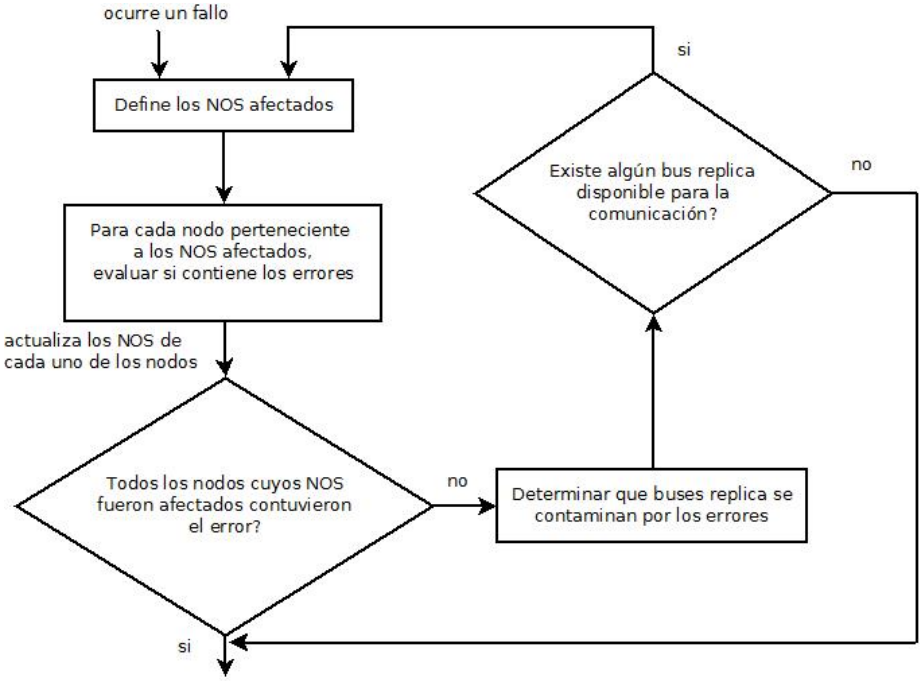

Figura 11: Esquema del proceso de cobertura

tura. A partir de este momento se pueden ejecutar hasta tres diferentes SANs según el NOS afectado. Sea el caso que sea, siempre llamará inmediatamente a la SAN *Evaluation* que se encarga de la evaluación del estado del sistema. Después, dependiendo de si el fallo del Controlador se propaga o no por los buses, puede llamar a *BusA_Failure_Str* y a *BusB_Failure_BitFlip* dependiendo del bus a que haya afectado. En el caso de que afecte a los dos buses, la evaluación ya habrá diagnosticado un fallo global del sistema y parará el proceso de cobertura. Las SANs del proceso de cobertura de los buses cambiarán los NOSs necesarios y llamarán a la evaluación de la SAN *Bus_Prop* que a su vez llama a la SAN *Evaluation*. Si se detecta el fallo global del sistema en algún momento, la SAN *Evaluation* llama a *Simlification*, una SAN encargada de reducir el número de estados para una transformación al CMTC más rápida.

14.5. Evaluación del servicio del sistema

Después de cada paso del proceso de cobertura, el grupo de SANs que evalúan el sistema se vuelven activas. Éstas SANs deciden si todo el sistema es operacional o falla dependiendo de la capacidad de los nodos para operar y comunicarse entre ellos. Esto depende en los NOS en que cada nodo se encuentra.

El número de nodos que deben poder operar y comunicarse entre ellos en orden de considerar si el sistema está operacional es parametizable, así pues el modelo se puede usar para calcular diferentes grados de fiabilidad. Cuando el sistema falla, se marca con un token un place denominado *System_Failure*. Este

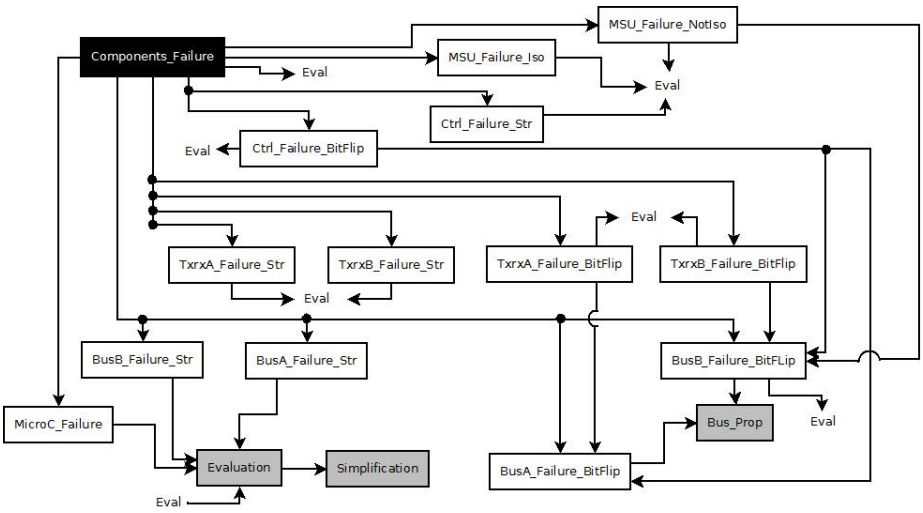

Figura 12: Proceso de cobertura del modelo de buses replicado

place es compartido por todo el sistema para que pare todas sus acciones cuando se llega a este estado. Además, se activa la SAN *Simplification*, que pone todos los estados menos el de *System_Failure* a 0. La razón de realizar esta acción es para simplificar el número de estados del modelo, y por tanto, simplificar el espacio de estados del CMTC. Además, *System_Failure* es el place que se usa para cuantificar la fiabilidad del sistema.

14.6. Conclusiones

La estrategia de modelado en el bus replicado se muestra mucho más complicada de aplicar que en los demás modelos (CAN, CANcentrate y ReCANcentrate). Esto es debido a la naturaleza de la localización de los mecanismos de detección de errores y tolerancia a fallos, que están más distribuidos que en los otros sistemas. Ésta característica implica definir unas Error-Containment Regions (ECR) algo diferentes a la de los otros modelos y mucho más localizadas dentro del nodo.

El hecho de que las ECRs estén localizados dentro del nodo hace que la información auxiliar para el proceso de cobertura sea bastante más compleja de definir. Por esto, se define el concepto de *Node Operational State* (NOS), que no es más que el estado operacional en el que se puede encontrar un nodo cualquiera. Mediante el modelado del número de nodos que se encuentran en un determinado NOS, se puede representar todos los posibles estados del sistema.

Para definir todos los NOSs que son necesarios se definen diferentes niveles de transmisión y se crea un árbol con todas las posibilidades con el que el nodo puede fallar. Una vez creado el árbol, se poda con los NOS que son equivalentes y se añade la información adicional de la MSU en los NOS necesarios.

Debido al gran número de NOS, el proceso de cobertura también es bastante

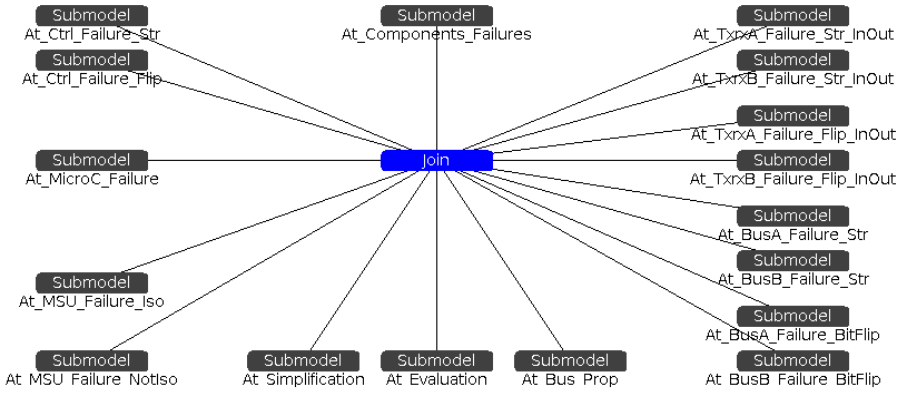

Figura 13: Modelo compuesto del sistema de bus replicado

complejo, y contiene un gran número de SANs que se comunican entre ellas. Se muestra el modelo de conexión y un pequeño ejemplo para ver la complejidad del proceso desde que falla un componente, hasta que termina el proceso de cobertura. Por último, en cada paso del proceso de cobertura se analiza el sistema para ver si todavía puede operar, o por el contrario falla.

15. Descripción de las SANs

15.1. Esquema de las SANs

En esta sección se describe el modelo de la SAN global y un ejemplo de cómo funciona el modelo desde que se produce un error en un tipo de ECR; pasando por el proceso de cobertura; y por último evaluando si el sistema sigue funcionando correctamente o ha fallado. El por qué sólo explicar un caso en concreto es debido a que el modelo es bastante grande, como se verá con el ejemplo, y además la estrategia a seguir es similar en todos los submodelos. Por eso una pequeña descripción del modelo y luego un ejemplo en profundidad que abarque todas las diferentes estrategias que se sigue en el modelo hará comprender mejor cómo funciona el modelo.

