UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Ingeniería del Software

Verificación de la fiabilidad del gestor de experimentos del proyecto 5Genesis

Verification of 5Genesis' experiment manager's reliability

Realizado por
Darío Arrebola Taza

Tutorizado por
María del Mar Gallardo Melgarejo

Departamento
Lenguajes y Ciencias de la Computación
Universidad de Málaga

MÁLAGA, FEBRERO DE 2021

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA

Resumen

Uno de los objetivos del proyecto europeo H2020 5Genesis, del que el grupo MORSE de la Universidad de Málaga forma parte, es proporcionar una plataforma 5G de experimentación para los desarrolladores de software. La gestión y coordinación de las pruebas solicitadas por los experimentadores a la plataforma es realizada por un software complejo, desarrollado en el proyecto, y descrito en distintos entregables (concretamente los 2.3, 2.4 y 3.15). El objetivo de este trabajo fin de grado es analizar este software (específicamente los procesos de despliegue y gestión de los experimentos en la plataforma) para comprobar que funciona correctamente con respecto a algunas propiedades esenciales. Para realizar las tareas de verificación, en el proyecto, se ha utilizado el model checker SPIN. Concretamente, haciendo uso de su lenguaje de entrada, denominado Promela, se ha construido un modelo del software que es fiel a la implementación realizada, en el sentido de que tiene en cuenta todos los escenarios posibles. El caso de uso ideal para este sistema consiste en un usuario que inicia correctamente una sesión, envía la configuración deseada para su experimento a la plataforma, ésta lo ejecuta y devuelve los resultados al usuario. Sin embargo, como es lógico, la implementación y el modelo construido también tienen en cuenta experimentos en los que se produce algún error por causa del usuario o por falta de recursos en la plataforma, por ejemplo. Esto último puede ocurrir si hay varios usuarios que simultáneamente ejecutando experimentos sobre la plataforma. En este caso, el sistema y el modelo se comportan de forma altamente concurrente ya que la plataforma tiene que gestionar a todos los usuarios. Para comprobar si la plataforma se comporta correctamente ante escenarios complejos en los que hay varios usuarios se han enunciado varias propiedades

críticas que se han especificado en la lógica temporal lineal LTL. Estas propiedades LTL se han suministrado a la herramienta SPIN que es capaz de comprobar automáticamente si el modelo Promela desarrollado cumple cada una de ellas.

Palabras clave: métodos formales, 5G, internet, Spin/promela, plataformas de experimentación.

Abstract

One of the objectives that European project H2020 5Genesis, which group MROSE from University of Málaga is a part of, is to provide a 5G platform for software developers to experiment with. Management and coordination of the requested tasks by the experimenters on the platform is handled by complex software, developed in the project and described in several deliverables (specifically 2.3, 2.4 and 3.15). The goal of this final degree project is to analyze this software (focusing on the processes of deployment and management of experiments in the platform) to verify the correct functioning of some vital properties. To perform the tasks of verification in the project, the tool used has been SPIN model checker. Specifically, making use of its own language, called Promela, a software model has been built, accurate to the actual implementation, in the sense that considers every possible scenario. The ideal use case for this system consists of the process in which a user logs in successfully, then sends the desired configuration for the experiment to the platform. Then the experiment will be executed, and the experimenter will have the results returned. However, the implementation and the constructed model also consider the experiments in which errors arise because of mistakes made by the users or lack of resources from the platform. The latter case may occur if several users are simultaneously executing experiments on the platform. In those cases, both the system and the model have a highly concurrent behavior provided that the platform must manage the totality of the users. To check if the platform behaves correctly in complex scenarios where there are multiple users, a series of critical properties have been defined in Lineal Temporal Logic

LTL. These properties have been provided to SPIN, a tool which can check automatically if the developed Promela model satisfies every one of them.

Keywords: formal methods, 5G, Internet, process simulation, Promela.

Índice

Resumen i
Abstract ii
Índice iii
1. Introducción 1
1.1 Motivación: 1
1.2 Objetivos: 3
1.3 Estructura: 4
2. Contexto: 5Genesis y las distintas generaciones de las comunicaciones móviles 5
2.1 Primera Generación: 8
2.2 Segunda Generación: 9
2.3 Tercera Generación: 10
2.4 Cuarta Generación e introducción al 5G y 5Genesis: 12
3. Estudio de las tecnologías a utilizar 15
3.1 Los métodos formales en el ámbito del software 15
3.2 El lenguaje Promela 17
3.2.1 Variables: 17
3.2.2 Canales: 18
3.2.3 Señales: 19
3.2.4 Procesos: 19
3.3 Logica Temporal Lineal (LTL) 22
3.4 La herramienta SPIN y su interfaz iSpin 23
4. Metodología del trabajo 29
4.1 Fase 0 30
4.2 Fase 1 30
4.3 Fase 2 30
4.4 Fase 3 31
4.5 Fase 4 31

5. Solución propuesta 33

5.1 Exposición de los diagramas, explicación de la plataforma e identificación de entidades y su función. 33

5.2 Máquinas de estado de los actores. Definición del sistema inicial. Pruebas, problemas y soluciones 37

5.3 Extensión con varios usuarios. Problemas y soluciones encontrados 46

5.4 Modelo final 47

5.5 Prueba de propiedades sobre el modelo 51

6. Conclusiones y líneas futuras 57

Bibliografía 59

1

1.Introducción

1.1 Motivación:

La elección de esta línea original de trabajo de fin de grado es el resultado de haber tenido una experiencia muy positiva en la asignatura de métodos formales para la ingeniería del software, en la que, tras haber trabajado con varios lenguajes de especificación formal, las aplicaciones prácticas que se podían hacer eran interesantes. Además, la obtención de una beca de colaboración en el grupo MORSE, situado en el edificio Ada Byron de la Universidad de Málaga, me ha permitido familiarizarme con el proyecto europeo 5Genesis. Dentro de este proyecto se están desarrollando diversos sistemas software complejos que nos han dado una buena oportunidad para aplicar lo aprendido en la asignatura sobre un caso real. En particular, se seleccionó el software que gestiona la realización de experimentos sobre la plataforma 5G que dispone el grupo. Así, se ha utilizado la metodología habitual en el contexto de los métodos formales estudiada en la asignatura, (1) construcción de un modelo promela del sistema, (2)

especificación de unas propiedades LTL que deseamos que el modelo satisfaga, y (3) uso de la herramienta SPIN para analizar las propiedades sobre el modelo.

1.2 Objetivos:

Como ya se ha mencionado anteriormente, el objetivo principal de este trabajo es el análisis formal del software que gestiona la plataforma de experimentación del proyecto 5Genesis. Para ello, un primer paso ha sido la creación de un modelo del sistema fiel al funcionamiento real del mismo. La verificación de este modelo es imprescindible para garantizar que el software desarrollado tiene un buen diseño carente de errores.

En particular, hemos abordado los siguientes objetivos secundarios:

En primer lugar, el dominio que abarca el sistema modelado debe cubrir las principales funcionalidades del sistema real. Para conseguir esto, hemos revisado exhaustivamente la documentación oficial del proyecto, y en base a su contenido escrito y a los diferentes diagramas que contiene y que muestran diversos casos de uso, hemos modelado los distintos procesos que representan cada entidad involucrada en el sistema real.

En segundo lugar, hemos analizado una corrección básica de este modelo inicial asumiendo que en el sistema hay un único usuario. Tras realizar muchas simulaciones con un usuario en el sistema, que nos han permitido depurar el modelo y obtener un comportamiento similar al del sistema real, hemos procedido a extender el modelo de manera que sea capaz de funcionar en entornos concurrentes con varios usuarios ejecutando experimentos de forma simultánea.

Por otra parte, aunque este objetivo no se contempló inicialmente, se decidió crear una estructura de datos que simulase los experimentos de los usuarios, guardando su estado y, de esta forma, mejorar la fidelidad del modelo con respecto al sistema real.

Por último, una vez modelado el sistema completo, se ha comprobado que el sistema satisface una serie de propiedades críticas de interés utilizando la herramienta SPIN. Este análisis nos ha permitido evaluar la calidad del software real desarrollado en el proyecto 5Genesis.

1.3 Estructura:

El presente documento se estructura del modo siguiente. En Capítulo 2, se describe el contexto en el que se ha desarrollado el trabajo con respecto a las comunicaciones móviles y al proyecto 5Genesis. En concreto, se describe la evolución de éstas y lo que han significado y significan en la sociedad actual. En el Capítulo 3, se introducen los métodos formales haciendo un especial hincapié en las herramientas y tecnologías que se han utilizado a lo largo de este trabajo. En particular, se describe el lenguaje Promela, la lógica LTL, y la herramienta Spin y su interfaz, denominado iSpin. En el Capítulo 4, se explican detalladamente la metodología de trabajo que se ha seguido en el trabajo. El proceso de desarrollo del modelo promela construido, así como los resultados obtenidos, se describen en el Capítulo 5. Finalmente, el Capítulo 6, está dedicado a discutir las conclusiones y posibles trabajos futuros.

2

2.Contexto: 5Genesis y las distintas generaciones de las comunicaciones móviles

Es por todos conocida la inminente llegada de la tecnología 5G a nuestro entorno tecnológico, económico y social. A lo largo del tiempo hemos sido testigos de la revolución que la llegada de cada nueva generación de comunicaciones móviles ha supuesto, siempre con un impacto mayor que la anterior. Pero detrás de todos esos avances y tecnologías punteras que se esperan, hay un más que notable trabajo por parte de grupos de investigación y desarrollo.

El proyecto europeo 5Genesis H2020 está compuesto por varios grupos de investigación y empresas internacionales del sector de las comunicaciones. El grupo MORSE de la Universidad de Málaga realiza la coordinación técnica de este proyecto. El objetivo de 5Genesis es el desarrollo de plataformas de experimentación en las que los desarrolladores de software para entornos puedan analizar distintos requisitos de calidad (KPIs) en diversos escenarios de aplicación. En el proyecto, por un lado, se han identificado los KPIs de mayor importancia y se ha desarrollado software que permite su evaluación. La evaluación consiste, de forma resumida, en la especificación de distintos casos de prueba que corresponden a potenciales casos de uso que abarcan tanto escenarios en los que se utilizan redes muy controladas como otros en los que la red tiene un uso masivo y a gran escala.

Más específicamente, el proyecto tiene cuatro objetivos principales:

· Implementar y verificar un estándar para las redes 5G.

· Integrar los avances tecnológicos de las distintas disciplinas involucradas en el desarrollo de 5G para confeccionar un marco en el que el 5G alcance su máximo potencial.

· Crear una estructura coordinada desde la cual los usuarios puedan realizar experimentos sobre plataformas 5G que favorezcan su desarrollo.

· Proporcionar apoyo a las nuevas aplicaciones y escenarios 5G que puedan surgir en el futuro.

Como ya se ha comentado, Málaga es una de las ciudades que participa en el proyecto junto con Atenas, Limassol, Berlín y Surrey. En concreto, en Málaga se está llevando a cabo la construcción de una plataforma de experimentación para la simulación de casos de uso de aplicaciones que se ejecutan sobre 5G teniendo en cuenta las propiedades específicas (KPIs) requeridas por el experimentador. Lógicamente, la correcta implementación de esta plataforma es crucial para el éxito del proyecto.

La implementación de la plataforma involucra varios actores (procesos) que se comunican y sincronizan continuamente para realizar los experimentos suministrados por los usuarios. La plataforma es, por lo tanto, un sistema concurrente con una gran cantidad de posibles comportamientos que no pueden analizarse de forma manual. Esta es la razón de que en este trabajo fin de grado se haya optado por el uso de la herramienta SPIN, que es un model checker capaz de analizar automáticamente y de forma exhaustiva todos los posibles comportamientos de un sistema concurrente. El primer paso para poder utilizar SPIN es la descripción o modelado de la plataforma en el lenguaje de entrada de SPIN, que se denomina Promela. El modelo puede ser analizado frente a propiedades generales como la ausencia de bloqueo, o propiedades más específicas descritas en la lógica temporal lineal LTL.

