Analizando Especificaciones Formales de Requisitos de Software usando un Model Checker *Off-the-Shelf*

Gastón Scilingo*, María Marta Novaira*, Renzo Degiovanni*†, y Nazareno Aguirre*†
*Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Argentina
†Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
{gaston, mnovaira, rdegiovanni, naguirre}@dc.exa.unrc.edu.ar

***Resumen*—La calidad de las especificaciones de requisitos tienen en general un enorme impacto en todo el proceso de desarrollo, pues gran parte de las actividades de validación y verificación dependen de estas especificaciones. Entre los enfoques para la especificación de requisitos, los basados en técnicas formales suelen ser más adecuados para realizar actividades de análisis automático. El método SCR (Software Cost Reduction), en particular, se basa en una notación formal tabular para organizar requisitos. Ofrece además una familia de herramientas poderosas de análisis, conocida como SCR Toolset, que permiten generar escenarios de ejecución, producir tests, verificar propiedades temporales, analizar la consistencia de la especificación, etc., tanto automáticamente como de manera asistida, usando una variedad de técnicas, como el model checking y la demostración automática. Sin embargo, las herramientas de análisis detrás de SCR no son de libre acceso, con lo cual la adopción de SCR como método formal de especificación se ve seriamente limitada, a pesar del éxito que ha demostrado tener en la especificación de requisitos de sistemas críticos.**

En este artículo, estudiamos la utilización de una herramienta convencional *off-the-shelf* de verificación formal, más precisamente un model checker de estados explícitos, para realizar diferentes análisis vinculados a una especificación SCR. A diferencia de otros trabajos, en los cuales se usa model checking para la verificación de propiedades temporales o para la generación automática de tests, en este artículo estudiamos la utilización de un model checker off-the-shelf para diversas actividades adicionales de análisis, en particular para el análisis de consistencia vinculado al método SCR. Además, y con el objetivo de evaluar las capacidades del model checker y nuestra caracterización de las diversas actividades de análisis, desarrollamos un caso de estudio basado en la captura formal de requisitos del sistema de control de un marcapasos, más complejo que los que se suelen analizar en la literatura de SCR, que en particular nos permite evaluar la escalabilidad del análisis.

I. Introducción

Es ampliamente aceptado que la depuración es una de las etapas más costosas en el desarrollo de software, y que los errores son más fáciles (y menos costosos) de corregir si se capturan lo más temprano posible en el proceso de desarrollo [8], [13], [17]. Luego, el descubrir inconsistencias, errores de comprensión e imprecisiones durante la captura de requisitos es de fundamental importancia práctica y económica en la actividad de producción de software. Debido a esto, la calidad de las especificaciones de requisitos tienen un enorme impacto en todo el proceso de desarrollo, en particular debido a que gran parte de las actividades de validación (contrastación del sistema solicitado por el usuario con el que describe la especificación) y verificación (contrastación del sistema especificado con el construido) dependen de estas especificaciones. Existe una amplia variedad de notaciones para la descripción de requisitos de software, pero aquellas con una semántica formal son particularmente apropiadas para el análisis, ya que las ambigüedades propias de la notación se eliminan absolutamente. Sin embargo, una semántica formal no es suficiente: las complejidades y sutilezas de los requisitos, y el grado de detalle propio de las notaciones formales demandan formas de modularizar especificaciones, de manera de colaborar con los desarrolladores en la comprensión y la manipulación para el análisis. Esto es crucial para que una notación formal sea prácticamente utilizable.

Una de estas notaciones es la del método SCR. El mismo está basado en la notación tabular de Parnas [6], y consiste en organizar especificaciones de requisitos en forma de tablas en una notación precisa, con el objetivo de conseguir modularidad y adecuación al análisis. Además, en la literatura se han aplicado exitosamente diversos tipos de análisis formales a especificaciones SCR, muchos de ellos asistidos por herramientas de software, que incrementan significativamente la capacidad de detección de errores en los requerimientos. Las herramientas de análisis para SCR se concentran en el denominado SCR Toolset, que permite generar escenarios de ejecución, producir tests, verificar propiedades temporales, analizar la consistencia de la especificación, etc., tanto automáticamente como de manera asistida, usando una variedad de técnicas, como el model checking y la demostración automática. Sin embargo, las herramientas de análisis detrás de SCR Toolset no son de libre acceso dado que pertenecen al Naval Reseach Laboratory de los Estados Unidos, con lo cual la adopción de SCR como método formal de especificación se ve seriamente limitada, a pesar del éxito que ha demostrado tener en la especificación de requisitos de sistemas críticos.

En este artículo, estudiaremos cómo podemos aprovechar una herramienta convencional *off-the-shelf* de verificación

formal, para analizar automáticamente diferentes aspectos de una especificación SCR. El objetivo es claro: evaluar cuan efectiva puede ser una herramienta estándar y en cierto sentido de propósito "más general" en el análisis de especificaciones de requisitos, y cuáles son las mayores dificultades con las cuales puede encontrarse un desarrollador en esta tarea. La herramienta a utilizar es el model checker de estados explícitos SPIN. A diferencia de otros trabajos que utilizan SPIN para analizar especificaciones SCR , en este caso estudiamos la utilización de SPIN para todas las actividades de análisis, en particular análisis de consistencia vinculado a SCR, a diferencia de otros en los cuales se usa model checking para la verificación de propiedades temporales o para la generación de tests [7].