En la figura 13 se puede ver el modelo del sistema de bus replicado, compuesto por submodelos. Como ya se ha mencionado, se clasifican en tres tipos, y cada uno contiene los siguientes submodelos de SAN:

- **Tipos de ECR:**
  - *At_Components_Failures*
- **Proceso de cobertura:**
  - *At_MicroC_Failure*

- *At_Ctrl_Failure_Str*
- *At_Ctrl_Failure_Flip*
- *At_MSU_Failure_Iso*
- *At_MSU_Failure_NotIso*
- *At_TxrxA_Failure_Str_InOut*
- *At_TxrxA_Failure_Flip_InOut*
- *At_TxrxB_Failure_Str_InOut*
- *At_TxrxB_Failure_Flip_InOut*
- *At_BusA_Failure_Str*
- *At_BusA_Failure_Flip*
- *At_BusB_Failure_Str*
- *At_BusB_Failure_Flip*

- **Evaluación:**
  - *At_Simplification*
  - *At_Evaluation*
  - *At_Bus_Prop*

La SAN de tipos de ECR contiene los places con el marcaje del número de tipos de ECR que todavía funcionan, así como los failure rates de las ECR y la probabilidad del modo de fallo de cada tipo. Una vez que un componente falla en alguno de sus modos, se activa una de las SAN del proceso de cobertura.

Existen 13 diferentes SANs para realizar el proceso de cobertura y, como se puede ver por el nombre, se tiene una SAN diferente por cada modo de fallo de cada tipo de ECR existente. Así pues, cada vez que se produce un fallo en la SAN de tipos de ECR, ésta activa una de las SANs dependiendo de en que tipo de ECR se ha producido el fallo y con que modo de fallo. La SAN que se activa, al finalizar su proceso de cobertura, activa las SANs de evaluación y, si algún bus se ve corrompido por algún error que no se ha contenido, se activa también otra SAN (alguna de las dedicadas al bus).

Por último, cada vez que acaba una SAN dedicada al proceso de cobertura, se activan SANs de evaluación que comprueban el estado del sistema. Siempre se activa la SAN *At_Evaluation* que evalúa cuantos nodos todavía se pueden comunicar. Si el marcaje de algún place auxiliar de los buses cambia, también se activa la SAN *At_Bus_Prop*, que evalúa si alguno de los dos buses funciona. Por último, si el sistema no puede seguir con su servicio, se cambia el marcaje del place *System_Failure* a uno y ésta acción activa la última de las SANs de evaluación. La SAN *At_Simplification* tiene como función reducir el espacio de estados del modelo. Esto se hace poniendo a cero todos los estados menos el de *System_Failure*. Se puede realizar esta simplificación ya que la información que se quiere extraer del modelo es la fiabilidad y sólo se necesita el marcaje del place *System_Failure*.

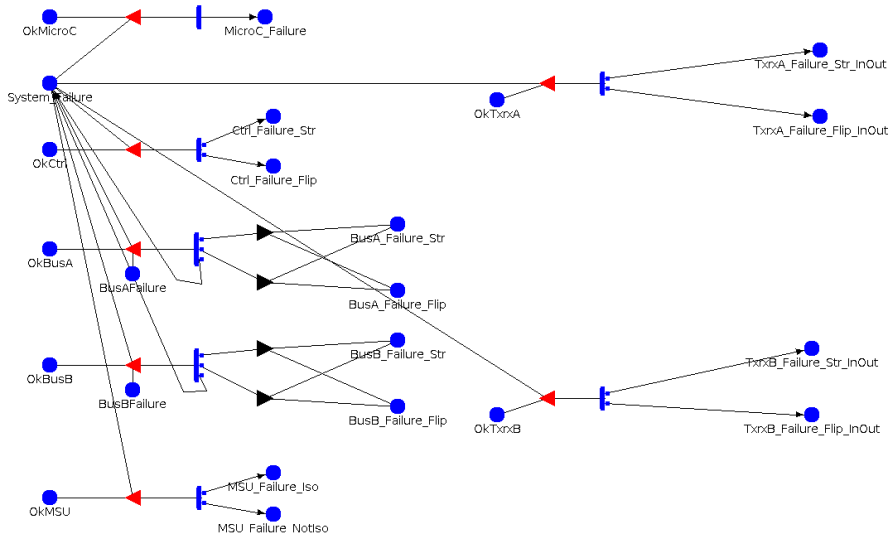

Figura 14: SAN *At_Components_Failures*

Aunque se ha mencionado que las SANs de evaluación son las que deciden si el sistema funciona o no correctamente, ésto no es del todo cierto. Todos los tipos de SANs pueden evaluar que el sistema falla y activar el marcaje del place de *System_Failure*, pero solo en un caso en concreto. Se debe a que para evitar places adicionales, alguna de las acciones formarían NOSs no definidos, ya que ese estado del nodo hace que el sistema falle ya que corrompe los dos buses del sistema. Por lo tanto, en vez de definir una NOS para este tipo de nodos, directamente se indica que el sistema ha fallado.

15.2. Ejemplo de un caso de las SANs

Como se ha mencionado en el apartado anterior, se intenta explicar la estrategia del modelo con un ejemplo de un caso de fallo de un tipo de ECR que recorre las diferentes estrategias usadas en el modelo y se pueda comprender la complejidad del modelo. Para esto se hará el recorrido por el modelo cuando un transceiver conectado al bus A falla con Bit-Flipping.

Al principio se produce un fallo en la SAN *At_Components_Failures*, figura 14. En ese instante el marcaje del place *System_Failure* es cero y el marcaje de *OkTxrxA* es mayor a 0. Ésta situación hace que la input gate esté activa para la activity con la condición:

$$System\_Failure \rightarrow Mark() == 0 \;\&\&\; OkTxrx \rightarrow Mark() > 0$$

y que cuando se active la activity conectada a ésta se realizará la función del input gate que reduce en uno el marcaje de *OkTxrxA*:

$$OkTxrxA \rightarrow Mark() - = 1;$$

La activity que habilita la input gate es temporizada y su función de distribución es la siguiente:

$$(1{,}0 * Txrx\_FailureRate) * (1{,}0 * OkTxrxA \rightarrow Mark())$$

y es el failure rate, que es una variable, de los transceivers por el número de transceivers conectados al bus A todavía en funcionamiento. El multiplicar por 1,0 se debe a una particularidad con los tipos del software Möbius, la idea es transformar todo en *floats* para que no se produzcan problemas de compatibilidad de tipos, que ocurren con bastante frecuencia en éste software. A continuación se entra en la probabilidad del modo de fallo en la activity, el camino elegido es con un fallo de Bit-Fliping con la variable de probabilidad *Txrx_Failure_Flip_Prob*. Este camino de la activity añade un token al place *TxrxA_Failure_BitFlip_InOut*, significa que un transceiver conectado al bus A ha fallado con Bit-Flipping hacia dentro y hacia fuera del bus, y una de las SANs del proceso de cobertura, *At_TxrxA_Failure_Flip_InOut*, se activa para comenzar con el proceso de cobertura.

La SAN *At_TxrxA_Failure_Flip_InOut*, se muestra en la figura 15 es bastante grande y la figura está orientada para ver su complejidad. A continuación se analiza paso por paso por donde se dirige nuestro proceso de cobertura para entender el submodelo SAN.

Al principio, se decide estadísticamente si este fallo afecta a un nodo que produce un cambio o no en el sistema. Por ejemplo, puede afectar a un nodo que la MSU ha fallado aislando la comunicación y además el bus A ya ha fallado y por tanto el transceiver no puede afectar al sistema. Después, si se decide que falla en un nodo que afecta al sistema y le produce un cambio, se decide, también estadísticamente, si afecta a un nodo cuyo MSU ha fallado de forma que sus mecanismos ya no funcionan, o no. Esto se lleva a cabo en éste trozo del modelo que se muestra en la figura 16.