SPIN es un método formal en el sentido de que sus veredictos (el modelo satisface una propiedad o no la satisface) son siempre fiables. Esto es así porque la técnica está basada en métodos matemáticos cuya validez ha sido demostrada. Como ya se ha mencionado anteriormente, la ventaja del uso de SPIN, o de los métodos formales en general frente a otras técnicas de pruebas de software tradicionales es que la herramienta analiza *todos* los posibles comportamientos del sistema modelado de manera que no haya posibilidad de que algún comportamiento erróneo sea pasado por alto. Además, la lógica temporal LTL permite especificar propiedades no triviales sobre el comportamiento del sistema que SPIN pueda analizar también de forma automática. En este trabajo de fin de grado, se describe el proceso seguido para el modelado y verificación de la plataforma de experimentación del proyecto 5Genesis. El proceso consta de varias fases en las que se han utilizado diferentes herramientas. Finalmente, se hace un resumen de los problemas que hemos encontrado a lo largo del proceso y de las soluciones adoptadas.

Para entender un poco mejor la relevancia e implicaciones que tiene la llegada de una nueva generación de comunicaciones móviles, a continuación, repasaremos el impacto que las generaciones pasadas han tenido, desde un enfoque tanto técnico como social. Hasta cierto

punto, el progreso de las comunicaciones móviles es uno de los principales factores que han conformado la sociedad en la que hoy en día vivimos.

2.1 Primera Generación:

En la década de los 80, aparecieron los llamados teléfonos celulares. Se denominaban así porque el área en el que operaba el servicio de comunicación estaba dividida en células, es decir, en pequeñas parcelas dentro de las cuales cada dispositivo podía comunicarse con otros dispositivos utilizando una frecuencia específica. La principal mejora con respecto a los sistemas de comunicación radiofónicos reside en el hecho de que los teléfonos celulares podían utilizar frecuencias de células distintas. Esto implica que una llamada podía permanecer activa incluso aunque se moviera por un territorio en el que la cobertura estuviera suministrada por células distintas. Anteriormente, en los sistemas de radio, la llamada sólo podía llevarse a cabo si el usuario que efectuaba la llamada se encontraba bajo la cobertura de la antena que estuviese haciendo de punto de conexión.

Los principales retos, inconvenientes y aspectos a mejorar que surgieron en esta generación fueron:

· La baja capacidad de carga de los sistemas, puesto que no se disponían de frecuencias suficientes para la creciente demanda de estos dispositivos.

· La cobertura y calidad del sonido aún tenían mucho margen de mejora.

· La falta de un estándar común provocó que como en cada territorio se utilizaba un estándar local la integración de sistemas que operaban bajo distintos estándares fuera muy complicada.

· El hecho de que las llamadas no estuvieran encriptadas suponía una falta de seguridad y privacidad.

A pesar de todo, las mejoras fueron significativas con respecto a su predecesor, el sistema radiofónico. Prueba de ello, es el anteriormente mencionado incremento en la demanda. Por entonces, el uso de las comunicaciones móviles aún se limitaba exclusivamente a la transmisión de señales de voz.

2.2 Segunda Generación:

En esta generación se amplían los límites de lo que hasta ese momento significaba la comunicación móvil. Se introducen los servicios de mensajería de texto instantánea (SMS) y la posibilidad de compartir archivos multimedia. Se mejoran algunos aspectos clave con respecto a la generación pasada: las llamadas ya pueden encriptarse y su calidad mejora de forma drástica, fundamentalmente debido al uso de señales digitales en lugar de las tradicionales las señales analógicas.

Además, un mejor aprovechamiento del espectro de frecuencias permitió que la capacidad de cada banda fuese mayor, permitiendo así que más usuarios utilizasen simultáneamente el servicio. Estos factores fueron clave para un crecimiento sin precedentes en el sector. Este un momento fue crucial para nuestra sociedad, pues con la segunda generación llegó el internet a los dispositivos móviles y, con ello, una cantidad innumerable de posibilidades. Por primera vez en la historia, se tuvo al alcance de nuestra mano toda la información que internet abarcaba (aunque aún era muy limitada en aquel entonces. En cualquier caso, la posibilidad de acceder en cualquier momento cualquier dato que fuese preciso conocer representa, sin duda, un punto de inflexión para la sociedad, como ya lo fueron otras tecnologías e invenciones pasadas. El primer teléfono que permitía el acceso web fue el Nokia 9000 Communicator, lanzado en 1996. Desde él, era posible tanto visitar páginas web como recibir fax.

A pesar de todo, aún quedaba un largo camino por recorrer, puesto que la velocidad de transmisión seguía siendo muy baja lo que constituyó el principal reto al que tuvo que enfrentarse la siguiente generación.

2.3 Tercera Generación:

La tercera generación introdujo cambios que mejoraban la cobertura, calidad y velocidad de la conexión con respecto a las anteriores generaciones notablemente. A través del uso de antenas con capacidades muy superiores y el uso de una arquitectura dividida en capas (de aplicación, control y transporte) se obtuvieron los avances previamente mencionados. También se introdujeron características muy útiles como el ancho de banda dinámico, que permitía que cada dispositivo consumiese un ancho de banda proporcional a sus necesidades.

Hasta ese momento, la utilidad de las comunicaciones móviles era limitada. Si bien las generaciones anteriores supusieron una revolución, su uso quedaba relegado al campo de la mensajería casi por completo. Sin embargo, la red 3G permitió la creación de multitud de nuevas aplicaciones interesantes.

Por ejemplo, surgieron los primeros aparatos GPS en tiempo real para los coches y se empezó a introducir la posibilidad de descargar música en cualquier momento (Iphone con Itunes en 2003), en lugar de obligar a tener canciones previamente en un disco o descargadas en el ordenador.

Es cierto que aún la velocidad de descarga no era del todo significativa lo que limitaba claramente las posibilidades de uso de la red móvil, pero fue un punto de no retorno. A partir de este momento, comienza la evolución de los teléfonos móviles hacia los smartphones de hoy en día.

En los primeros años, los modelos nuevos de teléfono trataban siempre de ser más pequeños que los de versiones anteriores. Se evolucionó desde teléfonos con forma de bloque hasta los modelos de bolsillo tan icónicos de esta época. A partir del momento en que los teléfonos deben también albergar nuevas funcionalidades, su diseño cambia de manera que se aumenta cada vez más el tamaño de los dispositivos. Uno de los primeros teléfonos en realizar esta adaptación para un

uso intensivo de la red fue la Blackberry Curve, dotada de un teclado especialmente pensado para poder usar rápidamente los servicios de mensajería instantánea y las redes sociales. Desde entonces, todos los dispositivos experimentaron una evolución similar. Las pantallas, por lo tanto, fueron creciendo, ya que cada vez era más relevante la posibilidad de que las pantallas mostraran diversos tipos de información como chats, fotos o vídeos compartidos por otras personas.

A pesar de todo, internet seguía siendo territorio del PC clásico. Un dato relevante de esta época es que, por ejemplo, en 2009 casi la totalidad del uso de internet se hacía desde ordenadores. Parte del problema era que el acceso a la red 3G no era muy asequible: en 2008 el precio de una videollamada era de entre 50 y 70 céntimos el minuto, si eras cliente de Vodafone, unos 28 céntimos en Orange mientras que Movistar tenía varias tarifas entre 10 y 50 céntimos, y esto solamente en red nacional [8]. Una videollamada internacional valía nada menos que 2 euros cada minuto de media. Para los demás servicios la condición era similar, tanto en contratos de prepago como de tarifa. Las tasas no eran baratas así que, si bien el uso de la red era posible, muchas veces era el último recurso.

Conforme avanzó el tiempo, se pudo acceder a estas tecnologías con tarifas de datos móviles a precios más competitivos, lo que permitió que muchas más personas utilizasen este servicio. De esta forma, nació el fenómeno de las aplicaciones móviles como WhatsApp mientras que el diseño de los teléfonos siguió acercándose hacia el modelo de Smartphone que hoy conocemos.

Aunque tanto la velocidad como la calidad de la cobertura seguían siendo aspectos que se podían mejorar, las redes 3G sentaron las bases de lo que serían las redes móviles modernas.

2.4 Cuarta Generación e introducción al 5G y 5Genesis:

La siguiente generación, la llamada 4G, fue una generación cuyas diferentes contribuciones aparecieron bastante más repartidas en el tiempo. Las siglas LTE que solemos ver acompañando siempre al 4G significan evolución a largo plazo (Long Term Evolution), que fueron las mejoras que se fueron aplicando a la red 4G lanzada inicialmente, concretamente a finales de 2009, en Noruega y Suecia.

Las redes 4G permitieron un incremento sustancial de la velocidad de acceso a la red móvil y una importante mejora en la cobertura, teniendo prácticamente la totalidad del terreno urbano cubierto.

En esta época, continúa la tendencia en el crecimiento del número usuarios ya iniciada en la generación anterior 3G. Este crecimiento ha sido tal que, hoy en día, gran parte de la población mundial cuenta con, al menos, un teléfono móvil. El aumento de la velocidad de la red y la posibilidad de disfrutar de precios mucho más asequibles permitieron que los teléfonos móviles con conexión a la red se convirtieran en una de las tecnologías más versátiles con las que cuenta la sociedad hoy en día.

En la actualidad, se puede acceder a servicios de streaming de alta calidad para, por ejemplo, hacer la compra de forma remota, pedir que el transporte venga a recogernos a nuestro domicilio, escanear códigos para poder ver documentos sin necesidad de generar copias físicas, poder acceder a toda la información que tengamos guardada en la nube. Es evidente que esta lista podría alargarse mucho más, pero al final todo se reduce a que, gracias a las múltiples mejoras que se dieron en las redes móviles, el mundo entero ha evolucionado con ellas, siendo sin duda un factor clave en nuestro desarrollo como sociedad.

Con respecto a los datos de uso de internet en ordenadores frente a su uso en móviles, se han cambiado las tornas por completo. Hoy día, la mayor parte de la población accede a internet

desde dispositivos móviles: según la Asociación para la Investigación de Medios de Comunicación el 91.5% de internautas accede a la red desde su teléfono móvil [7].

Además, ya no son sólo los teléfonos los dispositivos con acceso a la red, sino que cada vez hay más aparatos que hacen uso de esta tecnología. Un caso muy relevante es el de los drones, que han permitido un gran avance en el campo militar y en la exploración y reconocimiento de zonas de difícil acceso.

A pesar de todo, la red 4G no es apropiada para ciertas tareas críticas, pues aún hay aspectos a ser mejorados como el ancho de banda, la latencia y la cobertura. Las comunicaciones móviles siguen siendo un campo en continuo desarrollo y, por lo tanto, es crucial seguir identificando las oportunidades, retos y necesidades que surgirán en la nueva generación 5G.

Y es que, en la actualidad, no es sencillo predecir qué impacto tendrá la llegada de la nueva generación, cómo se le sacará el máximo provecho y cómo contribuirá a nuestra sociedad, pero en base a la experiencia obtenida de generaciones anteriores, tenemos motivos para ser ambiciosos.

Por ejemplo, algunas de las pruebas prácticas que se plantean en el proyecto 5Genesis son el uso de cámaras en tiempo real por parte de la policía y el intento de dar servicio a muy baja latencia en escenarios en los que hay una gran densidad de usuarios en un espacio limitado.

En cualquier caso, para que sea posible tanto esta como el resto de las aplicaciones que, sin duda, surgirán, el trabajo desempeñado en proyectos como 5Genesis está siendo crucial para que puedan llevarse a cabo tan pronto como sea posible.

3

3.Estudio de las tecnologías a utilizar

3.1 Los métodos formales en el ámbito del software

Son un enfoque del software desde una perspectiva analítica que permite verificar la corrección de programas, modelos y software, en general, haciendo uso de herramientas basadas tanto en la lógica como en las matemáticas. Como ya se comentó con anterioridad, la diferencia entre comprobar la corrección de un sistema mediante pruebas tradicionales y el uso de métodos formales es que en este segundo caso se analizan de forma exhaustiva todas las ejecuciones posibles del sistema, lo que hace posible detectar fallos en el comportamiento, difíciles de encontrar utilizando otras técnicas.