Con el objetivo de evaluar las capacidades del model checker y nuestra caracterización de las diversas actividades de análisis, desarrollamos un caso de estudio basado en la captura formal de requisitos del sistema de control de un marcapasos, que es más complejo que los que se suelen analizar en la literatura de SCR. Esto nos permitirá, en particular, evaluar la escalabilidad de los diversos análisis. La elección de este caso de estudio fue motivada por un desafío a los métodos formales del Software Quality Research Laboratory de la Universidad McMaster [1]. Este desafío propone el análisis de un marcapasos por dos razones principales: es un sistema crítico del cual dependen las vidas de sus usuarios, y la complejidad del software del mismo. Propondremos una especificación en SCR de este caso de estudio no trivial, y analizaremos todos los aspectos posibles de la especificación usando el model checker SPIN, un verificador temporal automático de libre distribución. Como veremos, esto requerirá capturar la semántica y condiciones de consistencia propias de la notación SCR en la notación aceptada por SPIN para la verificación. Buscaremos por supuesto descubrir inconsistencias en la especificación, problemas de completitud o casos no contemplados, y demás tareas de validación. Cabe aclarar que todas estas tareas serán realizadas sobre una especificación producida por los autores del artículo para la evaluación (no un caso de estudio de probada consistencia o corrección), con lo cual buscamos descubrir errores propios de nuestra interpretación del problema, o de nuestra actividad de captura de requisitos.

El artículo está organizado de la siguiente manera. En la Sección 2 se detalla el funcionamiento y la organización de un marcapasos. En la Sección 3 se describe brevemente el método SCR. En 4 se introducen las partes principales de la notación SCR, utilizando como ejemplo la especificación SCR del sistema descripto en la sección 2. Más adelante, en 5, se describe los tipos de análisis aplicados sobre el modelo, la forma en que los mismos fueron capturados para poder ser analizados automáticamente (i.e., el procedimiento aplicado para la traducción de la especificación para su análisis). En la Sección 6 vemos cómo los distintos análisis llevados a cabo contribuyeron a la detección de errores en los requerimentos. Por último, discutimos sobre las lecciones aprendidas, las mayores dificultades encontradas, nuestras conclusiones y propuestas de trabajo futuro (Sección 7).

II. Un Sistema de Marcapasos

En esta sección, y con el objetivo de hacer este artículo lo más autocontenido posible, describimos brevemente en qué consiste un sistema de marcapasos. Un sistema de marcapasos, al menos en lo que respecta a la captura y análisis de requisitos realizados en este artículo, tomando la descripción de [1], consta de tres componentes: *(i)* un dispositivo electrónico diseñado para producir impulsos eléctricos, llamado generador de pulsos (PG), cuya finalidad es la estimulación del corazón cuando falla la estimulación fisiológica o normal; *(ii)* una estación Controlador-Monitor (DCM), para programar y consultar al PG; y *(iii)* un set de cables conductores que llevan los pulsos del PG al corazón y las señales sensadas en el corazón al PG.

El DCM consiste de una plataforma hardware y una aplicación de software que se comunican con el PG usando un protocolo de comunicación específico, por telemetría bidireccional. Esto le permite al médico o al técnico programar, consultar (por ejemplo, medir el nivel de batería, obtener histogramas de frecuencia, etc.) y enviar comandos al dispositivo de forma no invasiva después de la implantación.

El PG monitorea y regula la frecuencia del corazón del paciente; detecta y provee terapia para condiciones de bradicardia[1]; puede ser programado para proporcionar una frecuencia adaptativa de ritmo, de simple y doble cámara (aurícula y ventrícula), tanto permanente como temporaria.

Los valores nominales y la configuración de valores de operación mínima del dispositivo quedan establecidos en su fabricación. Además, el dispositivo cuenta con una memoria flash para registrar datos históricos (de su operación y eventos sensados), y para almacenar la configuración de valores que establece y determina el médico para proveer terapia al paciente durante el período ambulatorio.

El PG puede sensar y/o estimular la cámara auricular, la ventricular o ambas con pulsos eléctricos (de voltaje y ancho programable), según el modo de operación de bradicardia que le sea programado. Un modo de operación indica básicamente lo siguiente: qué cámara/s estimular, qué cámara/s sensar, y cómo responder ante la detección de un pulso espontáneo o provocado en una cámara. El modelo de marcapasos utilizado en el presente trabajo permite 19 modos de operación de bradicardia en estado permanente, a saber: `off`, `DDD`, `VDD`, `DDI`, `DOO`, `VOO`, `AOO`, `VVI`, `AAI`, `VVT`, `AAT`, `DDDR`, `VDDR`, `DDIR`, `DOOR`, `VOOR`, `AOOR`, `VVIR`, `AAIR`. La primera letra indica qué cámara debe ser estimulada; la segunda qué cámara debe ser sensada;

[1]La bradicardia es una alteración del ritmo al que late el corazón. Concretamente se trata de un descenso en la frecuencia cardíaca.

la tercera en qué forma se debe responder ante un pulso sensado; y la cuarta, en caso de estar presente, indica si está habilitada la modulación de frecuencia. Por ejemplo, el modo de operación `VVI` indica que se debe pulsar y sensar la ventrícula, e inhibir un pulso programado si se sensa un pulso espontáneo previamente.