Continuando el caso, el place *TxrxA_Failure_BitFlip_InOut* tiene su marcaje a uno y activa la activity instantánea que decide si ese fallo se ha producido en un grupo de NOS que afecta el sistema o no. En este caso se escoge que afecte a un grupo de nodos que si afecta de alguna forma al sistema y su probabilidad se calcula con la suma de todos los marcajes de los NOSs que le pueden afectar el fallo dividido el número de ECRs de transceiver A aún funcionando:

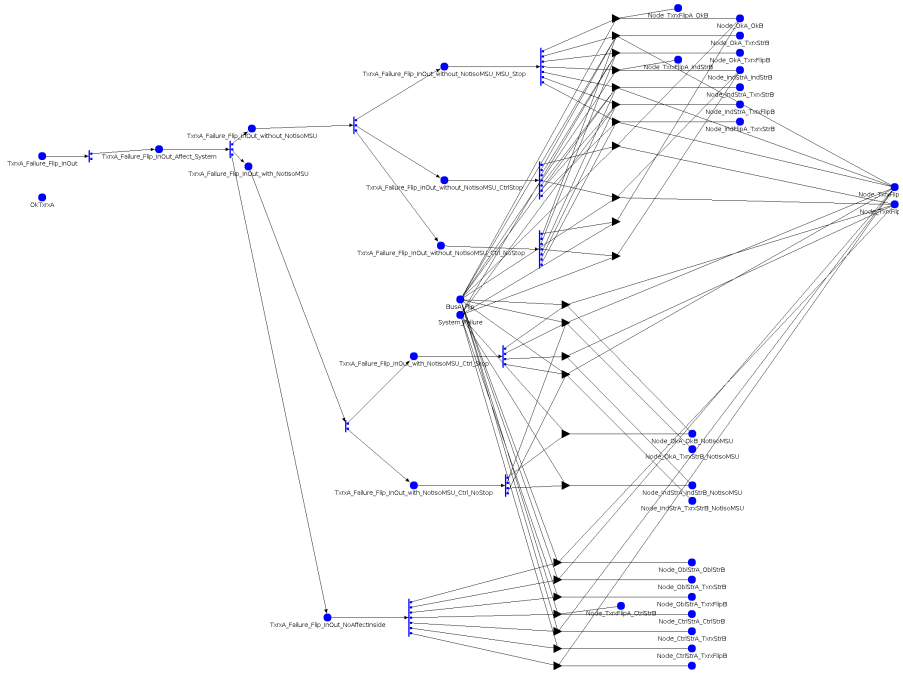

Figura 15: SAN *At_TxrxA_Failure_Flip_InOut*

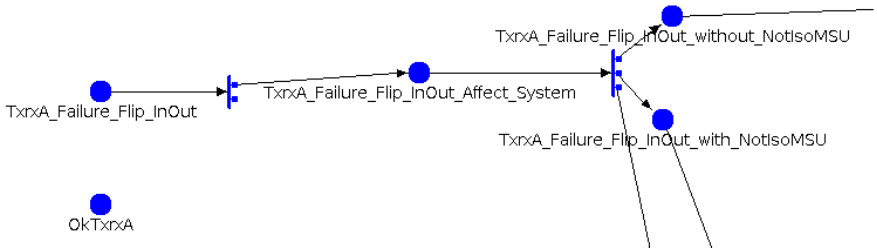

Figura 16: Parte de la SAN *At_TxrxA_Failure_Flip_InOut*

$$
\begin{aligned}
(((&Node\_OkA\_OkB \rightarrow Mark()+\\
&Node\_OkA\_OkB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_OkA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndFlipA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OblStrA\_OblStrB \rightarrow Mark()+\\
&Node\_OblStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OblStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_CtrlStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_CtrlFlipA\_TxrxStrB \rightarrow Mark()) * 1{,}0)\\
&/\\
&(OkTxrxA \rightarrow Mark()))
\end{aligned}
$$

a continuación el marcaje del place *TxrxA_Failure_Flip_InOut_Affect_System* se cambia a uno y por tanto se activa la activity instantánea que decide si el fallo puede afectar a un nodo que tenga la MSU con un fallo que no funcionan los mecanismos de detección de errores, sin este fallo o un caso en el que el fallo del transceiver no puede afectar internamente al nodo. Se escoge un fallo con la MSU funcionando para ver el funcionamiento de ésta, y su probabilidad es la suma de todos los nodos que tengan la MSU funcionando sus mecanismos dividido la suma de todos los nodos de la ecuación anterior:

$$
\begin{aligned}
(&Node\_OkA\_OkB \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OkA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndFlipA\_TxrxStrB \rightarrow Mark())\\
&/\\
((&Node\_OkA\_OkB \rightarrow Mark()+\\
&Node\_OkA\_OkB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_OkA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_IndFlipA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OblStrA\_OblStrB \rightarrow Mark()+\\
&Node\_OblStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OblStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_CtrlStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_CtrlFlipA\_TxrxStrB \rightarrow Mark()) * 1{,}0)
\end{aligned}
$$

con este camino de la activity el marcaje del place *TxrxA_Failure_Flip_InOut_without_NotIsoMSU* se cambia a uno. Ahora comienza la fase de mecanismos de detección de errores y tolerancia a fallos. Existen tres casos: que la MSU detecte el fallo con probabilidad MSU_Stops; que la MSU no detecte el error pero si lo haga el controlador CAN con probabilidad $(1{,}0 - MSU\_Stops) * Ctrl\_Stops$; por último, que no se detecte por ningún mecanismo y el error se propague por todo el nodo, su probabilidad es $(1{,}0 - MSU\_Stops) * (1{,}0 - Ctrl\_Stops)$. En este caso se elige que la MSU detecte el error y aísle la contribución el bus A. Por tanto, el nodo afectado transmitirá Bit-Flipping por el bus A pero no hacia dentro del nodo. Esta parte se puede ver en la figura 17.

La elección anterior incrementa el marcaje del place *TxrxA_Failure_Flip_InOut_without_NotIsoMSU_MSU_Stop* a uno y se activa la activity instantánea que finalmente decidirá que NOS es el afectado. Esta pro-

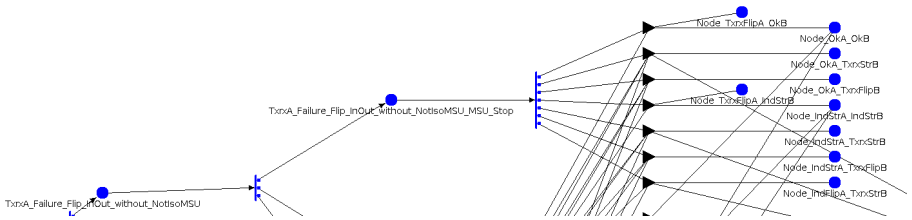

Figura 17: Parte de la SAN *At_TxrxA_Failure_Flip_InOut*

babilidad es el número de nodos en cada NOS dividido la suma de todos los nodos de la ecuación anterior. Se elige que afecta a un NOS que está totalmente sano y su probabilidad es la siguiente:

$$\begin{aligned} & Node\_OkA\_OkB \rightarrow Mark() \\ & / \\ & ((Node\_OkA\_OkB \rightarrow Mark()+ \\ & Node\_OkA\_TxrxStrB \rightarrow Mark()+ \\ & Node\_OkA\_TxrxFlipB \rightarrow Mark()+ \\ & Node\_IndStrA\_IndStrB \rightarrow Mark()+ \\ & Node\_IndStrA\_TxrxStrB \rightarrow Mark()+ \\ & Node\_IndStrA\_TxrxFlipB \rightarrow Mark()+ \\ & Node\_IndFlipA\_TxrxStrB \rightarrow Mark()) * 1{,}0) \end{aligned}$$

éste camino activa una output gate ya que tiene que hacer varias acciones: restar uno al marcaje del NOS totalmente sano; incrementarlo en el NOS que transmite Bit-Flipling por el transceiver A; y notificar que el bus A esta contagiado con Bit-Flipping:

$$\begin{aligned} Node\_OkA\_OkB \rightarrow Mark() - &= 1; \\ Node\_TxrxFlipA\_OkB \rightarrow Mark() + &= 1; \\ BusA\_Flip \rightarrow Mark() &= 1; \end{aligned}$$

éstos cambios producen que se active la SAN de evaluación y la SAN del proceso de cobertura de Bit-Flipping del bus A. Se explica ahora la SAN del proceso de cobertura del bus y se deja para más adelante la de la evaluación. Aunque antes del proceso de cobertura hay que analizar si se propaga el Bit-Flipping por el bus, ya que puede darse el caso que el bus A ya haya fallado y por tanto no se pueda influir en él. La SAN encargada se denomina *At_Bus_Prop* y se muestra en la figura 18.

