Diseño de sfiCAN: un inyector físico de fallos para redes CAN basado en una topología en estrella

David Gessner, Manuel Barranco, Alberto Ballesteros, Julián Proenza
Dpt. Matemàtiques i Informàtica, Universitat de les Illes Balears, Spain
{david.gessner, manuel.barranco, julian.proenza}@uib.es

***Abstract*— Este artículo presenta el diseño y parcial implementación de sfiCAN: un inyector físico de fallos para el bus CAN que permite la creación de una gran variedad de escenarios de fallos. El inyector de fallos reemplaza la topología bus de CAN por una estrella, cuyo elemento central es un *hub* con mecanismos de inyección de fallos. El inyector de fallos es fácilmente configurable, desde un PC conectado a un puerto dedicado del *hub*. Para ello se usa una especificación de inyección de fallos, la cual es traducida a un protocolo de configuración por encima de CAN. Este protocolo sólo es usado entre tests y por ello no interfiere en la ejecución de éstos. El propósito del inyector de fallos es comprobar el comportamiento de los nodos de una red CAN en presencia de errores en el canal. En particular, el comportamiento de los controladores CAN de los nodos y del *software* que se ejecuta en ellos, para los cuales la topología de estrella es transparente.**

I. Introducción

Controller Area Network (CAN) [1] es un bus de campo ampliamente usado en sistemas distribuidos empotrados (DES) y entornos hostiles. Aunque CAN posee múltiples mecanismos que lo hacen robusto en tales entornos, los fallos son todavía posibles. Por lo tanto, en aplicaciones críticas, se requiere un estudio preciso de la respuesta del sistema ante tales fallos. Para ello, la inyección de fallos, que consiste en la generación artificial de fallos, es particularmente útil.

Los fallos artificiales pueden ser generados en un modelo simulado del sistema o en un prototipo físico. El primero tiene la ventaja de que puede ser testeado antes de que el prototipo esté disponible, pero es menos realista y preciso. Por lo tanto, para sistemas críticos se recomienda evaluar el sistema con un inyector físico de fallos, una vez el prototipo está disponible. Esta evaluación debe incluir, particularmente, los nodos del DES, que deben asemejarse lo más posible a los de producción, es decir, deben ser evaluados con el *software* de producción. Es más, el comportamiento del sistema debe ser testeado bajo una variedad de fallos suficiente, que puede incluir escenarios de fallo complejos y/o no deseados, como fallos de inconsistencia, es decir, fallos que afectan sólo a algunos nodos. Hasta donde sabemos, en el momento de la redacción de este artículo, no existe ningún inyector de fallos para CAN que satisfaga estos requerimientos. Este artículo presenta el diseño y preliminar implementación de un inyector físico de fallos para CAN que los satisface.

El inyector utiliza una topología de estrella para CAN, la cual es transparente desde el punto de vista de los nodos. La figura 1 muestra la arquitectura de sfiCAN. Cada nodo se conecta al *hub*, donde reside un módulo de inyección de fallos,

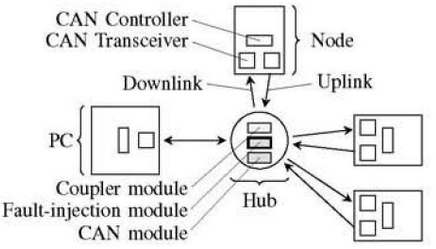

Fig. 1. Arquitectura de inyección de fallos.

por medio de un enlace dedicado, compuesto por un *uplink* y un *downlink*. El controlador CAN del nodo se conecta a su enlace mediante dos *transceivers* usando una configuración idéntica a la utilizada en CANcentrate [2]. Dentro del *hub*, un *módulo de acoplamiento* implementa la AND cableada de CAN, realizando una AND lógica de las señales del *uplink*, para después transmitir el resultado a través de todos los *downlinks*. El bus CAN se implementa de manera lógica, pero con la ventaja de poder inyectar los fallos localmente en los nodos, con suficiente granularidad para afectar únicamente a la señal entrante o saliente. El *hub* también posee un *módulo CAN*. Su función es observar la señal acoplada, obtenida del módulo de acoplamiento, y informa al módulo de inyección de fallos el campo y el bit de la trama transmitida en el instante. Si se añaden más módulos al *hub*, el inyector de fallos puede ser usado en topologías CAN de estrella más avanzadas, por ejemplo, si se implementa el módulo de tratamiento de fallos de CANcentrate [2], el inyector de fallos puede usarse para inyectar fallos en una red CANcentrate.