El PG puede encontrarse en cinco estados de funcionamiento distintos. El estado normal o permanente (`Normal`) es el estado en el cual el dispositivo provee al paciente la terapia programada por el médico; estimula, sensa y responde según el modo de operación de bradicardia y el conjunto de valores seteados para el resto de las variables, como por ejemplo ancho y voltaje del pulso. El estado temporal (`Temporal`) es utilizado para setear valores en el dispositivo, testear parámetros o proveer pruebas de diagnóstico. También existe el modo de operación de bradicardia para este estado, y de éste se deberá salir inmediatamente por una rotura en el enlace de telemetría, por un comando `CANCEL` o `PN` (Pace-Now) proveniente del DCM. El estado Pace-Now (`PaceNow`) es un estado de estimulación de emergencia, del cual no se sale hasta que no se reciba un cambio en el modo de estimulación desde el DCM. El estado magnético es un estado de funcionamiento provocado por la presencia de un buñuelo magnético[2] en una distancia menor a 2,5 cm del dispositivo; este estado es utilizado para determinar la salud de la batería y entrar en este estado provoca cambios en el modo de operación de bradicardia. El modo de operación anterior debe ser restituido al salir del estado magnético. Es posible además programar el PG para ignorar la detección del buñuelo. Finalmente, el dispositivo puede encontrarse en estado Power on Reset (`POR`): se deberá entrar en este estado cuando el nivel de voltaje de la batería se encuentre por debajo de un cierto umbral (`BatteryLevel`) que hace impredecible el comportamiento del PG; en tal caso, todas las funcionalidades se inhabilitan hasta que el nivel de voltaje exceda el umbral, manteniendo una actividad mínima para optimizar el uso de la batería y garantizar un mínimo de estimulación al paciente.

III. Especificaciones Tabulares: el Método SCR.

El método SCR [6] utiliza una notación formal para especificar requerimientos de software de manera concisa mediante tablas, utilizada principalmente en la práctica para la captura de requisitos de sistemas críticos. En la metodología SCR, las tablas son usadas para describir la relación que el sistema debe inducir entre variables *monitoreadas* y *controladas*. Para describir de forma organizada y modular esta relación, SCR hace uso de *eventos*, *condiciones*, *clase de modo* y *términos*.

Una clase de modo (*mode class* en inglés) es una partición de los estados del sistema. Cada clase de equivalencia en la partición es un modo del sistema. Un término es una función sobre las variables de la especificación. Una condición es una expresión lógica que hace referencia a las variables. Un evento ocurre cuando el valor de cualquier variable del sistema cambia (una variable del sistema es: una variable monitoreada o controlada, una clase de modo o un término). Un evento puede o no tener condiciones. La notación `@T(c) WHEN d` denota un evento condicionado definido como:

$$\texttt{@T(c) WHEN d} \equiv \neg\ \texttt{c} \wedge \texttt{c}' \wedge \texttt{d}$$

donde las condiciones no primadas `c` y `d` son evaluadas en el estado previo y la condición primada `c'` está evaluada en el nuevo estado, al que se arriba como consecuencia del evento. Informalmente, esto denota el siguiente evento:

> el predicado `c` se hace verdadero en el nuevo estado al mismo tiempo que el predicado `d` se cumple en el estado previo.

La notación `@F(c)` denota el evento `@T(¬ c)`.

Durante la operación del sistema, el ambiente cambia una variable monitoreada, causando un evento de entrada. En respuesta, el sistema actualiza los términos y las clases de modo, y cambia las variables controladas, de la forma en que se indica en las tablas. En otras palabras, las tablas definen términos, clases de modo y variables controladas.

El comportamiento del sistema se describe usando tres tipos de tablas: de condición, de eventos y de transición de modos. Las tablas permiten describir una gran cantidad de requisitos de manera concisa, bien organizada y de forma modular. Cada tabla define una función (aunque las especificaciones SCR pueden ser no deterministas). La tabla de condición describe variables de salida (controladas) o términos en función de modo y condición. Una tabla de eventos describe ambos (variables de salida y términos) como función de modos y eventos. La tabla de transición de modos describe modos como función de otros modos y eventos. Es decir, una tabla de transición de modos es una tabla de eventos que define una clase de modo. Las tablas de condición definen funciones totales, mientras que las tablas de eventos y de transición de modos pueden definir funciones parciales, ya que algunos eventos pueden no ocurrir cuando cierta condición es verdadera.

Utilizaremos el prefijo “m” para indicar los nombres de las variables monitoreadas, y el prefijo “c” para las variables controladas. El tipo de una variable monitoreada indica el rango de valores que pueden ser asignados a las variables.

IV. Captura de Requisitos del Marcapasos mediante SCR

En esta sección introducimos parte de nuestra especificación SCR del sistema de control del marcapasos descrito previamente. Mostraremos sólo una porción de la especificación completa, que será suficiente para ilustrar la forma en que haremos análisis, y los resultados obtenidos. También introduciremos algunos aspectos relevantes del

[2]El buñuelo magnético es un dispositivo imantado que se utiliza para medir el nivel de voltaje en baterías de marcapasos.

comportamiento esperado del dispositivo, representado en las construcciones mostradas.

A. Detección de Clases de Modo

Como se mencionó en la sección 2, el dispositivo PG puede estar en cinco estados de funcionamiento distintos. Estos estados de funcionamiento pueden ser capturados naturalmente como modos de operación, de la única clase de modo que tendremmos en la especificación, y que llamaremos `mcPulseCondition`. Cabe hacer notar que para satisfacer el requerimiento de que cada vez que se abandone el estado magnético se vuelva al modo de operación previo al test, es necesario modelar el estado magnético como cuatro modos distintos (`MAGnormal, MAGpacenow, MAGpor, MAGtemporal`). La Tabla 1 muestra un fragmento de la tabla de transición de modos. Esta tabla indica, por ejemplo, que si se está en modo Normal y el nivel de voltaje de la batería `mBATTERYvoltage` se hace menor que `BatteryLevel`, entonces el sistema entra en modo POR. Nótese que, para que esta tabla sea correcta (como mostraremos más adelante), cada par de filas distintas correspondiente a un mismo modo de salida debe ser disjuntas, en el sentido que los eventos correspondientes no pueden tener solapamiento. De lo contrario, tendríamos una contradicción, en algún sentido, pues los requisitos pedirían que se realicen dos cosas distintas en una misma situación. Esta será una de las propiedades a verificar de la especificación.