Su utilidad, por lo tanto, radica en su eficiencia para detectar errores de diseño poco convencionales. En resumen, los métodos formales proporcionan herramientas realmente fiables, capaces de garantizar el comportamiento correcto sistema.

De entre todos los métodos formales existentes, en este trabajo, nos hemos centrado en la técnica denominada "Model Checking". De manera muy reducida, una herramienta de model checking tiene como entrada dos especificaciones: (1) el modelo del sistema M que se quiere analizar (el modelo podría ser el propio sistema, según la herramienta utilizada) y (2) la propiedad f que queremos que satisfagan todos los comportamientos del modelo. La herramienta analiza de forma exhaustiva todo el modelo buscando comportamientos que no cumplan la propiedad. Este análisis es completamente algorítmico, es decir, no requiere de la intervención del usuario. Si la herramienta encuentra un comportamiento incorrecto lo devuelve como una traza de error que se llama contraejemplo. Los contraejemplos son muy importantes porque proporcionan información al usuario sobre cuál ha sido la causa del error lo que ayuda a la depuración del sistema. En caso contrario, si todos los comportamientos del sistema son acordes con la propiedad especificada, la herramienta concluye con el veredicto esperado: el modelo es correcto con respecto a la propiedad.

Uno de los problemas implícitos en la técnica de model checking es el llamado "problema de la explosión de estados", que ocurre cuando el sistema a analizar es demasiado grande, de forma que los recursos de la máquina impiden su análisis exhaustivo. Para resolver este problema se han desarrollado diferentes técnicas: como son la reducción de orden parcial, la compactación de los estados, etc. En cualquier caso, como es natural, los límites físicos de la máquina siempre impedirán que cuando los sistemas son muy grandes no sea posible analizarlos de forma exhaustiva. Sin embargo, en la práctica, el éxito del model checking en la industria se debe a que la técnica ha sido capaz de analizar con éxito gran cantidad de sistemas concurrentes complejos como el que se presenta en este proyecto.

De todas las herramientas de model checking, en este trabajo fin de grado se ha utilizado SPIN que analiza modelos descritos en su lenguaje de entrada (denominado Promela) frente a propiedades descritas en la lógica temporal lineal LTL. En lo que sigue, resumimos brevemente las características de estos dos lenguajes.

3.2 El lenguaje Promela

Para abordar la tarea de *Verificación de la fiabilidad del gestor de experimentos del proyecto 5Genesis* se hará uso de Promela, un lenguaje para la descripción de modelos que, utilizando SPIN [1][5][6], permite verificar la integridad y corrección de sistemas, analizarlos y encontrar cualquier fallo de diseño que pudiera existir.

Promela es un lenguaje imperativo con una sintaxis muy similar al lenguaje C. Como es un lenguaje de modelado no contiene todas los tipos y constructores de C, a la vez que añade otros constructores propios que permiten describir fácilmente la comunicación y sincronización de procesos en sistemas de memoria compartida y distribuido. Los elementos principales de cualquier modelo Promela son los siguientes:

3.2.1 Variables:

Constituyen la parte más elemental de cualquier programa Promela. No son distintas de las variables que se crean en los diferentes lenguajes de programación imperativa. Los tipos de variables que podemos usar son: bit, bool, byte, short, int y unsigned. Además, las variables se pueden organizar en estructuras de datos como arrays, para tener una colección ordenada de elementos del mismo tipo, o structs, que nos permitirán crear un tipo de datos personalizado más complejos. Es de destacar que promela no permite la declaración de variables de tipo punto flotante.

Como es habitual, las variables se dividen en dos categorías según el lugar en el que las declaremos: globales y locales. Las variables globales se declaran fuera de los procesos, de manera que todos ellos podrán acceder y modificar sus valores. En contraposición, las variables locales

se declaran en el interior de los procesos, por lo que no serán accesible para el resto. Para compartir el valor de una variable local (que no la variable en sí misma) entre dos procesos, estos podrán comunicarse a través de un canal que permita mandar ese tipo de valores. Los procesos también pueden recibir variables como parámetros en el momento de ser creados, como, por ejemplo, el tamaño del búfer que tendrá un proceso.

3.2.2 Canales:

Los canales son las vías a través de las cuales se comunican los procesos típicos de los sistemas de memoria distribuida. Si c es un canal, sobre c se pueden realizar dos operaciones básicas cuya sintaxis viene heredada del lenguaje CSP: escribir sobre el canal un mensaje m (c!m) o leer de un canal un mensaje (c?x). Cada canal tiene dos parámetros que deberemos configurar:

· **Capacidad:** es el número de mensajes que pueden ser guardados en un canal antes de que un proceso los consuma. Según si queremos que nuestro canal sea síncrono o asíncrono, le asignaremos una capacidad de 0, en el primer caso, y una que oscile entre 1 y N, en el segundo, siendo N preferiblemente un número pequeño, puesto que tener canales con mucha capacidad elevará significativamente la complejidad de un modelo a la hora de verificarlo. En caso de un canal ser síncrono, no podrá guardar mensajes y, por lo tanto, el proceso receptor y emisor habrán de estar sincronizados para poder comunicarse. Para este trabajo concreto, los canales tienen una capacidad de tres mensajes siendo por tanto asíncronos.

· **Contenido:** es la información que se transmitirá a través del canal, ya que los canales son variables tipadas. Puede componerse de cualquier conjunto de señales, tipo de datos o incluso canales.

Un ejemplo de declaración de un canal sería el siguiente:

chan foo = [2] of {int, bit, chan}

En este caso estaríamos creando un canal de capacidad dos capaz de almacenar mensajes consistentes en 3-uplas compuestas por un int, un bit y un canal.

Promela proporciona dos funciones interesantes que se pueden usar sobre cualquier canal c: full(c) y empty(c), que nos devolverán un valor de tipo bool en caso de cumplirse la condición de estar lleno o vacío, respectivamente.

3.2.3 Señales:

Aunque su nombre técnico es "constantes simbólicas", a efectos prácticos se tratan de las señales que los procesos se envían a través de los canales para simular sus interacciones. Podemos definir hasta 256 señales diferentes. Por ejemplo, la declaración *mtype = {Nombre1, Nombre2, Nombre3}*, donde define tres tipos de señal distintos.

3.2.4 Procesos:

Son la representación de los componentes del sistema que deseamos crear. Para declarar un proceso se utiliza la palabra reservada es *proctype*. Cuando se crean instancias de los proctypes, cada uno de ellos se ejecutará paralelamente junto a los demás utilizando la semántica del interleaving. Típicamente, los procesos podrán comunicarse entre sí para simular las interacciones que se producen entre dichos elementos en el mundo real.

Es posible definir tanto procesos cuya ejecución termine, como procesos que estén siempre vivos, introduciendo su comportamiento en bucles. En este segundo caso, podemos utilizar las palabras reservadas *do* y *od* al principio y al final de la parte que deseemos que se repita.

Los *proctypes* pueden contener los siguientes elementos típicos y exclusivos del lenguaje Promela:

Guardas lógicas:

Las instrucciones de iteración (do/od) y de selección (if/fi) se han definido para implementar de manera muy sencilla el indeterminismo a nivel de proceso.

Por ejemplo, el fragmento siguiente simula un jugador de piedra, papel o tijeras utilizando la iteración do/od:

Proctype agente(){

do

::canal!piedra;

::canal!papel;

::canal!tijera;

od

}

Como puede observarse, la iteración contiene una secuencia de ramas (que empiezan por los símbolos ::). La primera instrucción de cada rama se denomina guarda y permite la ejecución de la rama correspondiente. Cuando se ejecuta el código do/od se selecciona una de entre todas las ramas cuya guarda es ejecutable. Si ninguna es ejecutable, el proceso se suspende en el do. También es posible utilizar una última rama del tipo "::else" para evitar la suspensión del proceso.

Estas instrucciones son una de las herramientas más importantes para definir correctamente el comportamiento de los procesos.

Puede utilizarse como guarda cualquier instrucción del lenguaje. Por ejemplo, una guarda podría ser simplemente un *true* lógico, lo que significa que la rama correspondiente siempre está abierta (es seleccionable para su ejecución).

Etiquetas:

Las etiquetas permiten marcar un punto relevante del código al que se puede saltar (mediante la instrucción "goto") para redirigir la ejecución del proceso desde cualquier punto de este. Las etiquetas son fundamentales para implementar máquinas de estados en Promela. Para crear una etiqueta es suficiente con escribir un identificador seguido de dos puntos delante del código que se quiere etiquetar. Por ejemplo, una etiqueta podría ser: "*etiqueta:*". Una vez definida, podemos

hacer que el contador de programa del proceso vuelva hasta el punto en el que la hayamos definido mediante la instrucción "*goto etiqueta;*".

De esta manera podremos modelar cualquier comportamiento en el que sea necesaria la repetición de cualquier fragmento del proceso.

Existen tipos especiales de etiquetas que tienen un sisgnificado especial. Por ejemplo, si el nombre de una etiqueta empieza con la cadena *end*, como podría ser "*endEjemplo:*", estaremos definiendo un punto de control del proceso considerado como un estado final válido. Esto significa que en el caso de que la ejecución acabase en ese punto, no se consideraría un error. Es importante remarcar que si un proceso termina antes de ejecutar una instrucción que no está etiquetada como final (end) SPIN considerará que el proceso y el sistema ha terminado en el estado de error bloqueado.

Comunicación a través de canales:

Aunque ya se ha mencionado anteriormente, en esta sección, se describen con más detalle las instrucciones que permiten de lectura y escritura sobre canales. Siguiendo la sintaxis de CSP, el operador "!" se utiliza para enviar un mensaje a través de un canal mientras que el operador "?" se utiliza para leer el primer mensaje que encontremos en el canal. Es importante resaltar que los mensajes almacenados en un canal conservan su orden de inserción (se almacenan como en un buffer). Esta semántica del operador "?" puede tener efectos no deseados, si queremos leer los mensajes en un orden diferente. Para estos casos el operador, Promela proporciona el operador "??". Con él, se puede consumir una señal de un canal, independientemente de su posición en el mismo.

Por último, hay muchos casos en los que el valor de una parte del mensaje que se lee no es relevante para el proceso receptor. En esas ocasiones, se puede utilizar el símbolo "_", que funciona como un comodín aceptando todos los valores que pudieran llegar por el canal.

Creación de los procesos:

Los procesos se crean utilizando la palabra reservada *run* seguida el nombre de un typeproc junto con sus parámetros de inicialización. Aunque Promela permite crear procesos en ejecución, es habitual que la configuración inicial del sistema se cree utilizando un proceso especial del lenguaje denominado *init.* Este proceso se comporta como el procedimiento main en el lenguaje C. Su código es el primero que se ejecuta. El proceso init se utiliza normalmente para inicializar el sistema creando los procesos y dando valores iniciales a las variables globales, cuando es necesario.

3.3 Logica Temporal Lineal (LTL)

Una vez descrito brevemente el lenguaje de modelado Promela, en esta sección se resume el lenguaje para especificar propiedades LTL. Con este lenguaje, podemos escribir las propiedades que esperamos que cumpla nuestro sistema. La herramienta SPIN realizará la verificación del modelo frente a la propiedad para comprobar si surgen o no errores, tal y como se describió anteriormente.

Las fórmulas LTL se construyen utilizando un conjunto de fórmulas atómicas (que en nuestro caso son expresiones booleanas), los operadores lógicos habituales de lógica proposicional como son *and, or* o la negación lógica (¡), y los operadores modales propios de las lógicas temporales. De estos operadores, los tres que más relevantes que utilizaremos son *always*([]), *eventually*(<>) y *until*(U). Las fórmulas LTL se evalúan sobre las trazas de ejecución producidas por el modelo. Los dos primeros operadores son unarios ([] f, <>f), y su significado es que la expresión que los sucede (f) debe ser cierta, en el caso de always, en todos los estados de la traza que se está evaluando, mientras que en el caso de eventually, para que la fórmula <>f sea cierta es suficiente con que f sea cierta en algún estado futuro.