Existen tres casos cuando un nodo activa el place *BusA_Flip* indicando que propaga Bit-Flipping por el bus A: que el bus ya haya fallado y no afecte; que esté funcionando y afecte; o que el bus esté funcionando pero el bus B no

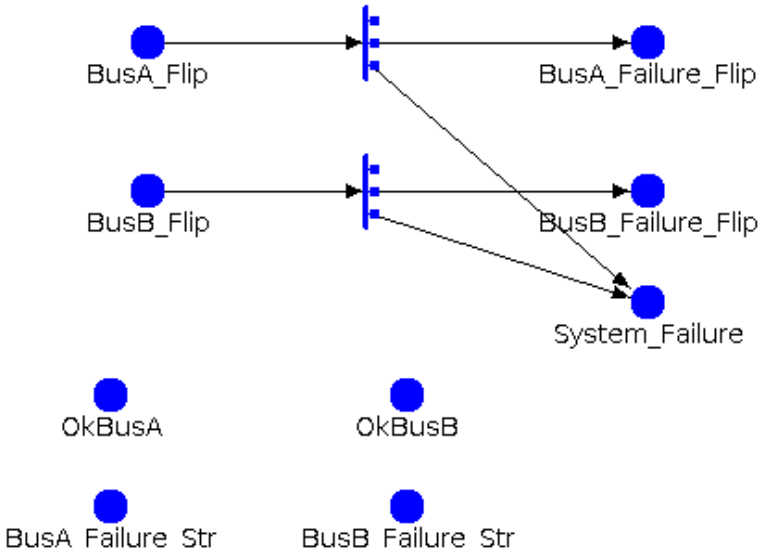

Figura 18: SAN *At_Bus_Prop*

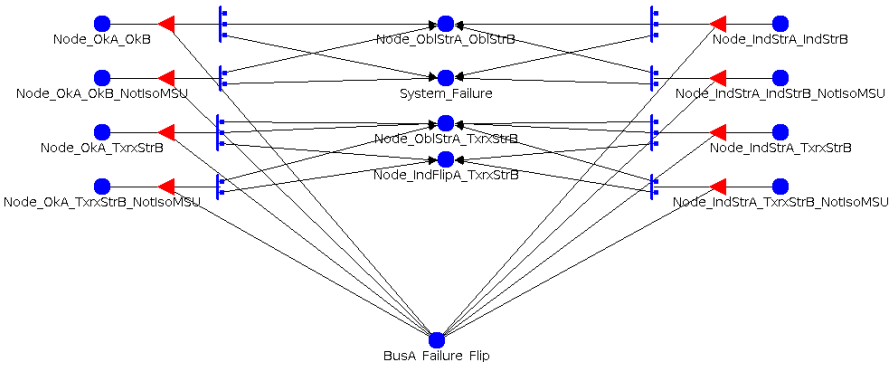

Figura 19: SAN *At_BusA_Failure_BitFlip*

y produce un fallo del sistema al no tener más buses en funcionamiento. Se decide que los dos buses funcionan correctamente y por tanto el bus A ahora tendrá un fallo de Bit-Flipping, ésto se calcula con la siguiente probabilidad, que será 100 %:

$$\begin{aligned}(OkBusA \rightarrow Mark()*\\ (1{,}0 - BusA\_Failure\_Flip \rightarrow Mark()))*\\ (1{,}0 - BusB\_Failure\_Str \rightarrow Mark()+\\ BusB\_Failure\_Flip \rightarrow Mark())\end{aligned}$$

Ahora ya si se activa la SAN del proceso de cobertura del bus A denominada *At_BusA_Failure_BitFlip* que se muestra en la figura 19.

Ésta SAN cambia el marcaje de todos los NOS a los que le afecta el fallo del bus según la cobertura de los mecanismos de detección de errores de la MSU y del controlador CAN. El proceso se hace por cada uno de los nodos que están en estos NOSs. Así que al final el marcaje de los NOSs afectados será 0 y se cambiará a otros nuevos. En nuestro caso, todos los nodos funcionaban correctamente (menos el que había fallado ya), y por ejemplo, tendremos 2 nodos más que tendrán que pasar por los mecanismos de detección de errores. Las posibilidades son las mismas que las explicadas en el fallo hacia dentro del transceiver: la MSU detecta el error; la MSU no lo detecta pero si el controlador CAN; ninguno lo detecta. Además las probabilidades son las mismas. En esta ocasión, se elige que el mecanismo de detección de la MSU falla pero no el del controlador CAN en las dos ocasiones del nodo. No tienen porque ser las mismas, podrían tener diferentes casos para cada nodo. Al final de la SAN, el marcaje del place *Node_OkA_OkB* será cero y el marcaje del place *Node_OblStrA_OblStrB* será dos. Éste último NOS indica que el controlador CAN ha desconectado totalmente el nodo de cualquier transmisión ya que le llega un error.

Como después de la SAN del proceso de cobertura del transceiver A, se activa otra vez la SAN de evaluación. La SAN encargada de la evaluación de los

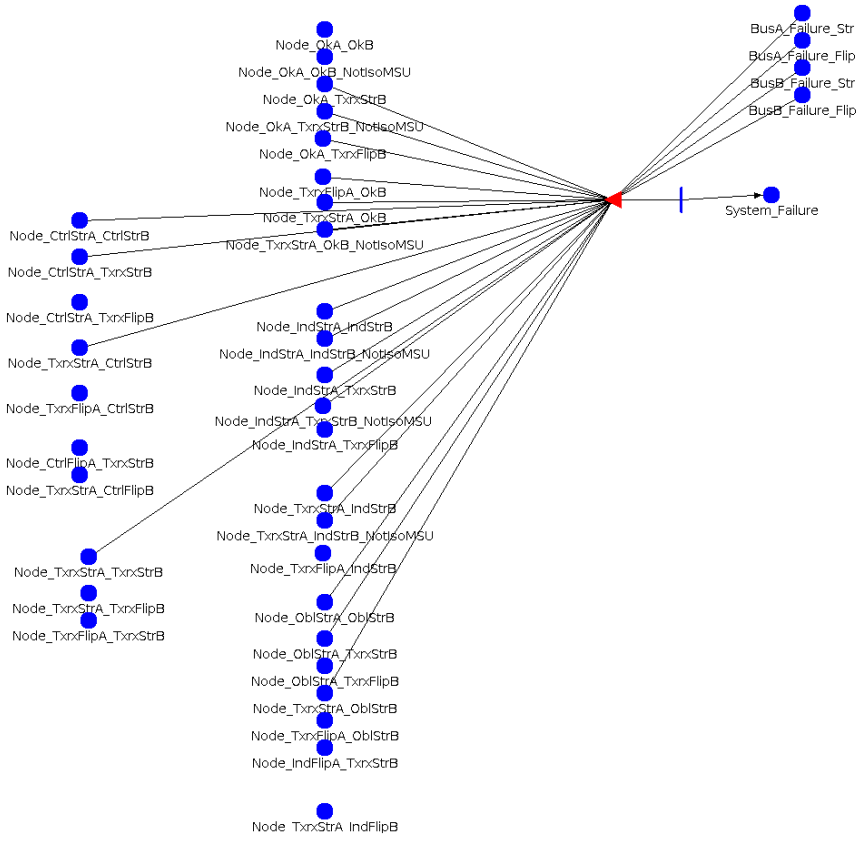

Figura 20: SAN *At_Evaluation*

nodos se denomina *At_Evaluation* y se muestra en la figura 20. La SAN tiene en cuenta el número de nodos en cada NOS, por eso es inteligible. Pero simplemente analiza dependiendo del estado de los buses, cuantos nodos todavía operan y se pueden comunicar. En el caso que estamos explicando, tenemos el bus A sin funcionar, por lo que se analiza cuantos nodos todavía se pueden comunicar en el bus B, que son uno de tres, y por lo tanto el sistema falla ya que definimos con la variable *Max_Nodes* que se necesita al menos dos nodos operativos. Se calcula la condición con una input gate que activa una activity instantánea que pone a uno el marcaje del place *System_Failure*. La condición de la input gate es la unión de los tres diferentes condiciones unidas por ORs y es la siguiente:

$$
\begin{aligned}
&((System\_Failure \rightarrow Mark() == 0)\ \&\& \\
&(BusA\_Failure\_Str \rightarrow Mark() == 0\ \&\& \\
&BusB\_Failure\_Str \rightarrow Mark() == 0\ \&\& \\
&BusA\_Failure\_Flip \rightarrow Mark() == 0\ \&\& \\
&BusB\_Failure\_Flip \rightarrow Mark() == 0)\ \&\& \\
&((Node\_IndStrA\_IndStrB \rightarrow Mark()+ \\
&Node\_IndStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+ \\
&Node\_IndStrA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_TxrxStrA\_IndStrB \rightarrow Mark()+ \\
&Node\_TxrxStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_OblStrA\_OblStrB \rightarrow Mark()+ \\
&Node\_OblStrA\_TxrxStrB \rightarrow Mark()+ \\
&Node\_TxrxStrA\_OblStrB \rightarrow Mark()+ \\
&Node\_CtrlStrA\_CtrlStrB \rightarrow Mark()+ \\
&Node\_CtrlStrA\_TxrxStrB \rightarrow Mark()+ \\
&Node\_TxrxStrA\_CtrlStrB \rightarrow Mark()+ \\
&Node\_TxrxStrA\_TxrxStrB \rightarrow Mark()) > Max\_Nodes))\| \\
&(((System\_Failure \rightarrow Mark() == 0)\ \&\& \\
&(BusA\_Failure\_Str \rightarrow Mark() == 1\ \| \\
&BusA\_Failure\_Flip \rightarrow Mark() == 1)\ \&\& \\
&BusB\_Failure\_Str \rightarrow Mark() == 0\ \&\& \\
&BusB\_Failure\_Flip \rightarrow Mark() == 0)\ \&\& \\
&((Node\_IndStrA\_IndStrB \rightarrow Mark()+ \\
&Node\_IndStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+ \\
&Node\_IndStrA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_TxrxStrA\_IndStrB \rightarrow Mark()+ \\
&Node\_TxrxStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+ \\
&Node\_OblStrA\_OblStrB \rightarrow Mark()+
\end{aligned}
$$