B. Valores monitoreados y valores controlados

El PG debe controlar qué cámaras se estimulan, cuáles se sensan y cómo se responde ante una detección de pulso, de acuerdo: al programa configurado por el médico, como respuesta a comandos recibidos por telemetría desde el DCM, o bien por otros factores como una caída en el nivel de batería o la presencia de un buñuelo magnético. Estos hechos, que representan cambios ambientales para el sistema de control del PG, se modelan como variables monitoreadas. El hecho de que el médico pueda programar un modo de operación de bradicardia para el estado `Normal` lo hemos modelado a través de una variable monitoreada `mMODEbrad`, cuyo rango de valores fue presentado en la sección 2. Los comandos recibidos vía telemetría los hemos modelado con la variable `mCommand`. El nivel de voltaje medido en la batería del PG es representado por `mBATTERYvoltage`. Finalmente, la presencia de un imán para test magnético (a menos de 2,5 cm del dispositivo) está indicada por el término `tMagnetON`. Para hacer más concisas las tablas y facilitar la escritura de propiedades hemos definido varios términos. Por ejemplo, en la Tabla 2 se muestra la tabla de condición de `tModebradPV`, que indica si el modo de operación de bradicardia está programado en algún programa de pulsado de cámara ventricular.

OLD MODE	EVENT	NEW MODE
Normal	@T(mBATTERYvoltage < BatteryLevel)	POR
Normal	@T(tMagnetON) when mMagnet=ON	MAGnormal
Normal	@T(mCommand=TMP)	Temporal
Temporal	@T(mBATTERYvoltage < BatteryLevel)	POR
Temporal	@T(tMagnetON) when mMagnet =ON	MAGtemporal
Temporal	@F(tTELEMETRYsignal)	Normal
PaceNow	@T(mBATTERYvoltage < BatteryLevel)	POR
PaceNow	@T(mCommand=NORMAL)	Normal
POR	@T(tMagnetON) when mMagnet=ON	MAGpor
POR	@T(mCommand=PN) when mBATTERYvoltage ≥ BatteryLevel	PaceNow
POR	@T(mCommand=NORMAL) when mBATTERYvoltage ≥ BatteryLevel	Normal
POR	@T(mCommand=TMP) when mBATTERYvoltage ≥ BatteryLevel	Temporal
.....	.....	.....
MAGpacenow	@F(tMagnetON)	PaceNow
MAGpor	@F(tMagnetON)	POR

Table I: Fragmento de la tabla de transición de modos `mcPulseCondition`.

La variable controlada `cCHAMBERSpaced` modela cambios controlados por el sistema, la forma de pulsado de las cámaras. Si el PG está en estado de operación `Normal` la forma de pulsado será acorde al programa configurado por el médico (por ejemplo, `cCHAMBERSpaced` tendrá el valor `A` si el programa elegido es `AAI`). Pero ante un evento como la señal de estimulación de emergencia (`PN`) o la caída en el nivel de batería, el sistema debe cambiar la forma de pulsado a pulsado de sólo ventrículo (`V`). En la Tabla 3 se muestra un fragmento de la tabla de eventos que define la variable antes mencionada.

	mMODEbrad = VDD or mMODEbrad = VDDR or mMODEbrad = VOOR or mMODEbrad = VVIR or mMODEbrad = VOO or mMODEbrad = VVI or mMODEbrad = VVT	NOT (mMODEbrad = VDD or mMODEbrad = VDDR or mMODEbrad = VOOR or mMODEbrad = VVIR or mMODEbrad = VOO or mMODEbrad = VVI or mMODEbrad = VVT)
tMODEbradPV	true	false

Table II: Tabla de condición para el término `tMODEbradPV`

V. Verificación de Especificaciones de Requisitos SCR

Debido a que SCR posee una semántica formal bien definida, ésta se puede aprovechar para realizar análisis de propiedades de las especificaciones, tales como ausencia de contradicciones, entre otras. De hecho, se han desarrollado varias técnicas de análisis automático aplicadas a este tipo de especificaciones. En particular, existe la herramienta SCR Toolset [11] que da soporte a varias de estas técnicas. Esta herramienta, o conjunto de herramientas para ser precisos, no está públicamente disponible dado que pertenece al laboratorio Naval de los Estados Unidos. La motivación

Normal	@T(mCOMMAND = TMP)	@T(tMagnetON) When (mMagnet = ON and tMODEbradPA)	@T(mBATTERYvoltage < BatteryLevel) or @T(mCOMMAND = PN) or (@T(tMagnetON) when (mMagnet = ON and tMODEbradPV))	@T(tMagnetON) When (mMagnet = ON and tMODEbradPD)
Temporal	...	...	...	...
PaceNow	@T(mCommand = TMP)	@T(mCommand = NORMAL) when tMODEbradPA	@T(mCommand = NORMAL) when tMODEbradPV	@T(mCommand = NORMAL) when tMODEbradPD
POR	@T(mCommand = TMP) when mBATTERYvoltage $\geq$ BatteryLevel	@T(mCommand = NORMAL) when (mBATTERYvoltage $\geq$ BatteryLevel) and tMODEbradPA	(@T(tMagnetON) when mMagnet = ON) or (@T(mCommand = PN) when mBATTERYvoltage$\geq$ BatteryLevel) or (@T(mCommand = NORMAL) when (mBATTERYvoltage$\geq$ BatteryLevel) and tMODEbradPV)	@T(mCommand = NORMAL) when (mBATTERYvoltage$\geq$ BatteryLevel) and tMODEbradPD
MAGnormal	Never	@F(TMagnetON) when tMODEbradPA	@F(TMagnetON) when tMODEbradPV	@F(TMagnetON) when tMODEbradPD
MAGtemporal	@F(tMagnetON)	Never	Never	Never
MAGpacenow	Never	Never	@F(tMagnetON)	Never
MAGpor	Never	Never	@F(tMagnetON)	Never
cCHAMBERSpaced	O	A	V	D

Table III: Fragmento de la tabla de Eventos para cCHAMBERSpaced.

de este trabajo es justamente estudiar la factibilidad de utilizar una herramienta de verificación off-the-shelf, ante la ausencia de contar con la familia de herramientas específicas, especialmente diseñadas y ajustadas para este método. Como veremos más adelante, usaremos el model checker de estados explícitos SPIN, una herramienta muy madura y de amplia adopción (además de ser de código abierto y de libre disponibilidad), para realizar los análisis necesarios.