A diferencia de estos dos operadores, el operador until es binario (f U g). Su significado es que en algún momento futuro g debe ser cierta, y en todos los estados anteriores debe haber sido cierta g.

Los operadores de la lógica pueden anidarse arbitrariamente, lo que permite expresar propiedades muy complejas. En cualquier caso, dada una fórmula LTL f, y un modelo, se dice que el modelo satisface f si *todas* las posibles ejecuciones del modelo satisfacen la propiedad.

Por ejemplo, la propiedad <> ([] condición$_1$) expresa la propiedad de que, a partir de cierto punto en todas las ejecuciones de nuestro sistema, la condición$_1$ será siempre cierta.

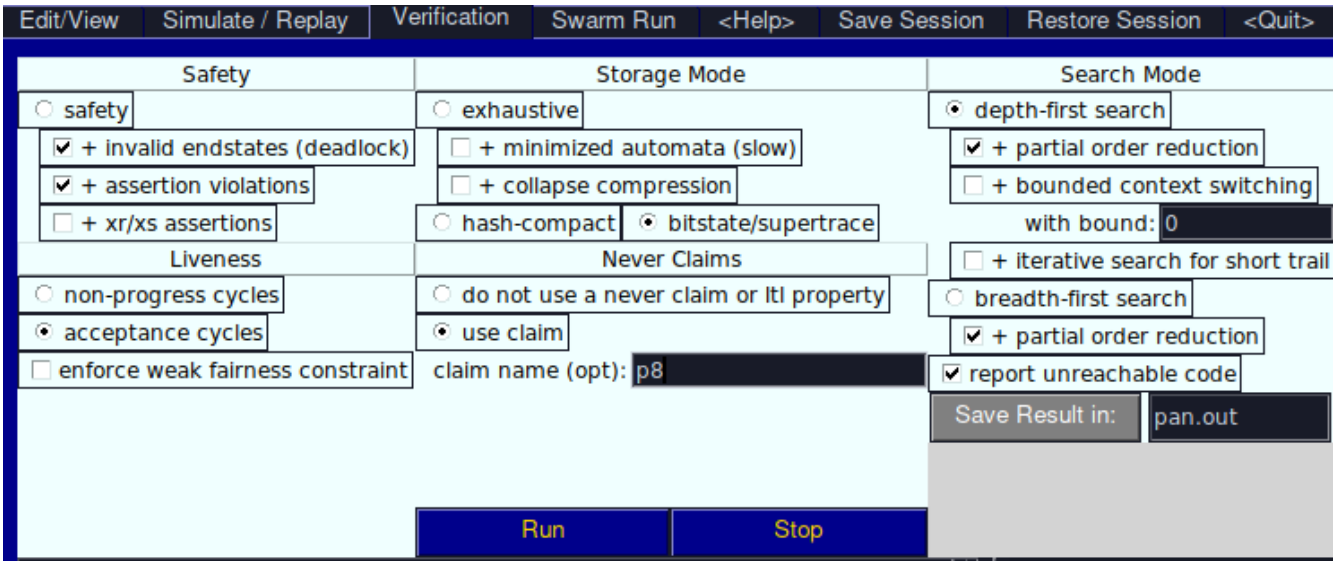
Otro aspecto interesante de las LTL es que, como ya se ha comentado antes, en caso la herramienta encuentre un comportamiento anómalo, que no se ajuste a la propiedad especificada, nos devolverá un contraejemplo en el que propiedad se viola. De este modo, podemos usar la herramienta también para probar propiedades existenciales (es decir, propiedades que se dan para algunas trazas, pero no para todas ellas). La forma más fácil de conseguir esto es escribiendo una propiedad LTL que exprese la *negación* de la propiedad existencial que queremos probar. Si la herramienta nos devuelve un contraejemplo, es la prueba de que existe la posibilidad de que dicho comportamiento se produzca.

3.4 La herramienta SPIN y su interfaz iSpin

En esta sección, describimos brevemente las funcionalidades de la herramienta SPIN, y de la interfaz iSpin que permite al usuario interaccionar con SPIN de una forma más cómoda. Como se ha descrito antes, SPIN toma como entrada un archivo .pml (un programa escrito en Promela) en el que se describe el sistema que queremos analizar. Por un lado, La herramienta permite simular el comportamiento del modelo y, por otro, permite verificar si satisface ciertas propiedades deseables. En esta sección, primero revisaremos la interfaz iSpin desde la que

podremos realizar la verificación. En la Figura 3.3.1 se muestran las distintas opciones disponibles, divididas en categorías.

A la izquierda de la figura aparece la opción *safety* (que en la figura aparece deshabilitada). Esta opción nos permite configurar la verificación para que se comprueben las siguientes propiedades de seguridad:



Interfaz de la verificación *Figura 3.3.1*

· **Ausencia de estados finales inválidos:** Esta opción equivale a analizar si el modelo está libre de bloqueo, que es una de las propiedades de seguridad más importantes de cualquier sistema concurrente. El bloqueo ocurre si en algún punto de la ejecución, alguno de los procesos que no ha terminado está suspendido en una instrucción no etiquetada como un estado final válido (con una etiqueta que empieza con end). Por ejemplo, esto puede pasar si un proceso se queda esperando una señal que necesita para seguir ejecutándose y ésta nunca llega.

Un ejemplo usado para ilustrar esta situación es el de la cena de los filósofos; sentados en una mesa circular tendremos a N filósofos con un número N de tenedores, dispuestos de forma que haya uno a cada lado de los filósofos. Para comer, un filósofo necesitará tener dos tenedores al

mismo tiempo. Si un filósofo tiene hambre, intentará coger ambos tenedores, y si solo tiene uno disponible, lo tomará y esperará pacientemente a que su compañero suelte el otro para usarlo. El bloqueo viene de la situación que se da cuando todos los filósofos cogen el tenedor que tienen a su derecha (o izquierda): todos se quedarán esperando que el filósofo que está a su izquierda (o derecha) suelte el tenedor para comer, pero dado que estos no sueltan los tenedores hasta que hayan comido, estarán bloqueados y por tanto no podrán comer.

Por este motivo, en los sistemas en los que varios procesos actúan de manera concurrente, es muy importante tener en cuenta sus interacciones y contemplar estas posibilidades para que, a la hora de trasladar el sistema hasta el mundo real, no se dé ninguna situación de bloqueo.

Violación de asertos: En nuestro programa podremos utilizar asertos que son instrucciones del tipo *assert(expresión lógica).* Si esta opción está habilitada y una ejecución evalúa a falso la expresión lógica, se obtendrá como un fallo.

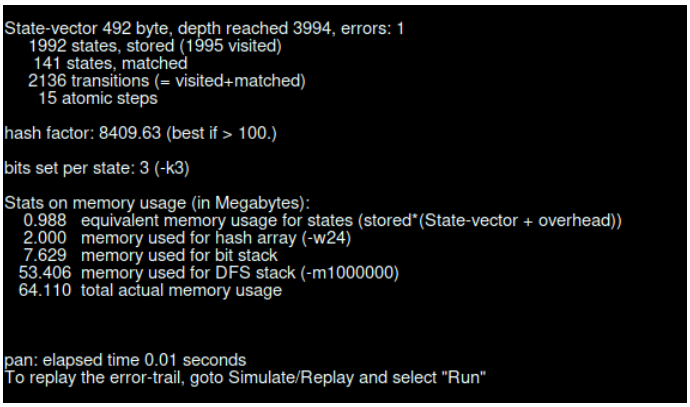
Asertos xr xs: Otra opción con la que contamos es la de definir algunos canales como de "sólo recepción" o "sólo envío" para un proceso dado, que serán xr y xs respectivamente. Esto nos sirve para comprobar si el canal definido es el único que envía o recibe información a través de ese canal, y en caso de no ser así, que se nos informe mediante un error.

Liveness: Esta opción nos permite analizar ciclos de aceptación en nuestro sistema. Los ciclos de aceptación están muy relacionados con las propiedades LTL que podemos analizar con la herramienta.

Modo de almacenamiento: Se referirá al modo que usará el programa para almacenar los estados que se den en el sistema. Aquí, el método exhaustivo es el que guarda cada estado de manera "bruta", podría decirse, mientras que las otras dos opciones hacen uso de técnicas de compresión de estados para optimizar el uso de la memoria del aparato en el que ejecutemos el análisis.

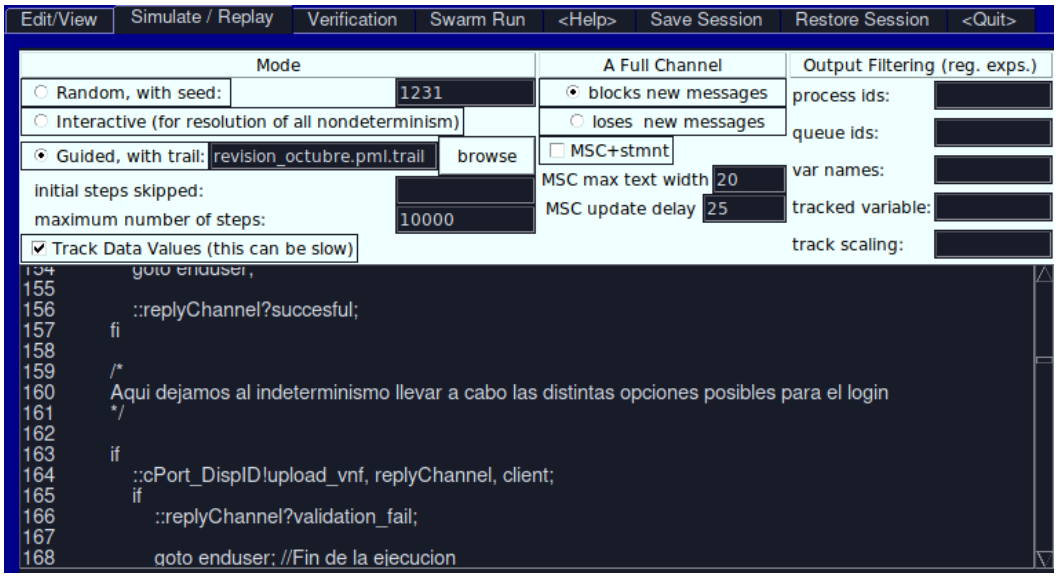
Never claims: Esta opción permite analizar las fórmulas LTL que hemos definido para que se evalúen sobre el sistema.

Método de búsqueda: Nos sirve para definir qué algoritmo de exploración se usará para ir avanzando a través de las ejecuciones. Podemos elegir entre búsqueda en profundidad y en amplitud.



Resultado de la verificación *Figura 3.3.2*

En la Figura 3.3.2 podemos observar el veredicto de la herramienta que se obtiene tras realizar una verificación. Lo más relevante es que podremos ver si el sistema contiene o no errores y la profundidad hasta la que llega la ejecución. En la figura, se muestra que la herramienta ha encontrado un error. La traza obtenida de la ejecución en la que se produce el fallo (el contraejemplo) se almacena en un archivo temporal, que podremos inspeccionar desde la ventana de simulación tal y como muestra la Figura 3.3.3. La pestaña de simulación nos permite realizar distintos tipos de simulaciones:



Interfaz de la simulación
Figura 3.3.3

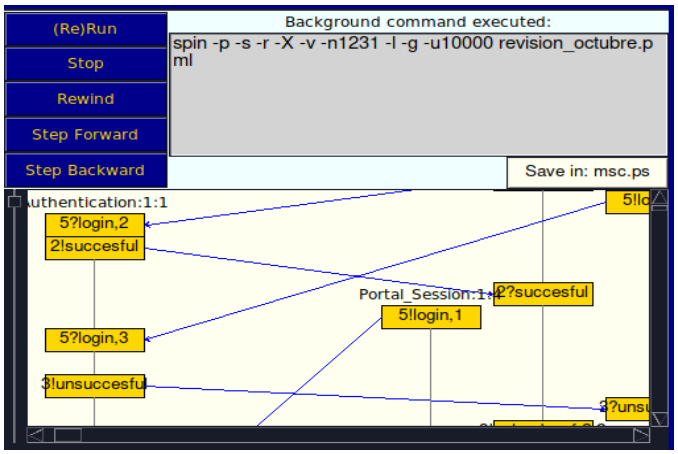
Simulación aleatoria: Esta simulación utiliza un valor como semilla para generar una traza aleatoria (una ejecución posible del sistema). Para una misma semilla, siempre obtenemos la misma simulación de tal modo que es posible volver a ejecutarla si detectamos algún error.