El *hub* posee un puerto dedicado al que se conecta un computador personal (PC). Este enlace no se divide en *up-* y *downlink* ya que no se inyectan fallos en él. Además, esto permite conectar el PC usando una controladora CAN estándar.

La conexión PC-*hub* permite configurar de manera sencilla el módulo de inyección de fallos. Esto es posible gracias a que este módulo es una instancia de un *componente de red configurable* (NCC), es decir, un componente conectado a la red que puede recibir su configuración desde esta misma red.

El resto del artículo se organiza de la siguiente manera. La sección II describe los modos de funcionamiento de un NCC, mientras que la sección III explica como se especifica la configuración que va a ser aplicada al módulo de inyección de fallos. La sección IV describe, a grandes rasgos, la implementación de este módulo. La sección V describe un ejemplo de inyección de fallos y la sección VI trabajos relacionados. Finalmente, la sección VII concluye este artículo

y pone énfasis al trabajo futuro.

II. Modos de operación de un NCC

Un componente de red configurable puede trabajar en cuatro modos: *modo de configuración*, *modo ocioso*, *modo de espera* y *modo de ejecución*.

Un NCC entra en modo de configuración cuando el PC envía una *trama entra-modo-configuración*. El identificador de esta trama es el de mayor prioridad, que está reservado para ser transmitido exclusivamente por el PC.

Durante el modo de configuración el PC transmite *comandos de configuración* codificados en tramas CAN. Es más, cada NCC se identifica mediante un *identificador de NCC*. Este ID es un identificador CAN corriente, pero durante el modo de configuración se interpreta como un NCC ID. Esto permite enviar comandos de configuración a un componente en particular sin que sea interpretado por otro, es decir, direccionamiento *unicast*. Por otro lado, los identificadores CAN pueden ser usados por los nodos para sus propósitos durante el funcionamiento normal de la red.

El único NCC implementado actualmente es el módulo de inyección de fallos. Éste posee su propio conjunto de comandos de configuración. Si más tarde se añaden nuevos NCCs, sería necesario crear nuevos conjuntos de comandos de configuración para éstos.

La configuración de un NCC acaba con una *trama entra-modo-ocioso* o *trama entra-modo-espera*. La primera obliga al NCC a entrar en modo ocioso, mientras que la segunda le obliga a entrar en modo de espera. Un componente en modo ocioso ignora todas las tramas excepto la de entra-modo-configuración; en el modo de espera, ignora todas las tramas excepto la de entra-modo-configuración y la de *entra-modo-ejecución*. La trama de entra-modo-ejecución es transmitida por el PC a todos los nodos simultáneamente, indicando el principio del modo de ejecución. En el modo de ejecución los componentes empiezan a realizar su tarea, por ejemplo, un inyector de fallos inyecta fallos según su configuración.

La distinción entre el modo ocioso y el modo de espera permite deshabilitar NCCs en un test determinado.

III. Especificación de inyección de fallos

La configuración del módulo de inyección de fallos se realiza mediante la especificación de un fichero de texto: la *especificación de inyección de fallos*. El listado 1 muestra su definición en la forma de Backus-Naur (BNF) con la sintaxis de ISO/IEC 14977 [3]. La especificación contiene una serie de *configuraciones de inyección* etiquetadas, que son un conjunto de tuplas clave-valor, cada una de las cuales denominamos *parámetro de configuración*. Por brevedad, del BNF se omite el carácter de final de línea, que debe suceder cada parámetro de configuración, así como la definición de una cadena de caracteres o *string*, un número *natural* y un valor *boolean*, cuya definición es la corriente.