Los tipos de análisis considerados en este trabajo son: análisis estructurales, tales como completitud de las tablas de condición, disjunción de las filas de las tablas, eliminación de posibles redundancias, alcanzabilidad de modos y verificación de propiedades.

Existen además otros tipos de análisis, con los cuales no trabajaremos en este artículo, tales como simulación de ejecuciones del sistema [11], generación de casos de test [4], [7] (cuyo tratamiento usando SPIN ha sido tratado previamente), y la generación automática de invariantes [14], etc.

A. Disjunción - Determinismo en la defición de las tablas

En general, el método SCR es aplicado a sistemas reactivos. El ambiente en el que se ejecutan estos sistemas es no-determinista, pero el comportamiento capturado por cada transición de dicho sistema debe ser *determinista* (básicamente, el no determinismo estará dado por la posibilidad de tener varias formas de avance en el sistema habilitadas en un momento dado, pero cada una de ellas tiene un efecto determinista en el estado del sistema).

Tal como se muestra en [9], para chequear que el comportamiento que define una tabla es funcional (o determinista), debemos corroborar que las condiciones que forman cada fila sean disjuntas. A este tipo de chequeo se lo conoce con el nombre de *disjointness* y es uno de los principales análisis estructurales en este tipo de especificaciones.

Dos filas distintas f_1 y f_2 cualesquiera pertenecientes a una tabla son disjuntas, si ocurre que $f_1 \wedge f_2 = false$. Es decir, si la intersección entre las dos filas es vacía, o expresado de manera similar, si ambas filas son disjuntas. Este tipo de chequeo requiere, usualmente, algún proceso de decisión para la lógica utilizada en las tablas (e.g., SMT solving, SAT solving), o una teoría (no decidible) capturada en algún demostrador de teoremas. Para comprobar disjointness, se utiliza entonces SMT, SAT o algún otro mecanismo de constraint solving, o alternativamente se demuestra deductivamente la disjunción.

En nuestro caso, debemos tratar de analizar disjointness con SPIN. SPIN no cuenta con constraint solvers subyacentes, con lo cual debemos capturar disjointness como alguna propiedad chequeable con el model checker. Notemos que si bien $f_1 \wedge f_2 = false$ es lo que efectivamente queremos verificar, lo que nos interesa es que *no sea alcanzable* una situación que falsifique la condición anterior. Es decir, que partiendo del estado inicial, *no* podemos alcanzar un estado en el cual $f_1 \wedge f_2 \neq false$. Esta propiedad es más débil que disjointness, pero equivalente para nuestros propósitos. Lo que estaremos verificando es disjointness de

f_1 y f_2 en los estados alcanzables de la especificación (no su disjointness absoluta). Suponiendo que podemos caracterizar f_1 y f_2 con propiedades de programa (lo veremos más adelante), disjointness será verificada chequeando la siguiente propiedad LTL:

$$\Box\neg(f_1 \wedge f_2)$$

B. Completitud de Tablas de Condiciones

Una propiedad importante de las tablas de condiciones es que la disyunción de todas las condiciones de una de estas tablas, para una misma fila, abarcan todo el espectro de posibilidades (técnicamente, la disyunción de las diferentes celdas de una misma fila equivalen a `true`). Nuevamente, para chequear este tipo de propiedades requeriríamos un constraint solver. Al igual que para el caso anterior, lo comprobaremos a través de propiedades de alcanzabilidad: comprobaremos si el sistema admite la alcanzabilidad de un estado en el cual no se dé ninguna de las condiciones de las celdas de una misma fila en una tabla (mostrando así su incompletitud). Esto puede expresarse para toda fila k con la siguiente propiedad LTL:

$$\Diamond\neg(C_1k \vee C_2k... \vee C_nk)$$

C. Alcanzabilidad de modos

Otro chequeo de consistencia de una especificación SCR es la alcanzabilidad de modos. De acuerdo a la semántica de SCR, todo modo en una especificación debe ser no redundante (necesario). La comprobación de esta propiedad también puede hacerse a través de chequeos de alcanzabilidad, uno por cada modo de la especificación. Estaremos comprobando entonces, mediante SPIN, que para toda clase de modo $\mathcal{M}$ y todo modo $m \in \mathcal{M}$, existe al menos una ejecución del sistema tal que, comenzando desde el estado inicial, visite algún estado cuyo modo coincida con m.

$$\Diamond(\texttt{mcPulseCondition} = m)$$

D. Cobertura de tablas - Eliminación de redundancia

Debido a la complejidad que pueden alcanzar las especificaciones SCR, dependiendo del sistema en cuestión, puede ocurrir que existan filas en las tablas que sean inconsistentes e inalcanzables. Es un análisis similar al de alcanzabilidad de modos, aunque más complejo.

Entendemos por filas *inconsistentes* a aquellas para las cuales no puede existir un estado que las satisfaga. De otra manera, pueden existir filas consistentes pero *inalcanzables* en el comportamiento del sistema. Es decir, no existe una ejecución, comenzando desde el estado inicial, de manera tal que alguno de los estados visitados satisfaga la condición de la fila. Es evidente que toda fila inconsistente es además inalcanzable.