Simulación interactiva: Este tipo de simulación permite que el usuario guíe la ejecución. En cada punto en el que hay indeterminismo (es decir, varios caminos por los que la ejecución podría evolucionar) la herramienta nos muestra una ventana para que sea el usuario el que decida cómo continuar. Este método de simulación es interesante si queremos comprobar una sucesión de estados concreta para el sistema.

Guiada a partir de una traza: Esta simulación nos muestra la traza de error encontrada por la verificación. En general, esta opción es la que se utiliza más veces cuando se está analizando un sistema. Esta traza de error nos permite comprobar en qué estado se producido el fallo y qué camino se ha seguido hasta llegar al punto en el que se ha producido.

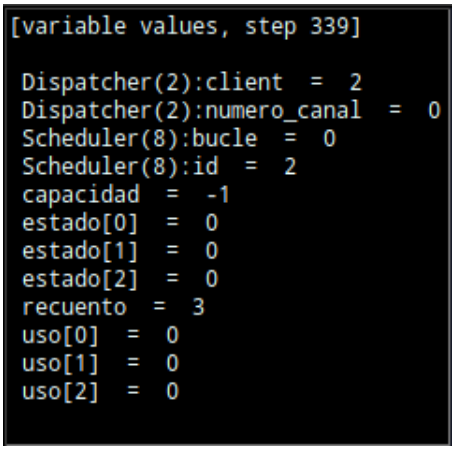
La herramienta permite seleccionar muchas otras opciones: definir el comportamiento del sistema cuando se envía un mensaje y el canal está lleno (suspender al proceso o descartar el mensaje), editar el número máximo de pasos a ejecutar y filtrar la información que se nos muestra en la consola. Esta consola irá mostrando la sucesión de estados del sistema, pero lo más útil es que, al darse un error, nos mostrará el estado específico en el que se encuentra cada proceso en ese momento, lo cual es vital para la depuración del sistema.

Además, la herramienta también muestra una representación gráfica (MSC) del sistema con los mensajes se han intercambiado los procesos (Figura 3.3.4). Desde ella podremos además parar la ejecución y moverla manualmente hacia adelante o hacia atrás, para así comprobar tanto el contenido de los canales como los valores de las variables en cada estado (Figura 3.3.5).



Representación gráfica de la simulación

Figura 3.3.4



Valores de las variables a lo largo de la simulación

Figura 3.3.5

La herramienta SPIN junto con los documentos que describen el comportamiento del sistema serán el punto de partida para modelar y analizar el sistema de gestión de experimentos desarrollados en el proyecto 5Genesis. A partir de los resultados aquí en este proyecto, se podrá extraer información crucial acerca de la corrección del sistema.

4

4.Metodología del trabajo

El proceso de desarrollo de este trabajo se divide en 5 fases que se describen brevemente, a continuación. El siguiente capítulo explica con detalle en el desarrollo, las pruebas, y los resultados de cada una de las fases aquí descritas. La metodología durante el desarrollo de este trabajo seguida podría catalogarse como incremental o en espiral.

4.1 Fase 0

La primera actividad fue un ejercicio intenso de estudio de la documentación del proyecto 5Genesis. Hubo que buscar y decidir cuáles eran los documentos clave para la realización del proyecto. En los documentos, el software que gestiona la realización de experimentos viene descrito como diagramas MSC (message sequence charts) que había que implementar en Promela.

Aunque la mayor parte de la información acerca del proyecto se encuentra en los entregables oficiales, fue necesaria la asistencia del compañero del grupo MORSE encargado de la implementación del módulo de experimentación, para aclarar muchos aspectos relativos al diseño del sistema que no quedaban claros en los entregables. Una vez comprendido el sistema, los procesos involucrados y sus interacciones, creamos un esqueleto con la funcionalidad básica de envío de mensajes entre cada proceso. Cabe destacar que aún no había ninguna lógica definida y que el sistema admitía como válida una muy amplia variedad de comportamientos que, más adelante, habrían de ser eliminados para que el modelo fuera fiel a la implementación.

4.2 Fase 1

En esta fase, se hizo un esbozo de las funcionalidades básicas que figuraban en los diagramas MSCs y probamos su funcionamiento.

A partir de este momento, al esqueleto básico se le fueron añadiendo funcionalidades hasta que se tuvo una versión inicial que simulaba un caso de uso básico consistente en un usuario que hace uso de la plataforma para ejecutar un experimento dado. Cuando esa versión fue totalmente funcional, pasamos a la siguiente etapa.

4.3 Fase 2

En la fase 2, extendimos la funcionalidad del sistema introduciendo la posibilidad de que varios usuarios accedan simultáneamente a la plataforma, lo que constituye una mejora sustancial con respecto al sistema implementado en la fase anterior.

El modelo construido en esta fase es una aproximación muy fiel del sistema real y es, por lo tanto, mucho más complejo. El hecho de que varios usuarios puedan acceder de manera concurrente a la plataforma hace posible multitud de posibles interacciones que complican el funcionamiento del sistema.

4.4 Fase 3

En esta fase, después de varias entrevistas con los miembros del equipo MORSE, quedó clara el papel de cada de uno de los procesos del sistema: los diferentes estados por los que pasaban, las estructuras de datos necesarias para almacenar esa información y el comportamiento detallado de los procesos (su sistema de transiciones). Toda esa información se incorporó al modelo, de modo que el modelo transformado permitía que los experimentos tengan su representación propia. Otro reto fue dotar al sistema de la noción de "tiempo", necesario para la realización del proyecto.

4.5 Fase 4

En esta última fase, se depuró el sistema, permitiendo comportamientos más fieles a la realidad, y mejorando algunos aspectos. También, se analizaron las propiedades deseables sobre el sistema para comprobar si la implementación realizada satisface los requisitos de calidad esperados.

5

5.Solución propuesta

En esta sección presentamos en detalle el comportamiento del gestor de experimentos de la plataforma 5G, así como la implementación realizada en el proyecto

5.1 Exposición de los diagramas, explicación de la plataforma e identificación de entidades y su función.

Como ya se ha comentado anteriormente, la primera tarea realizada en el proyecto fue el estudio de los deliverabables [2][3][4] del proyecto en los que se describe el comportamiento del gestor de experimentos. En esa primera fase, identificamos los ocho principales actores del sistema. A continuación, describimos la funcionalidad de cada una de estas entidades:

Usuario:

El proceso que simula las distintas peticiones que un usuario real le hará al sistema a través de la interfaz. Su comportamiento en una ejecución genérica es el siguiente:

1) Autenticación en el sistema ya que para acceder a la plataforma es necesario tener unas credenciales personales;
2) (opcional) Subir la llamada Virtual Network Function (VNF) nueva que, a efectos prácticos, constituye la configuración del experimento a realizar;
3) Seleccionar una de las configuraciones de experimentos y definir varios parámetros más.
4) Esperar el resultado del experimiento.

Portal:

El portal es la interfaz entre el usuario y el sistema. Según lo que el usuario quiera ejecutar, redirigirá su petición hacia el proceso que es capaz de realizar la petición deseada.

Autenticador:

Es el proceso que realiza la autenticación de los usuarios en el sistema.

Dispatcher:

El dispatcher es una capa de organización intermedia entre distintos elementos. Sus funciones son:

1) Enviar los VNF a un validador para comprobar si están bien definidas.
2) Recibir la configuración de una nueva ejecución en el sistema.

3) Pedir al scheduler la ejecución de los nuevos experimentos.

4) Si el scheduler lo aprueba, crear esos nuevos experimentos.

Validador:

El validador, en este sistema, se encarga de comprobar que todas las configuraciones nuevas que se introducen carecen de fallos.

Experiment lifecycle Manager:

Este proceso constituye la capa de la plataforma más cercana a los experimentos ejecutados. Toda la gestión de la realización de los experimentos se realiza en este proceso.

Management Layer:

Proceso que guarda las configuraciones y para que los usuarios puedan escogerlas antes de ejecutar un experimento.

Infraestructura:

Este proceso representa la infraestructura física sobre la que se ejecutarían los experimentos.

La siguiente figura muestra uno de los diagramas de secuencia, procedente del entregable 2.3 del proyecto 5Genesis.

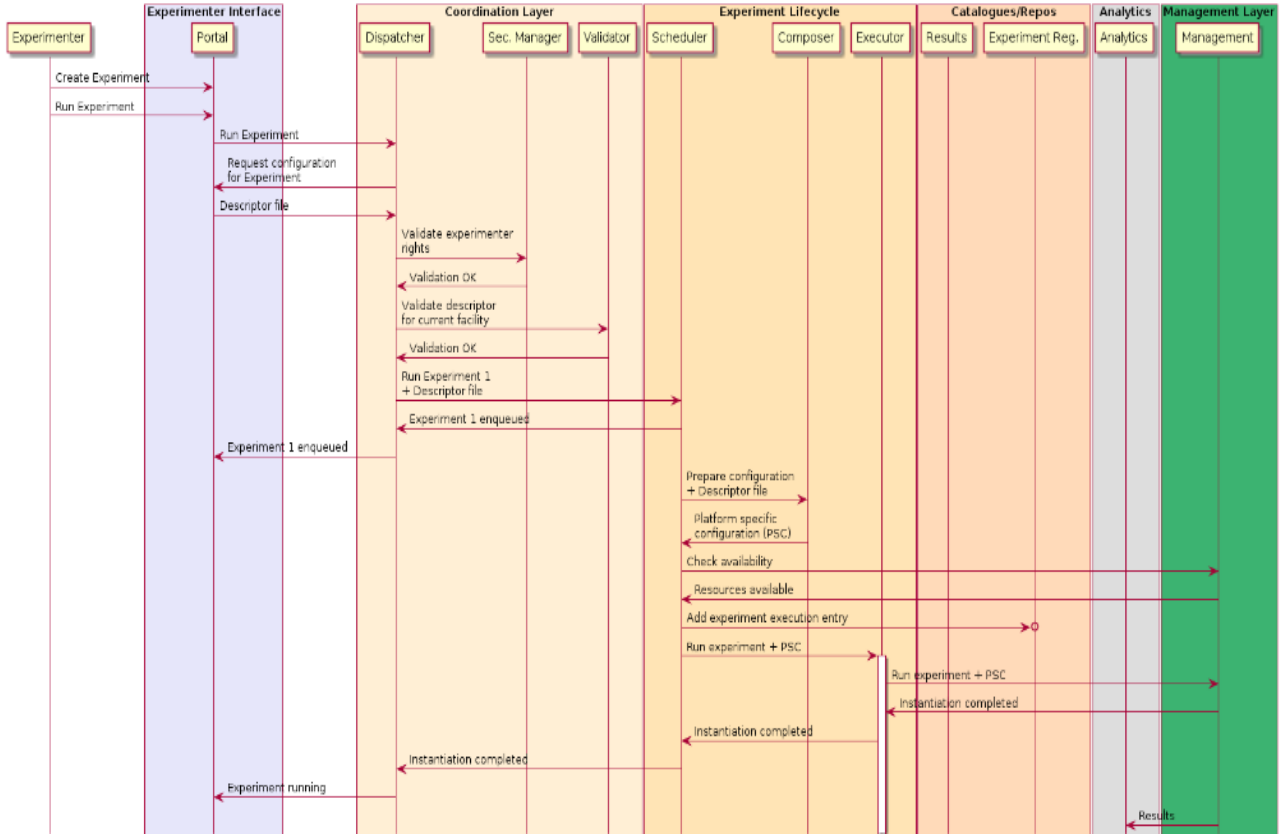


Diagrama de flujo para el uso de la plataforma por parte de un usuario simplificado

Figura 5.1.1

Dado que el proyecto, en esta primera fase, está en sus inicios, es importante notar que no todos los requisitos están bien definidos, así que, en algunos casos, se ha realizado una implementación ad-hoc de algunas funcionalidades.