Los parámetros de configuración indican qué, dónde y cuando inyectar. Qué inyectar se define mediante un *valor de inyección*, por ejemplo, un valor dominante (`stuckDominant`) o una secuencia de bits (`bitFlip`, definiendo la secuencia

```
specification = { '[', string, ']', fi_config }
fi_config = value, link, time ;
value = 'value_type', '=', value_type_value,
        ['value_bfvalue', '=', {'0'|'1'} ] ;
link = 'link', '=', link_value - 'coupled' ;
time = mode, start_trig, start,
       end, [end_trig] ;
start_trig = 'start_trig_count', '=', natural,
     'start_trig_filter', '=', filter_value,
     'start_trig_field', '=', field_value,
     'start_trig_link', '=', link_value,
   ['start_trig_role', '=', role_value ] ;
start = 'start_field', '=', field_value,
        'start_bit', '=', natural,
        'start_offset', '=', natural ;
end = 'end_bc', '=', natural
    | 'end_field', '=', field_value,
      'end_bit', '=', natural ;
end_trig = 'end_trig_count', '=', natural,
     'end_trig_filter', '=', filter_value,
     'end_trig_field', '=', field_value,
     'end_trig_link', '=', link_value,
   ['end_trig_role', '=', role_value ] ;
mode = 'full' | 'iterative' | 'selective' ;
value_type_value = 'stuckDominant' | 'stuckRecessive' |
     'bitFlip' | 'inverse' ;
filter_value = ('0'|'1'|'x'), {'0'|'1'|'x'} ;
link_value = 'port0up' | 'port0dw' | 'port1up'
    | 'port1dw' | 'port2up' | 'port2dw'
    | 'port3up' | 'port3dw' | 'coupled' ;
field_value = 'idle' | 'id' | 'rtr' | 'res'
    | 'dlc' | 'data' | 'crc' | 'crcdelim'
    | 'ack' | 'ackdelim' | 'eof'
    | 'interfield' | 'errflag' | 'errdelim' ;
role_value = 'dont_care' | 'tr' | 're' ;
```

Listing 1. Especificación de la inyección de fallos en BNF.

con `value_bfvalue`). Dónde inyectar se define mediante un *enlace destino de inyección*, por ejemplo, el *downlink* del puerto 1 (`link = port1dw`). La especificación de cuándo inyectar es más compleja. Primero, el *trigger de inicio* indica la condición que debe satisfacerse antes de inyectar un fallo. Por ejemplo, un *trigger* de inicio puede ser la tercera recepción de un CRC (`start_trig_field = crc`, `start_trig_count = 3`) que empieza con el prefijo '0101' o '0111' (`start_trig_filter = 01x1`) y es detectado en el *downlink* del puerto 2 (`start_trig_link = port2dw`) cuando el nodo conectado al enlace destino de la inyección es el transmisor (`start_trig_role = tr`). La trama que satisface el *trigger* de inicio se denomina la *trama de inicio*. Seguidamente, es necesario especificar el conjunto de bits a inyectar. Para este propósito se definen dos grupos de tuplas clave-valor, denominados `start` y `end`. El primero especifica el primer bit en el que inyectar, mientras que el segundo indica la duración de la inyección. Esta duración se define como una cuenta de bits o una condición: el campo y bit que ya no debe ser inyectado. También es posible definir un *trigger de fin*. Un *trigger* de fin es una condición que debe satisfacerse para detener la inyección. Se define de la misma manera que un *trigger* de inicio, excepto por la posibilidad de omitirlo, lo que indica una inyección ilimitada, es decir, un fallo permanente. La trama que satisface el *trigger* de fin se denomina *trama de final*. Entre las tramas de inicio y fin pueden ser transmitidas varias tramas. El *modo de inyección de fallos* define cómo se inyecta en estas tramas. En el modo *full-range*, se inyecta en todos los bits desde `start`, en la trama de inicio, hasta `end`, en la trama de fin. En el modo *iterative* `start` y `end` indican el rango, dentro de cada trama entre la trama de inicio y fin, donde se llevará a cabo la inyección. El modo *selective* es igual