La detección de filas inalcanzables pone en evidencia que la especificación contiene redundancias. Por lo que se debe de localizar el problema y eliminar esa redundancia en la definición de la tabla.

Es posible utilizar técnicas de generación de casos de tests, para lograr verificar si nuestra especificación posee o no filas inalcanzables. Tal como se muestra en [7], si la especificación cumple con el criterio de cobertura *Table Coverage*, entonces para cada fila existe un caso de test (una ejecución) que ejercita esa fila, es decir, obtiene un estado que satisface las condiciones de la fila.

E. Verificación de propiedades Temporales

Se han presentado en la literatura diversas formas de verificar propiedades sobre especificaciones SCR [11]. Entre éstas podemos mencionar el model checker de estado explícito SPIN [10], ALV (un model checker simbólico) [5], SALSA (un constraint solver basado en BDDs) [3], TAME (una interfaz para el demostrador de teoremas PVS [2] adaptada a especificaciones tabulares). Muchas de las herramientas mencionadas pertenecen al SCR Toolset, por lo que no pueden ser usadas para propósitos públicos.

En este artículo nos centramos en la verificación de un tipo particular de propiedades temporal, las conocidas como *invariantes* (propiedades de safety). En particular, un invariante puede ser de *estado*, si vale en todo estado alcanzable $s \in S$, o de *transición* si vale en todo par de estados alcanzables (s, s'), donde S es el conjunto de estados del sistema, $s, s' \in S$ y existe un evento e habilitado en s y cuya ejecución a partir de s obtiene s'. La elección de analizar invariantes del sistema se debe al tipo de propiedades identificadas desde la descripción informal.

F. SPIN como herramienta de análisis

SPIN [12] es la herramienta elegida para realizar análisis, como hemos mencionado anteriormente, debido a varias características, en particular su aplicación previa a ciertos análisis de especificaciones SCR [7], [10], [11] (aunque no todos los tratados en este artículo). SPIN es un model checker de estado explícitos que utiliza exploración de estados para verificar propiedades. La descripción del sistema debe ser realizado en el lenguaje Promela [12] y las propiedades pueden codificarse con la sentencia `assert` o fórmulas en lógica temporal lineal [15].

Considere la siguiente porción de la tabla de modos especificada en la Fig.1:

OLD MODE	EVENT	NEW MODE
Normal	@T(mBATTERYvoltage < BatteryLevel)	POR
Normal	@T(tMagnetON) when mMagnet=ON	MAGnormal
Temporal	@T(mBATTERYvoltage < BatteryLevel)	POR
...	...	...

Bharadwaj y Heitmeyer introdujeron en [10] una traducción de especificaciones tabulares SCR al lenguaje Promela. A continuación mostramos cuál sería la codificación en Promela de la porción de la tabla de modos mostrada anteriormente.

```
/*  definicion de tabla de modo mcPulseCondition */
if
:: mcPulseCondition == Normal ->
  if
  :: (mBATERYvoltageP < BatteryLevel) &&
     (!(mBATERYvoltage < BatteryLevel))
                          -> mcPulseConditionP = POR;
  :: (tMagnetONP && ! tMagnetON) && (mMagnet == On)
                        -> mcPulseConditionP = MAGnormal;
  ::else skip;
  fi;
:: mcPulseCondition == Temporal ->
  if
  :: (mBATERYvoltageP < BatteryLevel) &&
     (!(mBATERYvoltage < BatteryLevel))
                          -> mcPulseConditionP = POR;
  :: ...
  fi;
  ...
fi;
```

Se puede observar que por cada variable SCR, tenemos dos variables en Promela para modelar el estado previo y el siguiente. Por ejemplo, `mcPulseCondition` y `mcPulseConditionP` modelan el modo previo y el modo siguiente, respectivamente. De esta manera podemos expresar facilmente invariantes de transición utilizando la sentencia `assert`. Consideremos por ejemplo la siguiente aserción:

```
assert (!( !(mBATERYvoltage < BatteryLevel)
       && (mBATERYvoltageP < BatteryLevel))
       || (mcPulseConditionP == POR));
```

El invariante especificado por esta aserción indica que si el votaje de la batería se hace menor al nivel permitido, el nuevo modo debería ser `POR`.

VI. Resultados Experimentales

En esta sección mostraremos los resultados experimentales de aplicar los análisis descriptos en la sección anterior, a la especificación SCR del sistema de control de un marcapasos, presentado anteriormente en este artículo. De la especificación original expresada en tablas SCR (en un documento de textos) pasamos manualmente a una descripción en el lenguaje SAL (SCR Abstract Language), para luego ser traducida a Promela (ver subsección 5.6), y así poder utilizar SPIN como herramienta de análisis.

El análisis de *alcanzabilidad* de modos resultó exitoso. SPIN logró encontrar una ejecución que cubra cada modo en pocos segundos. Los chequeos de *disjunción* y de *cobertura* fueron aplicados a las tablas de eventos que definen a las variables controladas `cCHAMBERSpaced`, `cCHAMBERSsensed` y `cRESPONSEsensing`, y a la tabla de modos que define la variable `mcPulseCondition`. Los experimentos develaron un error de disjunción de filas en la tabla de `cRESPONSEsensing`. Este error tenía que ver con una falla a la hora de transcribir la especificación en SAL. Más precisamente, faltába incluir el término (`tMODEbradSA`) en la definición de la tabla, el cual debe ser mencionado para asegurar que el sistema de transiciones definido sea determinista.