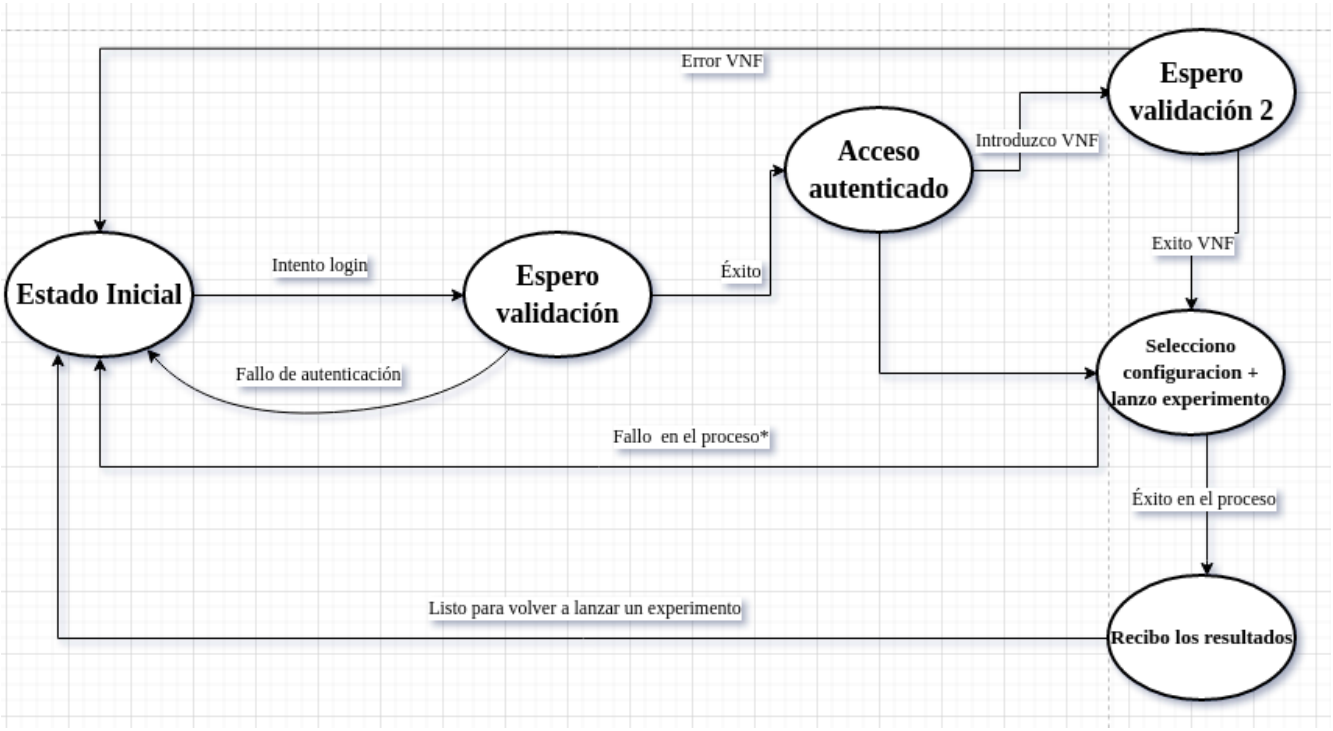
La primera implementación del sistema contiene todos los procesos mencionados anteriormente, que se comunicaban enviándose señales para obtener ejecuciones acordes con el diagrama presentado. Esas señales, sin embargo, no tenían ningún efecto en los procesos, simplemente se simulaba el flujo de ejecución descrito. Este fue el punto de partida, la arquitectura del sistema a más alto nivel, que luego se ha ido refinando hasta obtener un sistema final fiel al sistema real.

5.2 Máquinas de estado de los actores. Definición del sistema inicial. Pruebas, problemas y soluciones.

Una vez creado el esqueleto sobre el que formar el resto del sistema, el siguiente paso en el desarrollo fue definir de forma específica el comportamiento de cada elemento, de modo que, a partir de su implementación, se elaboraron máquinas de estado para cada elemento del sistema. Por otro lado, también fue relevante saber qué procesos del sistema tenían que ser implementados necesariamente y cuales no eran relevantes para simular el comportamiento de la plataforma.

A continuación, se muestran las máquinas de estado desarrolladas:

Usuario:

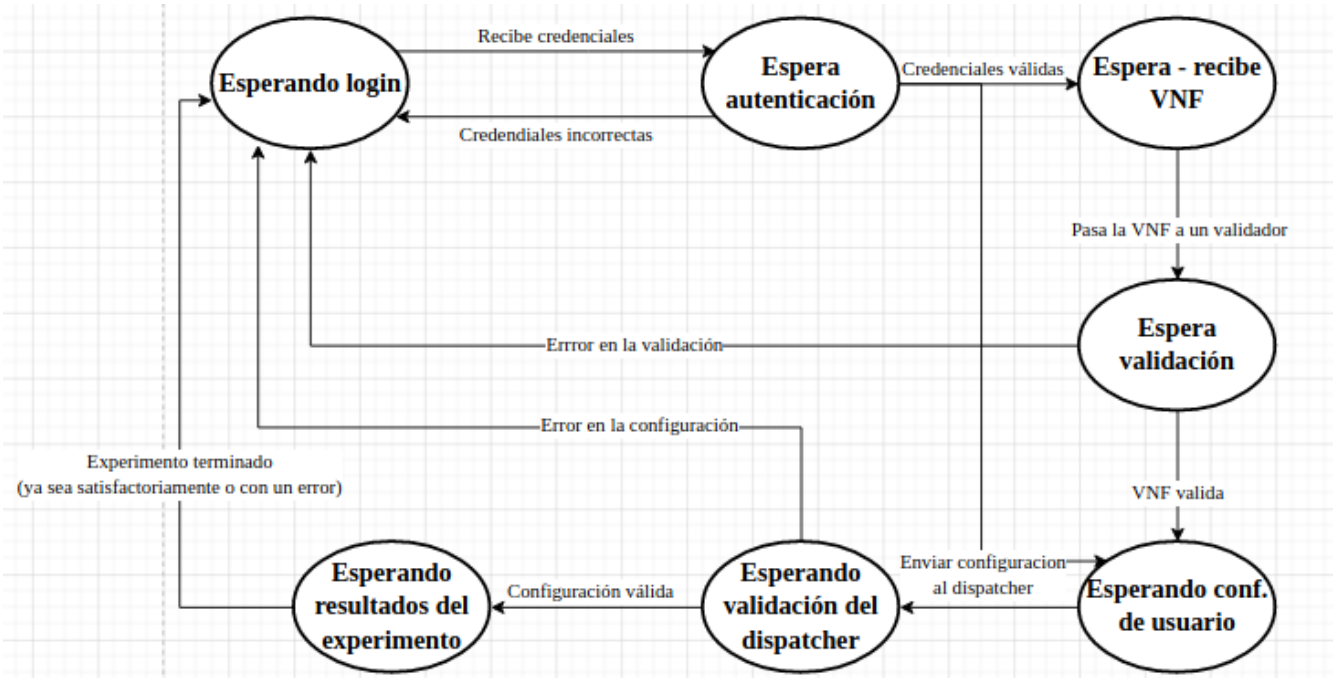


Máquina de estados del proceso Usuario

Figura 5.2.1

El usuario, para empezar a usar el sistema, debe iniciar una sesión válida. Una vez autenticado, podrá elegir entre crear una VNF propia o saltarse ese paso y elegir alguna de entre las que ya están disponibles en la plataforma. A continuación, recibirá los resultados con lo que podrá ver el estado de su experimento. La razón por la que, en caso de que se produzca un fallo en algún momento, se vuelve al estado inicial es porque en esta primera fase necesitábamos que el funcionamiento no se complicase demasiado introduciendo posibles bucles, como podría ser el caso de que un usuario siempre esté introduciendo credenciales erróneas.

Portal:

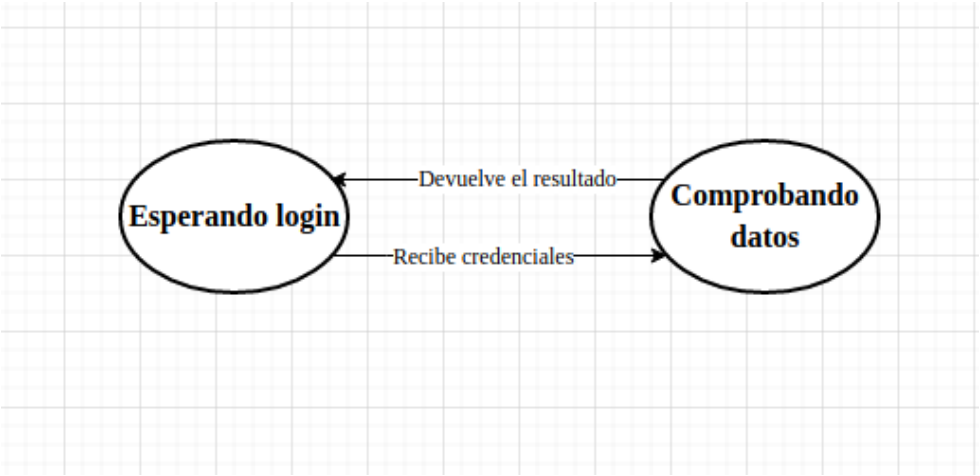


Máquina de estados del proceso Portal

Figura 5.2.2

El portal tiene un diagrama de flujo análogo al del usuario ya que es la interfaz a través de la cual el usuario interacciona con la plataforma. Después de esta fase, se llegó a la conclusión de que modelar al usuario y el portal era redundante y, en consecuencia, en las siguientes fases el proceso "Portal" se redefinió como "Sesión de portal", eliminando el proceso "Usuario" del sistema. Como se verá más adelante, esta simplificación mejoró el diseño del sistema, puesto que, aunque el portal se pueda considerar como una entidad independiente a la que acceden los distintos usuarios, realmente no comparten ningún recurso. Cada usuario está en una máquina diferente, solicitando experimentos diferentes y, por lo tanto, se puede suponer que constituyen sesiones separadas que interactuarán de forma independiente con el resto de los elementos.