$$
\begin{aligned}
&Node\_OblStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_OblStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OkA\_TxrxFlipB \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB \rightarrow Mark()+\\
&Node\_OkA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()) > Max\_Nodes))\|\|\\
&(((System\_Failure \rightarrow Mark() == 0)\ \&\&\\
&(BusB\_Failure\_Str \rightarrow Mark() == 1\ \|\|\\
&BusB\_Failure\_Flip \rightarrow Mark() == 1)\ \&\&\\
&BusA\_Failure\_Str \rightarrow Mark() == 0\ \&\&\\
&BusA\_Failure\_Flip \rightarrow Mark() == 0)\ \&\&\\
&((Node\_IndStrA\_IndStrB \rightarrow Mark()+\\
&Node\_IndStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_IndStrA\_TxrxStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_TxrxStrA\_IndStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_IndStrB\_NotIsoMSU \rightarrow Mark()+\\
&Node\_OblStrA\_OblStrB \rightarrow Mark()+\\
&Node\_OblStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_OblStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_CtrlStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_CtrlStrB \rightarrow Mark()+\\
&Node\_TxrxStrA\_TxrxStrB \rightarrow Mark()+\\
&Node\_TxrxFlipA\_OkB \rightarrow Mark()+\\
&Node\_TxrxStrA\_OkB \rightarrow Mark()+\\
&Node\_TxrxStrA\_OkB\_NotIsoMSU \rightarrow Mark()) > Max\_Nodes))
\end{aligned}
$$

por último se activa la SAN *At_Simplification* que simplemente cambia el marcaje de todos los places a cero menos el del place *System_Failure* para ahorrar estados cuando se transforme a CMTC.

Parte IV

Resultados

16. Introducción

A continuación se muestran algunos análisis de sensibilidad de la fiabilidad del bus replicado CANEly obtenidos mediante el modelo que se ha construido en este proyecto. Estos análisis muestran cuál es la influencia de algunos aspectos clave sobre la fiabilidad de un sistema que utilice este bus. Además, se ha utilizado el modelo de ReCANcentrate para comparar la influencia de estos aspectos sobre la fiabilidad de un sistema al utilizar una u otra topología.

Para realizar los diferentes análisis de sensibilidad se ha partido de un caso de referencia en el que se ha asignado a los parámetros valores por defecto lo más realistas posibles. Los valores por defecto utilizados en el modelo del bus replicado se muestran en la tabla 1. Notar que para ReCANcentrate se han utilizado los mismos valores por defecto para aquellos parámetros que tiene en común con el bus replicado; el resto de valores por defecto para ReCANcentrate se pueden encontrar en [Bar10]. Para realizar cada análisis se parte de este caso de referencia y se varía, de la misma forma en uno y otro modelo, el valor de los parámetros que caracterizan el aspecto objeto del análisis.

Las métricas utilizadas para medir la fiabilidad son la FT/AR_1 -fiabilidad de un sistema que tolera/acepta el fallo/desconexión de uno de sus nodos- y la NFT/AR -fiabilidad de un sistema que no tolera/acepta el fallo/desconexión de ningún nodo (ver Sección 10). Sin embargo, se debe notar que en las figuras no se muestran los resultados de NFT/AR de ReCANcentrate, ya que la fiabilidad obtenida para la estrella replicada utilizando esta métrica está por muy por debajo del resto de resultados.

De todas formas es necesario aclarar que con el fin de presentar los resultados de la forma más clara posible, las figuras no muestran directamente los valores de la FT/AR_1 y la NFT/AR. En lugar de ello, muestran el *Tiempo de Misión* (*Mission Time*, TM) [MK05] que se obtiene al mediar la fiabilidad con cada una de estas métricas. El MT es el máximo tiempo durante el cual el sistema exhibe una fiabilidad (FT/AR_1 o NFT/AR, según el caso) igual o mayor a un cierto umbral. En particular, el umbral escogido es un valor de fiabilidad (FT/AR_1 o NFT/AR) igual a 0,99999, el cuál se corresponde con la fiabilidad que requieren los sistemas de control de aceleración en automoción durante un MT de 10 horas [MK05].

Finamente, es importante notar que todos los análisis que se muestran aquí consideran sistemas constituidos por 3 nodos. Esto se debe a que si bien el tiempo necesario para ejecutar y resolver el modelo presentado en este proyecto es de aproximadamente una hora para 3 nodos, la memoria principal del computador que se ha utilizado se agota al considerar un número mayor de ellos. Concretamente, el computador que se ha utilizado tiene las siguientes características:

- Procesador Intel(R) Core(TM) i7 CPU 870 2.93GHz (8cores).
- Ram 16 GB.
- Linux: Ubuntu 12.04 Kernel 3.8.0-46 generic.

Como se comentará en la Sección 22, una de las futuras extensiones del trabajo presentado en este proyecto consistirá en tratar de resolver el modelo mediante un computador con mayores prestaciones, así como investigar la forma de modelar el sistema de una forma más eficiente para poder analizar la fiabilidad con un mayor número de nodos.

De todas formas, los resultados obtenidos con 3 nodos permiten extraer una serie de conclusiones muy interesantes sobre la fiabilidad de un bus replicado en comparación con la fiabilidad de una estrella replicada. En este sentido hay que notar que si bien no se ha podido medir la fiabilidad para un mayor número de nodos, existen aplicaciones críticas en las que se utilizan 3 réplicas para conseguir una alta fiabilidad, p.e. para construir una Faul Tolerant Unit (FTU) [PS]. En cualquier caso, los resultados obtenidos con 3 nodos, además de ser interesantes per se, muestran el interés que tiene construir un modelo que permita estudiar cómo varía la fiabilidad del bus replicado a medida que el número de nodos aumenta.

17. Failure rate de la réplica de bus

En esta sección se analiza el impacto del failure rate de la réplica de bus en un sistema que utiliza una topología de bus replicado.

Como se puede ver en la figura 21, el MT del bus replicado es muy diferente dependiendo de si se utiliza la NFT/AR o la FT/AR_1 para medir su fiabilidad. Esto significa que para obtener beneficios al utilizar un bus replicado es muy importante dotar al sistema con la capacidad para tolerar o aceptar el fallo o la desconexión de un nodo.

En el caso de ReCANcentrate, al no tener un bus de campo, el MT es constante.

También es importante notar que no merece la pena aumentar la fiabilidad del bus en sí, ya que apenas se observan diferencias en el MT cuando el orden de magnitud del failure rate del bus disminuye por debajo de su valor por defecto, i.e. 10^{-6}.

18. Failure rate del transceiver

En esta sección se analiza cómo varía el MT para diferentes failure rates del transceiver. Intuitivamente, el failure rate del transceiver es un aspecto que debe tener una influencia notable sobre la fiabilidad que un sistema puede alcanzar al utilizar tanto un bus replicado como una estrella replicada. En el caso del bus replicado el número de transceivers se duplica con respecto a un bus simple, ya que en el primero cada nodo necesita un transceiver independiente para transmitir y recibir por cada réplica del bus. En el caso de ReCANcentrate el incremento del número de transceivers por nodo es aún más acusado, ya que incluye 4 transceivers por nodo.