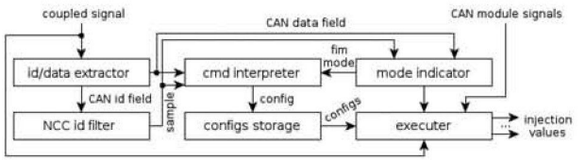

Fig. 2. Diagrama del módulo de inyección de fallos.

al iterativo, excepto que la inyección sólo se lleva a cabo en las tramas que satisfacen la condición del *trigger* de inicio.

No todas las especificaciones sintácticas correctas son válidas semánticamente. Por ejemplo, en el arbitraje (campo ID) la especificación de un rol es inválida, ya que éste solo se puede determinar al final del arbitraje.

Cada parámetro de configuración es codificado por el PC como una o varias instrucciones de configuración, cada una de los cuales se convierte en una trama CAN. Estas tramas poseen el NCC ID del módulo de inyección de fallos destino. Para configurar el módulo de inyección de fallos, el PC transmite la trama global de entra-modo-configuración para después hacer lo propio con las instrucciones de configuración. El módulo de inyección de fallos decodifica estas instrucciones y se auto-configura en consecuencia.

IV. Aspectos de implementación

Implementamos tres módulos en el *hub*: acoplamiento, CAN y inyección de fallos. El de acoplamiento y CAN se han reutilizado de CANcentrate[2], mientras que el módulo de inyección de fallos es nuevo. Se ha sintetizado todo en una FPGA Xilinx Spartan-3 XC3S1000.

La figura 2 muestra el interior del módulo de inyección de fallos. Contiene los siguientes submódulos. (I) El *ID/data extractor* extrae el campo ID y el campo de datos de la última trama transmitida. (II) El *NCC ID filter* comprueba si el ID anteriormente extraído concuerda con el NCC ID del módulo de inyección de fallos. En caso afirmativo, genera una señal que provoca que tanto el *mode indicator* como el *command interpreter* lean el campo de datos facilitado por el *ID/data extractor*. (III) El *mode indicator* comprueba si el campo de datos se corresponde con una instrucción de cambio de modo, por ejemplo, desde el modo de configuración al ocioso. En tal caso, señaliza el nuevo modo al *command interpreter* y al *executer*. (IV) El *command interpreter*, durante el modo de configuración, comprueba si el campo de datos recibido contiene una instrucción de configuración de un parámetro, en vez de uno de cambio de modo. A partir de todas estas instrucciones construye una configuración de inyección de fallos física, que es enviada al módulo *configurations storage* cuando la instrucción de final de configuración es recibido. (V) El *configurations storage* almacena todas las configuraciones de inyección de fallos y las hace disponibles para el módulo *executer*. (VI) El *executer* contiene un conjunto de *ejecutores de configuración programables*, que son módulos que, durante el modo de configuración, son programados según una de las configuraciones almacenadas. Durante el modo de ejecución estos módulos llevan a cabo, según su programación, la inyección de fallos.

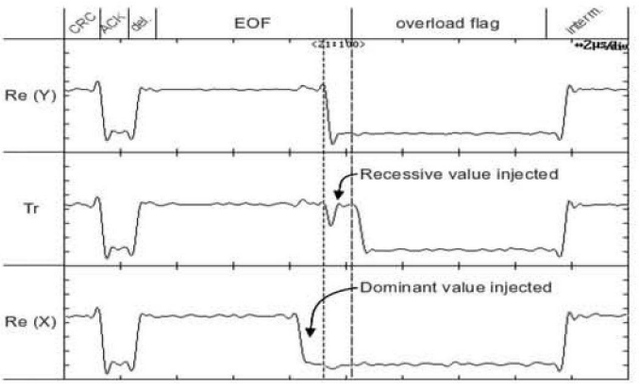

Fig. 3. Escenario IMO.