Autenticador



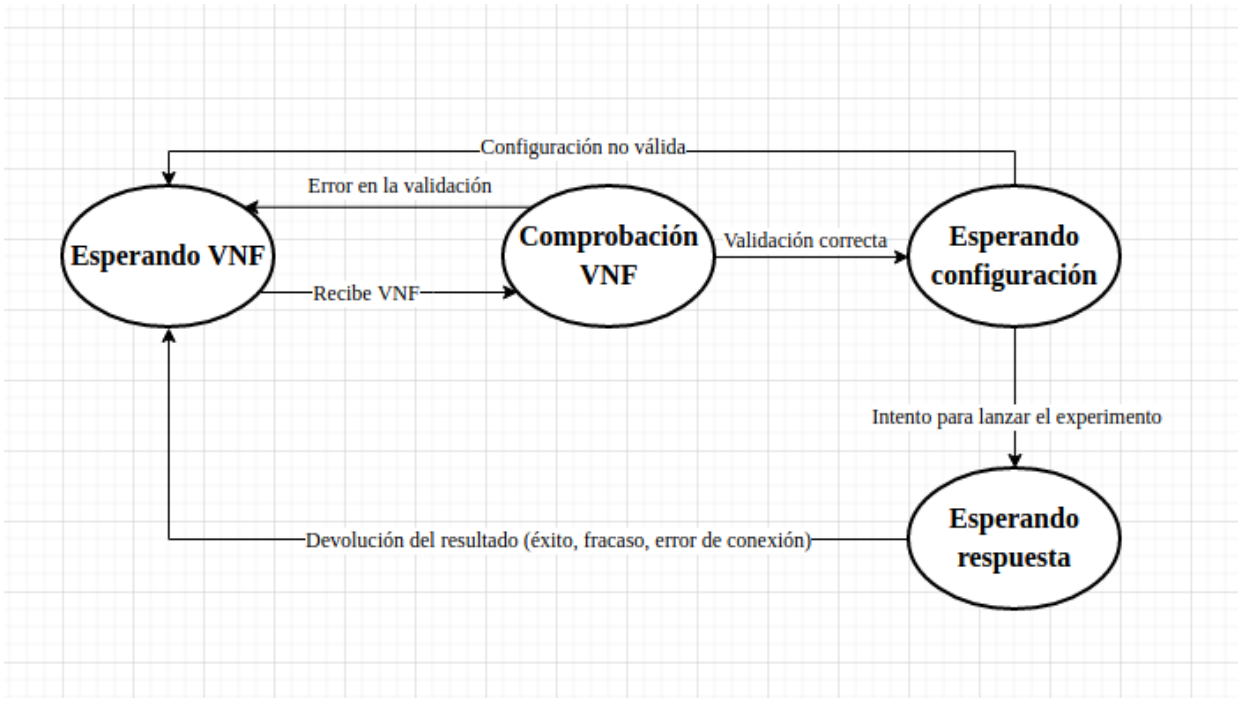
Máquina de estados del proceso Autenticador *Figura 5.2.3*

El proceso de autenticación al que se somete un usuario consiste únicamente en una comprobación, en la base de datos de la plataforma, de sus credenciales. Está claro que no tendría sentido para lo que queremos evaluar en este proyecto, modelar fielmente la autenticación real. Así que en nuestro modelo el autenticador simplemente considerará las

credenciales correctas o incorrectas de manera no determinista. Si hiciéramos un modelo más detallado, no añadiríamos ningún comportamiento nuevo, y complicaríamos innecesariamente nuestro sistema.

Dispatcher:

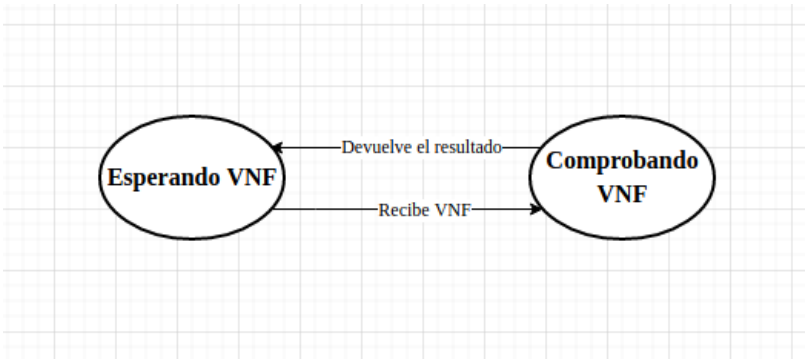
El siguiente diagrama muestra la naturaleza del dispatcher que, como se ha mencionado anteriormente, es un punto intermedio entre la capa más baja en la que se llevan a cabo los experimentos y la más alta en la que el usuario define sus parámetros. Su máquina de estados no tiene gran complejidad, pues de lo que se encarga es de que los mensajes lleguen de manera adecuada desde el origen hasta su destino. En caso de que la VNF escogida o los parámetros seleccionados provoquen un fallo, el dispatcher volverá a su estado inicial. Si se produce un error en la ejecución, su comportamiento es el mismo que si el experimento funcionase bien pues, en ambos casos, su función es transmitir al portal la información que le llega de la ejecución, ya sea de éxito o error.



Máquina de estados del proceso Dispatcher *Figura 5.2.4*

Validador

La funcionalidad del validador se parece a la del autenticador. Debe decidir si una configuración para una VNF es correcta. Para ello, en la implementación real debe tener en cuenta muchos parámetros que, además, tienen una serie de interacciones complejas con el resto del sistema. Por lo tanto, igual que se hizo anteriormente, el validador dará como correcta o incorrecta la configuración de la VNF de manera no determinista, lo que será suficiente para modelar cualquier comportamiento del sistema real. La máquina de estados del Validador aparece a continuación.

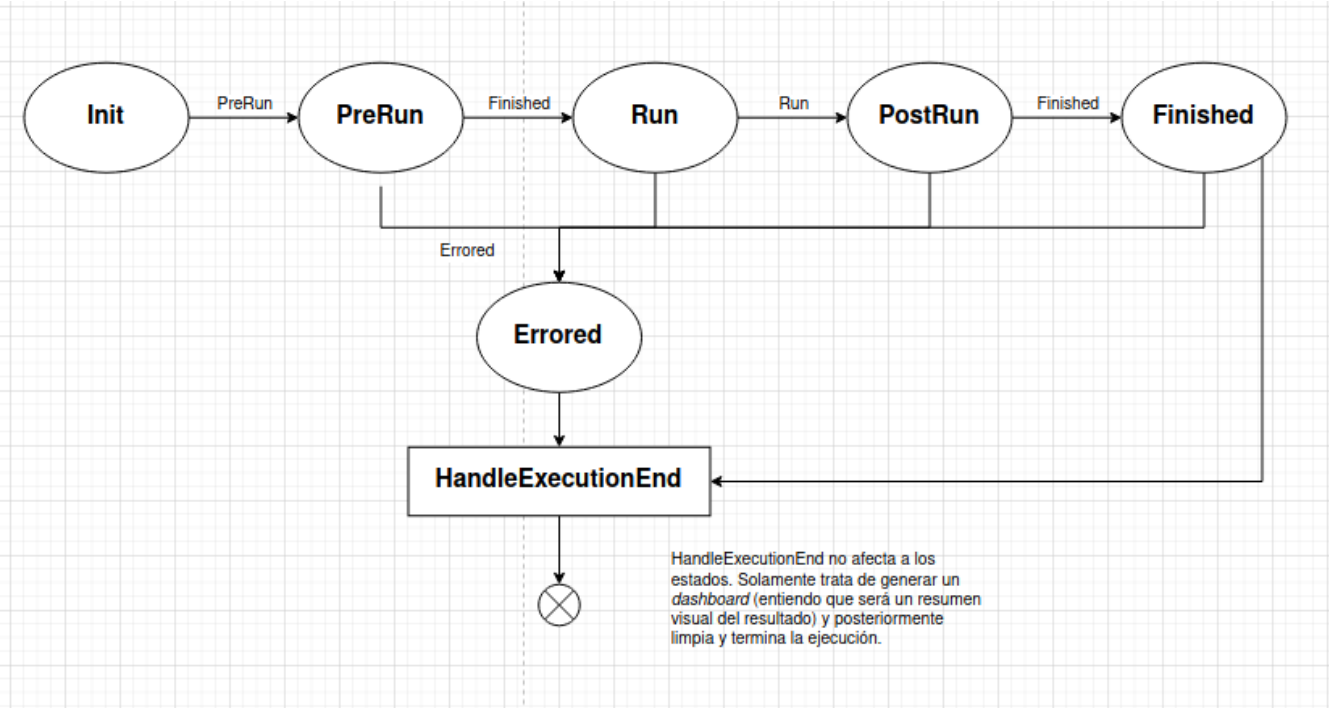


Máquina de estados del proceso Validador
Figura 5.2.5

ELCM

En esta fase, la funcionalidad del Experiment Lifecycle Manager está limitada a representar el estado de cada experimento durante su ejecución. Este actor será el que se refinará introduciendo distintos comportamientos en las siguientes versiones del modelo para lograr un sistema lo más fiel al sistema real posible.

A continuación, se describe brevemente cada uno de los estados que están representados en la máquina de estados que se muestra en la Figura:



Máquina de estados del proceso ELCM *Figura 5.2.6*

Init:

Un experimento se encuentra en este estado justo después de haber sido añadido a la cola. Es un estado común para todos los experimentos.

PreRun:

En este estado, se realizan una serie de actividades necesarias antes de lanzar el experimento. Incluye las tareas de inicialización y preparado para ejecutarse en la plataforma.

Run:

Los experimentos que se encuentran en este estado están ejecutando la tarea solicitada por los usuarios.

PostRun:

En este estado se procesa la información recogida durante la ejecución del experimento y se preparan los datos relevantes para el usuario que lo inició.

Finished:

El experimento ha terminado y se ejecutan tareas de finalización y liberación de recursos.

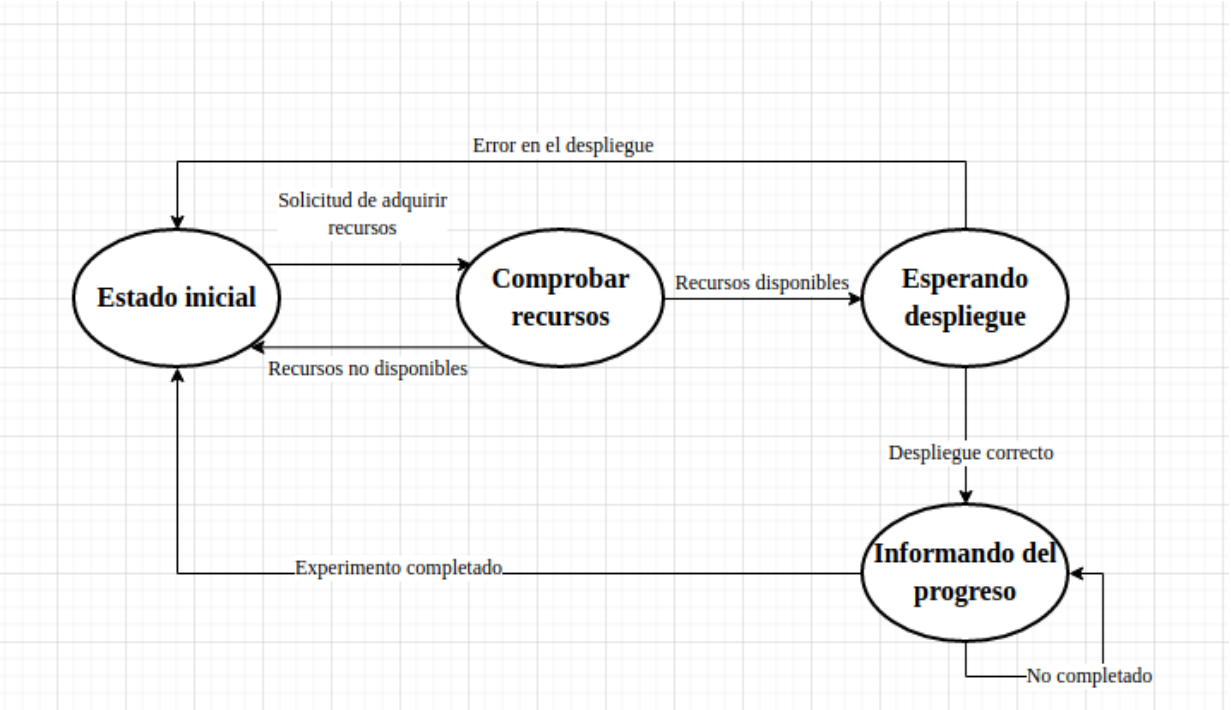
En cada uno de los distintos estados del experimento, es posible que se produzca un fallo, si algo no funciona como debería. Sea cual sea el estado en el que éste se produzca, el experimento pasará a un estado de error, que será manejado por el sistema para que cese el experimento y se elimine de la cola, informando de que la ejecución acabó con un error.

Más adelante, se decidió que sería más interesante ver la evolución de los experimentos durante las simulaciones del modelo, pues este proceso es esencial en la plataforma.