Además de las tareas arriba descriptas, es sumamente importante poder analizar si la especificación de requisitos garantiza ciertas propiedades (lo que podemos entender como verificación de propiedades de la descripción de requisitos). La importancia de tal verificación es particularmente relevante en el contexto de sistemas críticos, como es el caso del sistema de control del marcapasos.

Desde la descripción informal fueron identificadas las siguientes propiedades que deberían ser *invariantes* a lo largo de la ejecución del sistema.

1) `@T(mBATTERYvoltage < BatteryLevel)` $\Rightarrow$ `mcPulseCondition' = POR`
2) `@T(tMagnetON)` $\wedge$ `(mMagnet = On)` $\Rightarrow$ `( cCHAMBERSpaced' = cCHAMBERSpaced )`
3) `mcPulseCondition' = Temporal` $\Rightarrow$ `( cCHAMBERSpaced' = O` $\wedge$ `cRESPONSEsensing' = O )`
4) `cCHAMBERSpaced' = O` $\Rightarrow$ `mcPulseCondition' = Temporal`
5) `mcPulseCondition' = POR` $\Rightarrow$ `( cCHAMBERSpaced'=V` $\wedge$ `cCHAMBERSsensed'=V` $\wedge$ `cRESPONSEsensing'=I ).`

La propiedad 1 describe una de las situaciones críticas del sistema: si el voltage de la batería cae por debajo del nivel normal, el marcapasos debe funcionar en modo `POR`, el cual posee una configuración por defecto para que el consumo de batería sea mínimo. La propiedad 2 indica que al pasar a modo magnético, la configuración de la camara de pulsado no debe cambiar. Las propiedades 3, 4 y 5, por otra parte, reflejan los valores que deben mantener las variables controladas en los modos respectivos.

Como se mencionó anteriormente, la propiedad 1 es una de las situaciones críticas del sistema. SPIN fué capaz, en pocos segundos, de encontrar varias violaciones a esta propiedad. Estos contraejemplos mostaron que la tabla de modos que aparece en la figura 1 era incompleta; la condición crítica `@T(mBATTERYvoltage < BatteryLevel)` no fue considerada en diversas situaciones. Por ejemplo, se debieron agregar las siguientes nuevas filas en la definición de `mcPulseCondition`:

OLD MODE	EVENT	NEW MODE
MAGpacenow	@T(mBATTERYvoltage < BatteryLevel)	POR
MAGtemporal	@T(mBATTERYvoltage < BatteryLevel)	POR

Por otra parte, la omisión de la condición crítica mencionada anteriormente también afectó las definiciones de las variables controladas, ya que el modo de bradicardia en `POR` debe ser `VVI`. Las tablas no contemplaban dicha situación, por lo que debimos extender la definición, por ejemplo, de la variable `cCHAMBERSspaced` (figura 3) agregando filas similares a las siguientes:

PaceNow	...	...	@T(mBATTERYvoltage < BatteryLevel)	...
MAGpacenow	...	...	@T(mBATTERYvoltage < BatteryLevel)	...
cCHAMBERSpaced	O	A	V	D

Luego de solucionar los errores mencionados, las propiedades 1, 2 y 3 pudieron ser verificadas en menos de 10 segundos usando SPIN.

Las propiedades 4 y 5 están relacionadas; ambas tratan de establecer la relación que debe cumplirse entre la clase de modos y las variables controladas. El análisis de dichas propiedades manifestaron que existían diversos errores en nuestra especificación. A partir de la información provista por los contraejemplos encontrados, se decidió primero verificar la validez de una propiedad adicional:

6) mcPulseCondition′ = Normal ⇒ (tMODEbradPA ∨ tMODEbradPV ∨ tMODEbradPD).

Esta propiedad indica que si el marcapasos funciona en modo Normal, entonces el médico debería haber seteado previamente la configuración del modo de bradicardia que indica cómo debe de realizarse el pulsado. El resultado fue sorpresivo, ya que la condición resultó ser inválida. Luego de una revisión exhaustiva del modelo se advirtió que la variable de modo de bradicardia (mMODEbrad) admite también el valor off, indicando que no tiene programa. En este punto se retomó la lectura de la especificacíon informal del marcapasos y se encontró que la descripción en este caso era incompleta. Luego de una búsqueda de información adicional, se encontró la solución al problema planteado, que consistió en agregar nuevas condiciones (por ejemplo, mMODEbrad!=off) a la tabla de modos que aseguren que el marcapasos puede pasar a modo Normal sólo si previamente el médico ha seteado la configuración requerida. Así también, se decidió que el valor inicial correcto de mMODEbrad debe ser el valor off, para que luego el médico sea quién lo configure adecuadamente. A modo de ejemplo, a continuación se presenta una de las modificaciones realizadas a la tabla de modo de la figura 1 para solucionar el problema mencionado anteriormente:

OLD MODE	EVENT	NEW MODE
Temporal	@F(tTELEMETRYsignal) when (mMODEbrad!=off)	Normal

Finalmente, se descubrió que la definición de la tabla de condición del término tMODEbradPD era incompleta debido a que uno de los diecinueve valores posibles fue omitido durante la transcripción de la especificación informal a SCR.