V. Ejemplo de uso

La figura 3 muestra una captura tomada con un osciloscopio digital Yokogawa DL7440. La imagen fue tomada mientras el inyector de fallos inyectaba los fallos que conducen a un escenario de inconsistencia por omisión de mensaje, descrito en [4]. Se ha insertado una anotación, a la izquierda de la captura, con el rol del nodo correspondiente a cada señal mostrada. Re(Y) y Re(X) son las señales del *downlink* de los nodos receptores Y y X, respectivamente; Tr es la señal del downlink del nodo transmisor. Además, en la parte superior, se muestran los campos de la trama CAN transmitida, vista por el nodo etiquetado como Y. Las marcas en el eje X no delimitan bits, sino intervalos de 2 μs. Las dos líneas verticales discontinuas en el centro de la imagen marcan los límites del último bit del campo *end of frame*.

Como se muestra en la captura, un bit dominante es inyectado en el penúltimo bit del *downlink* del nodo etiquetado como X, el cual, consecuentemente genera un *error flag*. El primer bit de este *error flag*, sin embargo, no es detectado por el transmisor, etiquetado como Tr, gracias a un bit recesivo inyectado en su *donwlink*. En el nodo etiquetado como Y no se inyecta ningún error, el cual, por tanto, detecta el primer bit del error flag. El comportamiento de los nodos se describe en [4]: el nodo X rechaza la trama, el nodo Y la acepta y el nodo Tr no la retransmite.

El listado 2 muestra la especificación usada para generar el escenario IMO. Sin embargo, debido al limitado espacio se obvia su descripción.

VI. Trabajo relacionado

En 2003, se propuso un inyector físico de fallos para CAN [5], afirmando ser el único capaz de generar escenarios de fallo complejos. En este artículo, sólo se consideran el citado inyector y un segundo, siendo los únicos inyectores físicos de fallos para CAN publicados entre 2003 y la fecha de escritura de este artículo.

El inyector de fallos presentado en [5] utiliza una herramienta *software* llamada CANfidant, y varios *inyectores de fallos individuales* (IFI), que son circuitos *hardware* insertados entre el *transceiver* y el controlador CAN de cada nodo. Este inyector de fallos requiere, a priori, un completo conocimiento de las tramas transmitidas en el bus. Conociendo el tráfico, CANfidant ayuda al ingeniero en el diseño del escenario de

error, simulando el comportamiento bit a bit y generando las instrucciones para los IFIs. Los IFIs, posteriormente, inyectan cada fallo contando las tramas y bits desde el principio del test. De este modo, si las tramas no se transmiten en el orden esperado, lo cual no es difícil en redes CAN dirigidas por eventos, la inyección se realiza incorrectamente. Por el contrario, sfiCAN, además de poder generar complejos escenarios de fallo, no requiere de un conocimiento previo del tráfico. No obstante, es posible usarlo junto con un simulador. Por ejemplo, es factible simular con CANfidant escenarios específicos, para posteriormente crear una especificación de inyección de fallos para sfiCAN.

En [6] se presenta un inyector de fallos que puede ser acoplado a una red CAN existente, como un nodo más. Aunque este acercamiento tiene la ventaja de no requerir una modificación de la red o de los *transceivers* de los nodos, proporciona una baja resolución espacial en la inyección, ya que éstas siempre afectan a toda red. Por lo tanto, este acercamiento no permite generar escenarios de inconsistencia como los descritos anteriormente. Además, las posibilidades de configuración son mucho más restringidas que en sfiCAN.

VII. Conclusiones y trabajo futuro

En este artículo se presenta sfiCAN: un inyector de fallos, configurable por red, basado en una topología de estrella, que permite la creación de escenarios de fallo complejos mediante la inyección de fallos físicos en una red CAN. El uso de la topología en estrella, con un hub central, cuyos enlaces están separados en *uplink* y *downlink*, permite que los fallos sean inyectados con una alta resolución espacial; mientras que el módulo CAN residente en el *hub*, que lleva el seguimiento del campo y bit de la trama transmitida, permite inyecciones de fallos con alta resolución temporal. Además, el hecho de poder almacenar y ejecutar múltiples configuraciones simultáneamente, en el módulo de inyección de fallos, y que éstos pueden ser especificados usando diferentes parámetros de configuración (como se ha ilustrado en el BNF del listado 1), hacen de sfiCAN un inyector muy flexible y potente. Finalmente, el impacto sobre el sistema de test es muy bajo: introduce un pequeño retardo, que puede ser tratado como un retardo adicional en la transmisión, y requiere la reserva de un identificador CAN durante la ejecución del test.