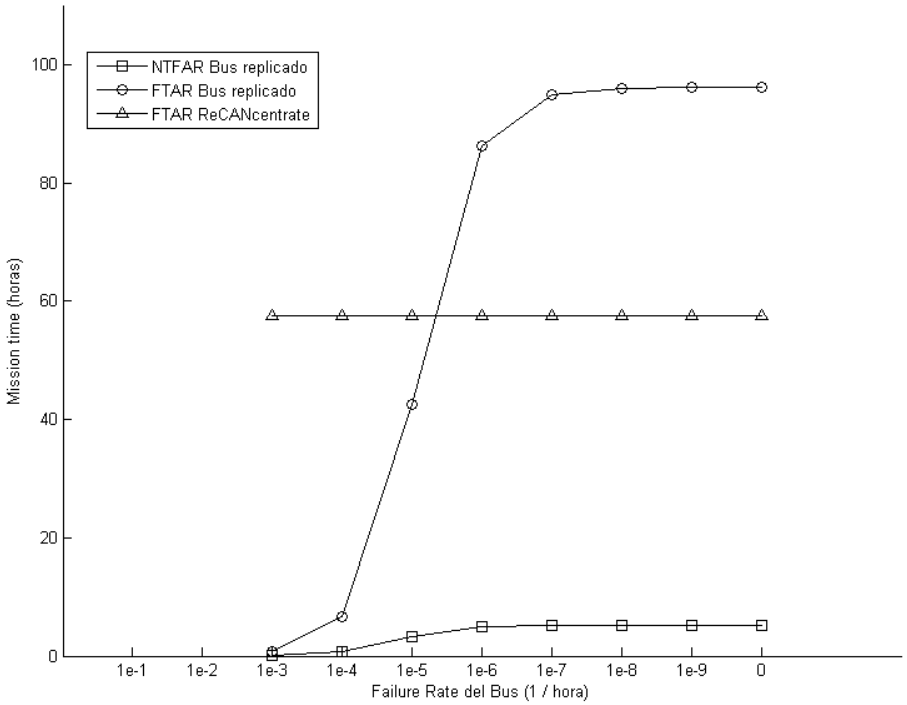

Figura 21: Mission Time vs Failure Rate del Bus

Como se puede ver en la figura 22, los resultados del análisis muestran que el failure rate del transceiver tiene una gran influencia sobre la fiabilidad que se puede conseguir con un bus replicado. Cuando se utiliza la NFT/AR para medir la fiabilidad, el MT del bus replicado decrece rápidamente cuando el orden de magnitud del failure rate del transceiver aumenta con respecto a su valor por defecto, i.e. 10^{-7} hour^{-1}. Esto indica que es importante utilizar transceivers de calidad suficiente cuando se utiliza un bus replicado.

Pero más interesante aún es el hecho de que cuando se mide la fiabilidad utilizando la FT/AR_1, el failure rate del transceiver tiene una influencia muy grande. Esta influencia es enorme cuando la calidad del transceiver decrece. Por el contrario, la influencia es menor a medida que la calidad del transceiver mejora. De todas formas, la figura muestra que, aún así, puede ser interesante mejorar los transceivers. Concretamente, si se mejora la fiabilidad del transceiver para conseguir que su failure rate decrezca e un orden de magnitud con respecto a su valor por defecto, se puede conseguir un incremento de casi 10 horas en el MT.

También es interesante notar que si se comparan estos resultados con la sensibilidad del MT respecto al failure rate del bus, se observa que merece más la pena mejorar la fiabilidad de los transceivers que la de las réplicas de bus. Concretamente, el MT que se puede conseguir utilizando buses y transceivers perfectos (con un failure rate igual a 0) es de hasta 96 y 130 horas respectivamente. Por un lado hay que notar que el fallo de un transceiver puede corromper

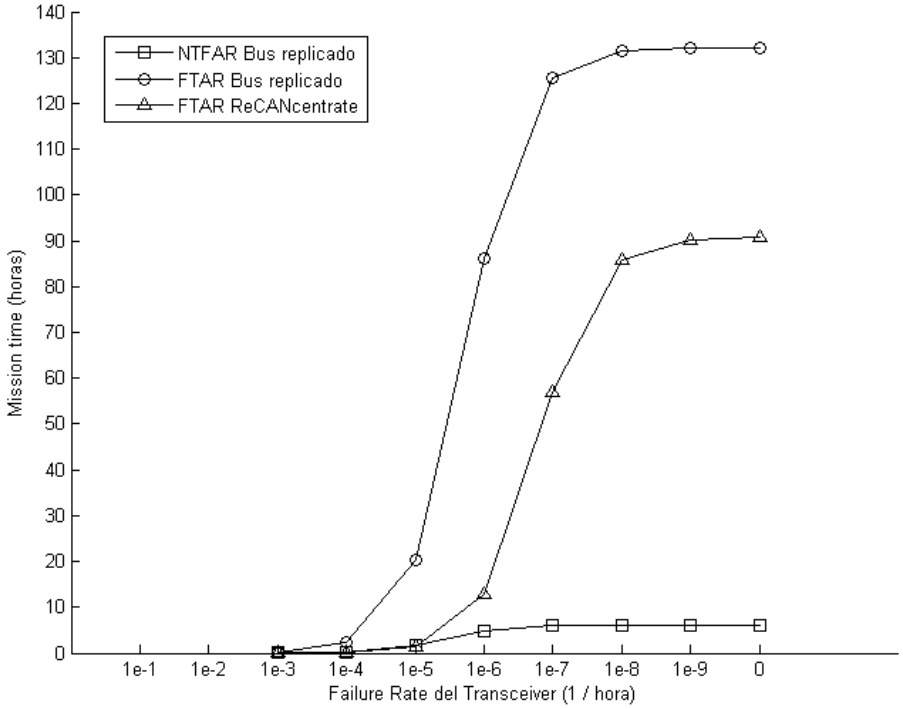

Figura 22: Mission Time vs Failure Rate del Transceiver

la replica de bus a la que está conectado y, por otro lado, se debe tener en cuenta que el sistema incluye un número mucho mayor de transceivers que de replicas de bus. Por tanto, los transceivers tienen una mayor influencia sobre la fiabilidad del sistema que las réplicas de bus.

En cuanto a ReCANcentrate, se puede observar que al medir su fiabilidad con la FT/AR_1, su MT está muy por debajo de la del bus replicado.

Por último, observando las tres curvas se puede ver que a partir de failure rates del transceiver del orden de 10^{-4}, 10^{-5} el MT mejora considerablemente en todos los casos hasta que se llega a un failure rate del orden de 10^{-7}, 10^{-8}. Éste resultado es muy útil a la hora de intentar mejorar el MT del sistema, independientemente de la topología utilizada. Cómo se menciona en la sección 12, el failure rate de un transceiver comercial es del orden de 10^{-7}; por tanto invertir en la calidad de los transceivers para conseguir failure rates extremadamente bajos (cercanos a 0) no tiene mucho sentido en la práctica para mejorar el sistema.

19. Failure rate del Controlador CAN

Esta sección presenta los resultados obtenidos al analizar la sensibilidad con respecto al failure rate del controlador CAN. Intuitivamente, la fiabilidad del

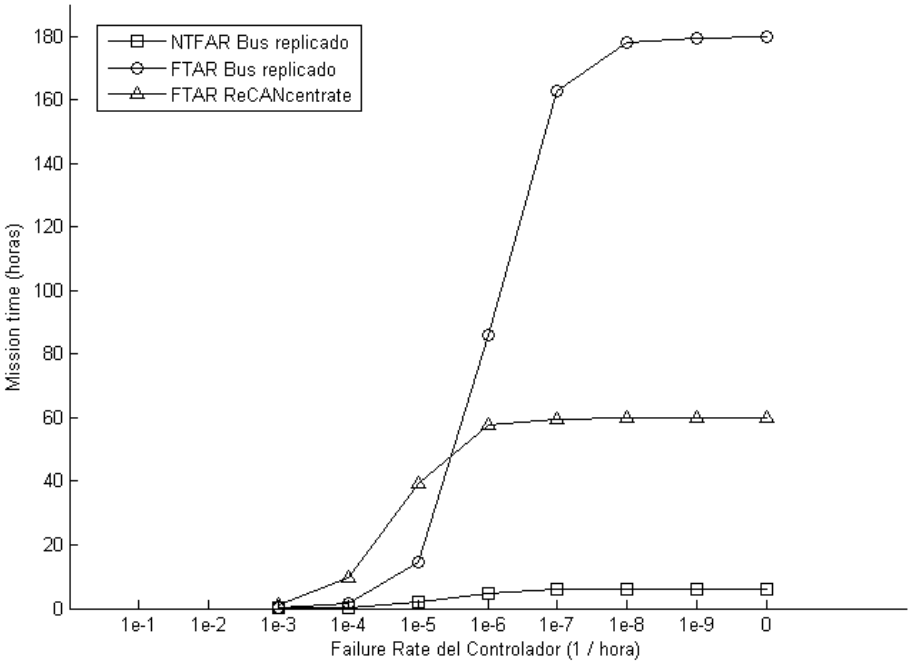

Figura 23: Mission Time vs Failure Rate del Controlador

controlador debería tener un impacto incluso mayor que la del transceiver en el bus replicado; ya que no existe ningún mecanismo que impida que un controlador CAN que falle corrompa todas las réplicas de bus. Esto no ocurre en el caso del fallo de un transceiver, ya que los errores generados por éste sólo pueden corromper a todas las réplicas de bus si la MSU y el controlador no lo consiguen aislar.

La figura 23 corrobora estas consideraciones cuando se utiliza la FT/AR_1 para medir la fiabilidad. Como se puede observar, si se mejora el failure rate por defecto del controlador en un orden de magnitud, i.e. si dicho orden pasa de 10^{-6} a 10^{-7}, el MT se incrementa aproximadamente en un 100 %; mientras que en el caso del transceiver se consigue un incremento aproximado del 5 % al decrementar en un orden de magnitud el valor por defecto de su failure rate (ver figura 22).