VII. Conclusión y Trabajos Futuros

En este artículo hemos estudiado la posibilidad de utilizar una herramienta de verificación estándar, un model checker off-the-shelf, para el análisis de especificaciones de requisitos realizadas utilizando el método SCR. Para evaluar las capacidades de una herramienta de estas caracterśticas para analizar especificaciones de requisitos, y ver en qué medida la herramienta puede contribuir a mejorar la calidad de la especificación, se desarrolló un caso de estudio de captura de requisitos usando el método SCR. El mismo consistió en la captura de requisitos de parte de un sistema crítico, más precisamente, un sistema de marcapasos. Este caso de estudio, un desafío del grupo Software Quality Research Laboratory de la Universidad McMaster [1], constituye un caso no trivial, sobre el cual aplicar la notación SCR resulta relevante. La relevancia de este estudio tiene que ver con que si bien SCR cuenta con un set de herramientas potente, las mismas no son de acceso público, lo que limita la utilización del método (que ha demostrado ser sumamente útil para sistemas críticos). Es generalmente aceptado que contar con herramientas de análisis automático es hoy en día indispensable para conseguir que una técnica formal sea adoptada en la práctica. Hemos evaluado si usar una herramienta off-the-shelf (en contraste con las herramientas del SCR Toolset, diseñadas y adaptadas específicamente para aprovechar las características de la notación) contribuye al proceso de captura de requisitos. Hemos analizado con esta herramienta la especificación en busca de inconsistencias, porciones incompletas de la especificación, y otros problemas de validación de la misma.

La evaluación de la especificación formal en SCR, y el análisis automático de la misma, resultó en nuestra opinión satisfactorio. Gracias al análisis realizado se encontraron y repararon errores importantes: incompletitud de algunas tablas, falta de información en la especificación informal de requerimientos, errores en la traducción de los requerimientos formales a SCR y en el estado inicial. Para las propiedades identificadas como importantes para la especificación, y a pesar de que la traducción a SPIN es relativamente simple (comparándola probablemente con la forma en que SCR Toolset, una herramienta con varias décadas de madurez, hace uso de model checking), no encontramos en el análisis problemas de escalabilidad. Cabe destacar que nuestro análisis no contempla la totalidad del marcapasos en la especificación, sino una porción no trivial del mismo. Una de las tareas que resultó más difícil fue la interpretación de los contraejemplos obtenidos del model checker. Estos contraejemplos están expresados en términos de la forma en que se codifica la especificación original en Promela, y su "bajo nivel de abstracción" complica la tarea de depuración de la especificación.

Como parte de nuestro trabajo futuro, planeamos extender el trabajo aquí expuesto a la totalidad del marcapasos, extendiendo la especificación a un modelo completo del aparato, y aplicando análisis automático a la especificación resultante. Buscamos en particular chocarnos con problemas de escalabilidad, que demanden un uso más sofisticado de model checking (e.g., a través de abstracción). Es indudable que eventualmente deberemos lidiar con problemas de escalabilidad; en estos casos, trabajaremos en identificar las

fuentes y adaptar las técnicas de análisis vía model checking, aprovechando características particulares de la notación y la forma en la que se emplea. También planeamos trabajar en formas de visualizar la salida obtenida del model checker de manera que la misma sea de más fácil intepretación por parte del desarrollador. Diferentes mecanismos de visualización gráfica, al nivel de abstracción de la especificación original, serán útiles para esta tarea.

REFERENCIAS

[1] Pacemaker Formal Methods Challenge, *PACEMAKER System Specification*, Software Quality Research Laboratory, McMaster Univerity.

[2] M. Archer, *TAME: Using PVS strategies for special-purpose theorem proving*, Annals of Mathematics and Artificial Intelligence, 29(1-4):131189, February 2001.

[3] R. Bharadwaj and S. Sims, *Salsa: Combining constraint solvers with BDDs for automatic invariant checking*, in proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000), Berlin, March 2000.

[4] D. Beyer, A. Chlipala, T. Henzinger, R. Jhala and R. Majumdar, *Generating Tests from Counterexamples*, in Proc. of ICSE 2004, IEEE, 2004.

[5] T. Bultan and C. Heitmeyer, *Applying Infinite State Model Checking and Other Analysis Techniques to Tabular Requirements Specifications of Safety-Critical Systems*, Design Automation for Embedded Systems, 12(1-2), 2008

[6] M. Engel, M. Kubica, J. Madey, D. Parnas, A. Ravn, A. van Schowen, *A Formal Approach to Computer Systems Requirements Documentation*, Hybrid Systems, LNCS 736, Springer, 1993.

[7] A. Gargantini and C. Heitmeyer, *Using Model Checking to Generate Tests from Requirements Specifications*, in Proc. of ESEC/FSE 1999, LNCS, Springer, 1999.

[8] C. Ghezzi, M. Jazayeri y D. Mandrioli, *Fundamentals of Software Engineering*, 2nd. Edition, Prentice-Hall, 2002.

[9] C. Heitmeyer, B. Labaw, D. Kiskis, *Consistency Checking of SCR-Style Requirements Specifications*, en Proceedings de IEEE International Symposium on Requirements Engineering, York, Inglaterra. IEEE Computer Society 1995.

[10] R. Bharadwaj, C. Heitmeyer, *Model Checking Complete Requirements Specifications Using Abstraction*, in Proc. Automated Software Engineering, 1999.

[11] C. Heitmeyer, M. Archer, R. Bharadwaj, R. Jeffords, *Tools for constructing requirements specifications: the SCR Toolset at the age of ten*, Computer Systems Science and Engineering, 20(1), CRL Publishing, 2005.

[12] G. J. Holzmann, *Design and Validation of Computer Protocols*, Prentice-Hall, 1991.

[13] P. Jalote, *An Integrated Approach to Software Engineering*, 3rd. Edition, Springer, 2005.

[14] R.Jeffords and C. Heitmeyer. *Automatic generation of state invariants from requirements specifications*, in Proc. of 6th ACM SIGSOFT Symp. Foundations Softw. Eng., 1998.

[15] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems specification*, Springer, 978-3-540-97664-6, pp. I-XIV, 1-427, 1992.

[16] D.L Parnas et al. *Software Requirements for the A-7E Aircraft*, Naval Laboratory Report Documentation, 1992.

[17] I. Sommerville, *Software Engineering*, 8th Edition, Addison-Wesley, 2006.