Se proponen dos usos para este inyector de fallos. Primero, comprobar como los controladores y el software de una red CAN responden ante fallos en el canal, en cuyo caso modificar la topología a una estrella no es relevante, ya que no es el canal lo que se comprueba. Segundo, comprobar topologías CAN en estrella —en particular los sistemas (Re)CANcentrate desarrollados bajo el proyecto CANbids [1].

Este artículo también aborda el uso de NCCs en CAN. En general, es una solución muy flexible, ya que permite desplegar componentes heterogéneos dentro de los nodos u otros dispositivos de la red CAN. El uso de diferentes NCC permite la construcción de una infraestructura de test automatizada para CAN.

[1] http://srv.uib.es/project/12

```
1  [fault injection 1]
2  value_type          = stuckDominant
3  link                = port1dw
4  start_trig_count    = 1
5  start_trig_filter   = 10110101010
6  start_trig_field    = id
7  start_trig_link     = coupled
8  start_trig_role     = re
9  start_field         = eof
10 start_bit           = 5
11 start_offset        = 0
12 end_bc              = 1
13 mode                = full
14 [fault injection 2]
15 value_type          = stuckRecessive
16 link                = port0dw
17 start_trig_count    = 1
18 start_trig_filter   = 10110101010
19 start_trig_field    = id
20 start_trig_link     = coupled
21 start_trig_role     = tr
22 start_field         = eof
23 start_bit           = 6
24 start_offset        = 0
25 end_bc              = 1
26 mode                = full
```

Listing 2. Ejemplo especificación de una inyección de fallos

Un NCC relevante que se planea añadir al *hub* es el *logging module*. Su propósito es capturar información valiosa durante la fase de ejecución de un test, para después transmitir estos datos en la fase de configuración, cuando se le requiera. El hecho de situar este NCC dentro del *hub* le otorga una visión privilegiada del sistema, lo que le permite conocer qué ha sido transmitido hacia y desde cada nodo, así como información del estado interno de otros módulos residentes dentro del *hub*. Por otro lado, la transmisión de todos esos datos al PC permitirá su correcto análisis.

Reconocimiento

Este trabajo ha recibido el apoyo del Ministerio Español de Ciencia e Innovación con la subvención DPI2008-02195 y financiación de FEDER, además de la portuguesa Fundação para Ciência e a Tecnologia con la subvención SFRH/BP-D/70317/2010.

References

[1] Robert Bosch GmbH, "CAN Specification Version 2.0," 1991. [Online]. Available: http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[2] M. Barranco, J. Proenza, G. Rodríguez-Navas, and L. Almeida, "An Active Star Topology for Improving Fault Confinement in CAN Networks," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 2, pp. 78–85, May 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1377714

[3] "International Standard ISO/IEC 14977 - Information technology - Syntactic metalanguage - Extended BNF," 1996.

[4] J. Proenza and J. Miro-Julia, "MajorCAN: A Modification to the Controller Area Network Protocol to Achieve Atomic Broadcast," *IEEE International Workshop on Group Communication and Computations, Taipei, Taiwan*, 2000.

[5] G. Rodríguez-Navas, J. Jiménez, and J. Proenza, "An architecture for physical injection of complex fault scenarios in CAN networks," in *Proc. 9th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, vol. 2, 2003. [Online]. Available: /home/david/Documents/bibliography/papers/arch-for-phys-fault-inj-in-CAN.pdf

[6] M. S. Reorda and M. Violante, "On-line analysis and perturbation of CAN networks," in *Proc. 19th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1347867