Finalmente, la figura 23 muestra que, en ReCANcentrate, la influencia del failure rate del controlador CAN sobre la fiabilidad del sistema es muy similar a la del transceiver. También se puede observar que la calidad del controlador no es un factor determinante para incrementar su MT. Esto quiere decir que en ReCANcentrate hay otros aspectos que tienen una influencia mucho mayor que los transceivers. Por ejemplo, en ReCANcentrate hay componentes que juegan un papel vital en la fiabilidad del sistema y que son menos fiables que los transceivers, p.e. los hubs.

20. Cobertura de bit-flipping de la MSU y del hub

En esta sección se compara la fiabilidad que se puede alcanzar al variar la probabilidad con la que el nodo (la MSU) y el hub son capaces de aislar correctamente un fallo de tipo bit-flipping, respectivamente, en un bus replicado y en una estrella replicada.

La primera conclusión que se puede extraer de la figura 24 es que esta cobertura es muy importante en ambas topologías. En el caso del bus replicado es igual o más importante que el failure rate de los transceivers y del controlador CAN. Pero lo más interesante es que para el caso de ReCANcentrate, un aumento de esta cobertura dispara el MT que se puede conseguir con la estrella replicada. Simplemente con incrementar la cobertura que provee el hub ligeramente por encima del valor por defecto, p.e. incrementándola desde el 95 % hasta el 96 %, ReCANcentrate es mejor que el bus replicado. Y no solamente eso, sino que el MT de ReCANcentrate se incrementa hasta en un 400 % para coberturas cercanas al 100 %.

Otro resultado interesante que se puede observar en la figura 24 es que el MT del bus replicado desciende rápidamente al disminuir la cobertura por debajo de su valor por defecto. Este hecho puede indicar, indirectamente, que la fiabilidad del bus replicado también decrecería rápidamente a medida que aumentase el número de nodos. Para entender mejor esta posibilidad, primero se debe notar que un fallo de tipo bit-flipping que corrompe una réplica de bus requiere que la MSU de cada uno de los nodos conectados a dicha réplica la aisle. En este punto, imaginemos que se tienen 3 nodos conectados a una réplica de bus que se ha corrompido por causa de un bit-flipping. En este caso, la probabilidad de que los nodos la aislen con éxito es de $0{,}95^3 = 0,857$. Ahora imaginemos que son 15 los nodos que están conectados a la réplica de bus que está corrompida. En este caso, la probabilidad con la que los nodos la aislan es de $0{,}95^{15} = 0,463$. Ahora bien, hay que notar que ésta es la probabilidad con la que 3 nodos conseguirían aislar a una réplica de bus corrompida por un bit-flipping si la cobertura de la MSU fuera de $0,463^{\frac{1}{3}} = 0,78$. Por tanto, no es descabellado pensar que los resultados de la figura 24 se pueden extrapolar hasta cierto punto para el caso en el que en lugar de estar analizando la influencia de la cobertura que provee la MSU, se está analizando la influencia del número de nodos.

Por último, es importante destacar que, al contrario de lo que sucede cuando se analiza la sensibilidad de la MT respecto al failure rate del transcevier y el controlador CAN, la MT aumenta exponencialmente con la cobertura. Por tanto, a efectos prácticos, es recomendable intentar proveer siempre la máxima cobertura que sea posible.

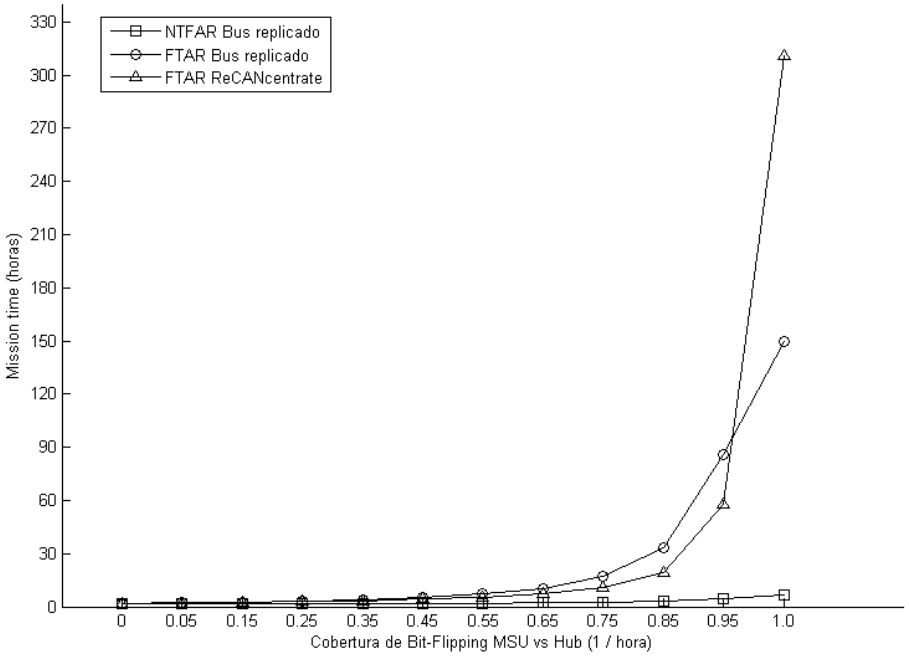

Figura 24: Mission Time vs Cobertura de Bit-Flipping en la MSU o Hub

Parte V

Conclusiones

21. Resumen y conclusiones sobre el trabajo realizado

Este documento presenta el diseño e implementación de un modelo que permite cuantificar la fiabilidad de un sistema basado en un bus CAN replicado denominado CANEly. El modelo ha sido diseñado e implementado mediante una extensión al formalismo de las redes de Petri denominado *Stochastic Activity Network* (SAN), y ha sido implementado mediante la herramienta software Möbius.

Es importante destacar que este proyecto ha cumplido con todos los objetivos que se plantearon inicialmente.

Un aspecto que es muy importante resaltar en este sentido es que la estrategia de modelado aquí empleada se basa en la que el SRV utilizó para construir los modelos de fiabilidad de sistemas basados en un bus CAN simple, en una topología de estrella simple llamada CANcentrate, y en una topología de estrella replicada conocida como ReCANcentrate. Se decidió utilizar esta estrategia para poder comparar los resultados de fiabilidad del bus replicado con los valores de fiabilidad de estas otras topologías.

Gracias a esta decisión el proyecto no sólo ha servido para cumplir con los objetivos planteados inicialmente, sino también para mostrar ciertas limitaciones que presentan tanto la estrategia de modelado de partida, como el formalismo SAN. Concretamente, se ha puesto de manifiesto la dificultad de evitar una explosión de estados al modelar la dependability de topologías de comunicaciones en las que, a diferencia de las estrellas, los mecanismos de tolerancia a fallos se encuentran fuertemente distribuidos.

Para evitar la explosión de estados ha sido necesario adaptar y extender la estrategia de modelado inicial. De esta forma, y dados los recursos de computación de los que se disponía en este proyecto, el modelo resultante es capaz de calcular la fiabilidad para sistemas que tienen hasta un máximo de 3 nodos.

Aunque el número de nodos que se puede modelar es pequeño, los resultados obtenidos son reveladores. Se ha mostrado que cuando un sistema de control distribuido consta de pocos nodos, el bus replicado puede alcanzar una fiabilidad mucho mayor que una estrella replicada.

Es más, se ha cuantificado la fiabilidad que se puede alcanzar con este tipo de sistemas pequeños basados en buses replicados, y se ha comparado con la que se conseguiría con una estrella replicada, dependiendo de aspectos clave como el failure rate de los componentes hardware, o la cobertura de la contención de errores. En general, se puede concluir que con pocos nodos el bus replicado es mejor que la estrella replicada, excepto cuando la cobertura que ésta provee aumenta por encima del 95 %.

Incluso, algunos de los resultados obtenidos permiten especular sobre el hecho de que la fiabilidad del bus replicado es particularmente sensible al número de nodos. En este sentido, estos resultados invitan más si cabe a extender el

trabajo realizado en este proyecto para corroborar esta sospecha.

22. Trabajo futuro

El trabajo futuro más relevante que se ha planteado al acabar este proyecto consiste en estudiar cómo se puede cuantificar la fiabilidad de un sistema basado en un bus replicado con un número elevado de nodos.

Una posibilidad que se puede explorar a corto plazo es intentar resolver el modelo desarrollado en este proyecto mediante un computador con mayores prestaciones que el que aquí se ha utilizado.

Otra alternativa a medio o largo plazo consiste en investigar cómo modelar el sistema de una forma más eficiente, ya sea replanteando o extendiendo aún más la estrategia de modelado en sí y/o utilizando otro formalismo.

En este sentido, una posibilidad sería abandonar la idea de utilizar places para representar el número de nodos que están en un determinado *Node Operational State* (NOS). En lugar de ello se podría intentar representar dentro de un submodelo el estado de un nodo mediante los places que hagan falta y, después, replicar dicho modelo tantas veces como nodos haya en el sistema. De hecho, esta solución se descartó al principio del proyecto por su alta complejidad para diseñar el modelo; por no seguir la misma estrategia de modelado que se usó para los modelos de CAN, CANcentrate y ReCANcentrate; y por el alto consumo de procesador que necesitaría el software para solucionar el modelo resultante. Sin embargo, y a pesar de consumir más procesador, es posible que esta alternativa reduzca las necesidades de memoria y permita resolver el modelo para un número mayor de nodos.

23. Opinión personal

He disfrutado mucho con la realización del proyecto por muchas razones. Primero, por trabajar en un proyecto que no se circunscribe a un único campo, sino que toca varios temas, incluyendo aspectos relacionados con la fiabilidad de componentes hardware, protocolos y topologías de comunicaciones, diseño de modelos estocásticos, etc. En este sentido, el proyecto me ha permitido ampliar mis conocimientos en diferentes materias. Además, he aprendido que a pesar de que para solucionar un problema se recorren caminos que no llevan a ninguna parte, se aprende igual de estos caminos que de los caminos correctos.

Segundo, debido a la imposibilidad de realizar el proyecto en la UIB, debido a mi beca Erasmus en Suecia y a estudiar un máster fuera de las Islas Baleares, he podido comprobar cómo es trabajar a distancia con mi tutor. Hemos utilizado no sólo llamadas por Skype y e-mails, sino que además hemos usado otras herramientas para organizar el trabajo, intercambiar ideas, sincronizar diferentes versiones del modelo y la memoria, etc. Creo que es un factor que ha enriquecido mi experiencia, ya que es mucho más difícil comunicar el estado de

un trabajo a distancia, que en persona; y por tanto me ha permitido mejorar cómo expresar mis opiniones y explicar mis ideas.

Tercero, en este proyecto también he tenido la oportunidad de mejorar mis habilidades de escritura. En mi opinión, éste ha sido uno de los aspectos más complicados del proyecto y aún me queda un largo camino por recorrer. Sin embargo, ya empiezo a comprender cómo organizar y escribir apropiadamente un documento técnico en el campo de la ingeniería.

Por último, este proyecto tiene una fuerte componente de investigación y, por tanto, es diferente de un proyecto típico de ingeniería. Ésto me ha permitido experimentar la investigación en primera persona, y descubrir una nueva pasión que, finalmente, me ha ayudado a dar un giro en mi carrera profesional aceptando un puesto de estudiante de doctorado en Suecia.

Bibliografía

[AL81] T. Anderson and P.A. Lee. *Fault Tolerance - Principles and Practice*. Prentice Hall, 1981.

[AM02] M. Abdollahi and A. Mocaghar. Application of stochastic activity networks on network modelling. In *10th International Conference on Software, Telecommunications and Computer Networks*, Dubrovnik, Croatia, 2002. SoftCom'02.

[AvDF11] Damien Aza-vallina, Bruno Denis, and Jean-marc Faure. Communications reliability analysis in networked embedded systems. In *European Conference on Safety and Reliability - ESREL 2011*, 2011.

[Avi95] Algirdas Avizienis. Building dependable systems: How to keep up with complexity. In *Special Issue of the IEEE 25th Int. Symp, Fault-Tolerant Computing. FTCS-25*, pages 4–15, Cancun, Mexico, 1995.

[Bar10] Manuel Barranco. *Improving Error Containment and Reliability of Communication Subsystems Based on Controller Are Network (CAN) by Mean of Adequate Star Topologies*. PhD thesis, Universitat de les Illbes Balears, 2010.

[BB03] I. Broster and A. Burns. An analyzable bus-guardian for event-triggered communication. In *Proceedings of the 24th Real-time System Symposium (RTSS)*, pages 410–419, Cancun, Mexico, 2003. IEEE.

[BPA] M. Barranco, J. Proenza, and L. Almeida. Reliability improvement achiavable in can-based systems by means of the recan-centrate replicated star topology. In *8th IEEE International Workshop on Factory Communication Systems*, Nancy, France. IEEE.

[BPA11] M Barranco, J Proenza, and L Almeida. Quantitative Comparison of the Error-Containment Capabilities of a Bus and a Star Topology in {CAN} Networks. *IEEE Transactions on Industrial Electronics*, 53(3):802–803, 2011.

[Car82] W.C. Carter. A time for reflection. In *Proceedings of the IEEE 12th Int. Symp. Fault-Tolerant Computing. FTCS-12*, Santa Monica, California, USA, June 1982.

[CFJ+91] Joshep A. Couvillion, Roberto Freire, Ron Johnson, W. Douglas, M. Akber Qureshi, Manish Rai, WIlliam H. Sanders, and Janet E. Trevdt. IEEE Software, September 1991.

[CGP99] E. M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.

[Cha09] Scott Chacon. *Pro Git*. Apress Media LLC, Berkeley, CA, 2009.

[CiA] Can data link layer. technical report, CAN in Automation (CiA), Am Weichselgarten 26.

[ISO93] Iso11898. road vehicles - interchange of digital information - controller area network (can) for high-speed communication, 1993.

[ISO03a] Iso11898-1. Controller Area Network (CAN) - Part 1: Data link layer and physical signalling, 2003.

[ISO03b] Iso11898-1. Controller Area Network (CAN) - Part 2: High-speed medium access unit, 2003.

[JJJ00] J.Proenza and J.Miro-Julia. Majorcan: A modification to the controller area network to achieve atomic broadcast. In *IEEE Int. Work-shop on Group Communication and Computions*, Taipei, Taiwan, 2000.

[KNM90] David J. Klinger, Yoshinao Nakada, and Maria A. Menendez. 1990.

[Kok97] Hermann Koketz. *Dependability In Real-Time Systems: Design Principles for Distributed Embedded Applications*, chapter 2.4, pages 39–42. Kluwer Academic Publishers, Boston, Dordrech, London, 1997.

[Lap92] Jean-Claude Laprie. *Dependability: Basic Concepts and Terminology*. Springer-Verlag Wien, New York, USA, 1992.

[Lap01] Jean-Claude Laprie. Fundamental concepts of dependability. Technical report 739, University of Newcastle upon Tyne, School of Computing Science, 2001.

[MCACGC+11] J. Munoz-Castaner, R. Asorey-Cacheda, F.J. Gil-Castineira, F.J. Gonzalez-Castano, and P.S. Rodriguez-Hernandez. A Review of Aeronautical Electronics and Its Parallelism With Automotive Electronics. *IEEE Trans. on Industrial Electronics*, 58(7):3090–3100, 2011.

[MFT00] Joseph K. Muppala, Ricardo M. Fricks, and Kishor S. Trivedi. *Techniques of System Dependability Evaluation*, chapter 12. Grassmann Winfried, 2000.

[MK05] J Morris and P Koopman. Representing Design Tradeoffs in Safety-Critical Systems. In *Proceedings WADS, St. Louis, MO.*, 2005.

[MTK95] Malhotra, M. Trivedi, and K.S. 1995.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, 1981.

[Pol96] S. Poledna. *System model and terminology. In Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism, Real Time Systes*, chapter 3, pages 21–30. Kluwer Academic Publishers, Boston, Dordrech, London, 1996.

[Pow92] D. Powell. Failure mode assumptions and assumption coverage. In *Digest of Papers of the IEEE 22th Int. Symp. Fault-Tolerant Computting FTCS-22*, pages 386–395, Boston, Massachusetts, USA, July 1992.

[Pro07] Julián Proenza. *RCMBnet: A distributed Hardware and Firmware Support for Software Fault Tolerance.* PhD thesis, Universitat de les Illbes Balears, January 2007.

[PS] J.R. Pimentel and M. Salazar. Dependability of distributed control system fault tolerant units. *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, pages 3164–3169.

[RVA99] José Rufino, Paulo Veríssimo, and Guilherme Arroz. A columbus egg idea for can media redundancy, 1999.

[RVAR98] J. Rufino, P. Veríssimo, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in can. In *FTCS-29. The 28th International Symposium on Fault-Tolerant Computing*, Munich, Germany, 1998.

[SdFP11] Randal L. Schwartz, Brian d Foy, and Tom Phoenix. *Learning Perl. Making Easy Things Easy and Hard Things Possible. 6th Edition.* O'Reilly Media, 2011.

[Sho02] Martin L. Shooman. *Reliability of Computer Systems and Networds.* John Wiley & Sons, Inc, 605 Third Avenue, New York, USA, 2002.

[StBoT04] W. Sanders and the Board of Trustees. 2004.

[STP96] Robin A. Sahner, Kishor S. Trivedi, and Antonio Puliafito. *Perfomance and Reliability Analysis of Computer Systems.* Kluwer Academic Publisher, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061, USA, 1996.

[TMGT93] L. Tomek, V. Mainkar, R. Geist, and K. Trivedi. Reliability modeling of life-critical, real-time systems. In *Proceding of the IEEE*, 1993.