Infraestructura

En esta fase, el controlador de la infraestructura se define para que cuando se lance un experimento, se comprueben los recursos y, en caso de estar disponibles, avise al manejador de experimentos para que lo ejecute. A partir de este punto, el estado de cada experimento que tiene lugar en la plataforma se monitoriza, enviando al usuario que ejecuta el experimento cualquier información relevante de vuelta.

En esta fase, se modeló este proceso de forma no muy precisa, implementando comportamiento más natural. La siguiente figura muestra la máquina de estados de la infraestructura.

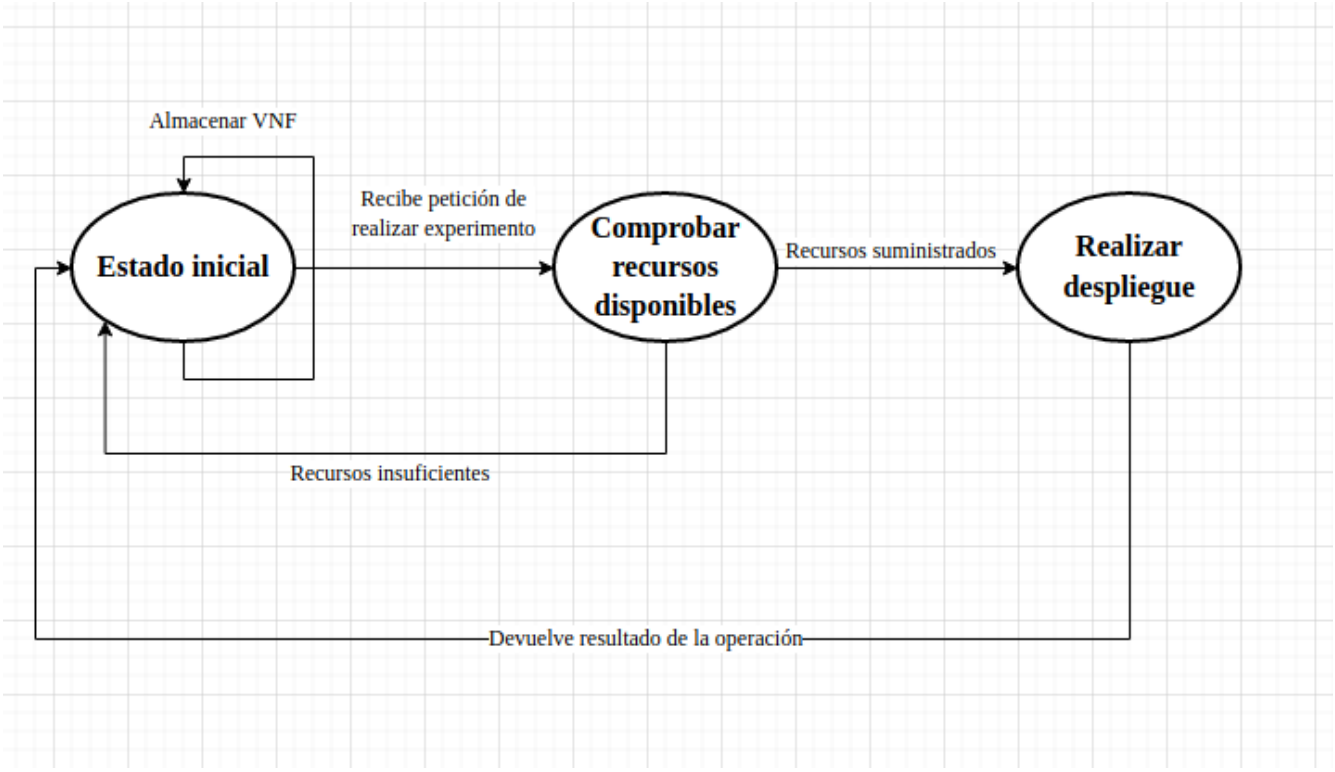


Máquina de estados del proceso Infraestructura

Figura 5.2.7

Management:

Como se ha comentado antes, este proceso es el encargado de gestionar los recursos a más bajo nivel, de ocuparse de los despliegues en la plataforma y almacenar los datos. Como en el caso anterior también, en la primera fase, se hizo un modelo conceptual a partir del comportamiento que se describe entregables. La máquina de estados de este proceso se muestra en la siguiente figura.



Máquina de estados del proceso Management.

Figura 5.2.8

A partir de estas máquinas de estado, el código implementado tuvo un comportamiento similar al del sistema real. Sin embargo, aún quedaban por definir algunos detalles, como decidir cuál sería la representación del espacio disponible. En esta fase, lo principal era que el comportamiento del modelo se ajustase lo más posible al sistema.

En esta primera fase, el modelo suponía la existencia de un único usuario que hacía distintas solicitudes. En la siguiente fase, el modelo se extendió asumiendo la existencia de varios usuarios.

5.3 Extensión con varios usuarios. Problemas y soluciones encontrados

La extensión del modelo a varios usuarios que hacen uso de la plataforma simultáneamente no es una tarea trivial. El primer modelo, al tener solo un usuario, no necesitaba diferenciar los mensajes, puesto que iban todos en secuencia. Sin embargo, si varios experimentos se están ejecutando simultáneamente, es necesario distinguir el estado por el que va cada experimento ya que, en otro caso, la ejecución de un experimento podría interferir en la de otro. Además, como el proceso del ELCM en el primer modelo estaba implementado asumiendo un usuario, si hay varios, es necesario una instancia del ELCM por usuario. Esto complica el modelo porque la comunicación entre procesos, que implementamos utilizando canales de comunicación, ha de adaptarse al nuevo escenario.

Para resolver el problema de la comunicación entre procesos, se hizo obligando a que cada sesión en el portal crease un canal de respuesta distinto, que se envía junto al resto de información en los mensajes que ya teníamos en el primer modelo. Así, cuando es necesario que una respuesta llegue a un proceso dado, se usará el canal de respuesta que dicho proceso envió como suyo. Esta solución hizo que el código tuviera que ser cuidadosamente modificado.

Cuando la plataforma está atendiendo a varios usuarios simultáneamente es posible que un usuario envíe un mensaje a un proceso y que éste esté ocupado en ese momento, de forma que el mensaje debe almacenarse temporalmente. De esta forma, los mensajes pueden estar desordenados en el canal de entrada al proceso receptor, de manera que, si se leen en el orden incorrecto, el sistema puede bloquearse.

Si bien en esta fase el contenido y comportamiento del sistema no se vio muy afectado, los cambios con respecto a la estructura interna del código fueron significativos.

5.4 Modelo final

En esta sección, se describe el modelo final implementado, así como la verificación de las propiedades de seguridad y viveza analizadas sobre el código.

La última versión del modelo se realizó a partir de la versión previa, pero con cambios muy importantes. El ELCM era uno de los procesos más relevantes a la hora de llevar a cabo los experimentos y su comportamiento era muy limitado en las primeras versiones. Tras discutir con los implementadores del sistema real se pudo llegar a un modelo muy cercano descrito en Promela y que, por lo tanto, puede ser analizado por la herramienta SPIN.

El proceso ELCM se dividió en varios procesos y estructuras independientes. El cambio más relevante con respecto a la versión anterior es que ahora los cada experimento se representa con un proceso llamado *executor* (ejecutor), que se encarga de guardar su estado (los mismos que ya se definieron anteriormente para el ELCM). Todos los ejecutores activos están almacenados en una estructura FIFO, y es el proceso scheduler el que se encarga de la gestionarlos para que la estructura sea siempre consistente durante la ejecución del sistema.

Además, como cada cierto tiempo, el scheduler tiene que actualizar el estado de cada ejecutor de la cola de ejecución, fue necesario introducir una representación del tiempo en el modelo, lo que no es sencillo. En el modelo, lo que se hizo fue crear un proceso que envía un mensaje periódicamente, como si fueran latidos. Esto simula una parte del sistema real en el que cada 10 segundos se hace esto mismo: enviar una señal al scheduler para que actualice cada elemento de la cola.

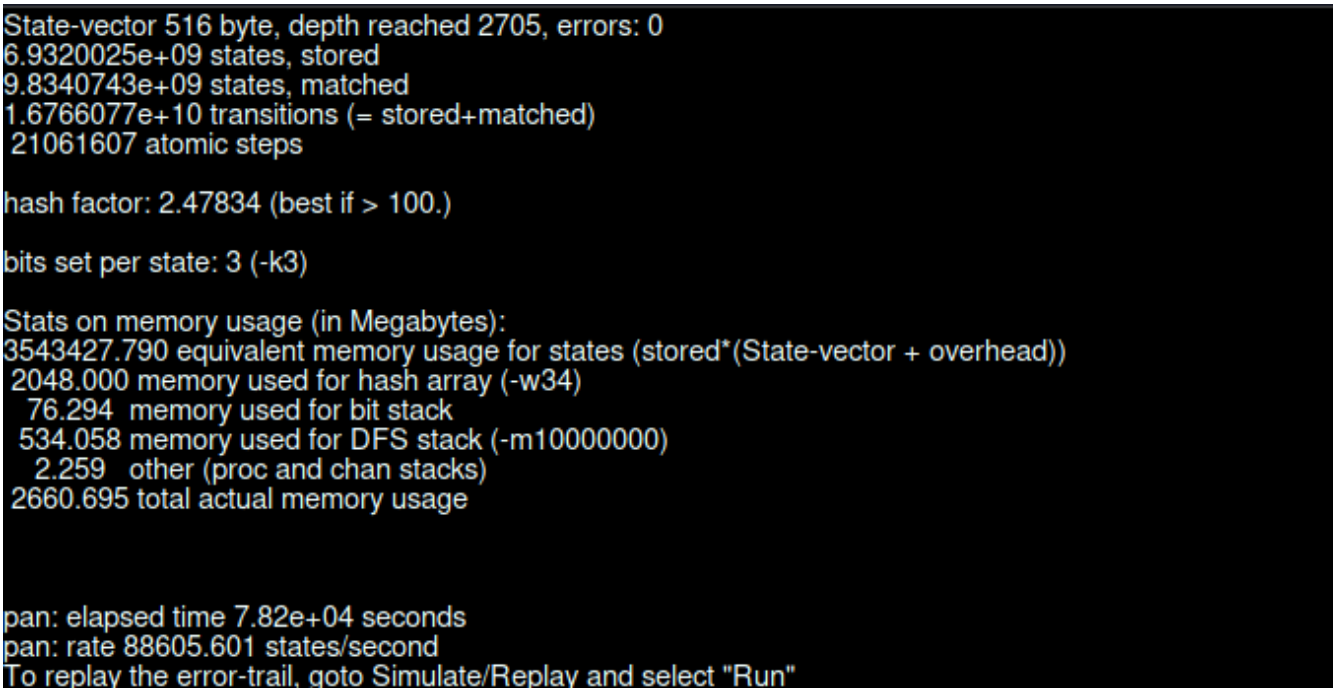
La introducción del tiempo simulado en Promela conlleva varias complicaciones debido a la semántica por entrelazado que, por defecto, tiene implementado SPIN. Como el proceso que simula el tiempo lo hace de forma indeterministas en algunas simulaciones los canales se llenan con lo que algunos mensajes pueden perderse. En el modelo final, evitamos este comportamiento no deseable, haciendo que los latidos solo se enviasen bajo unas condiciones específicas.

En el nuevo modelo, se utiliza un array de canales, siendo cada una de sus posiciones un canal distinto, de forma que cuando un nuevo proceso se lanza, éste intentará tomar para sí una posición del array, y de esa forma consigue acceso a uno de los canales disponibles. Cuando termine su ejecución, dejará libre el canal para otros procesos. Con esta implementación, además de resolver eficientemente los problemas de comunicación que había en versiones anteriores, nos permitió representar la capacidad física del sistema para admitir nuevos experimentos. De esta forma, los procesos que se realizaban esta tarea previamente se eliminaron.

El modelo obtenido con estas modificaciones permite simular fielmente el comportamiento de la plataforma real. El modelo genera una gran cantidad de simulaciones distinta. Por ejemplo, para analizar que el sistema no bloquea se utilizaron 8GB de memoria RAM.

A continuación, describimos el trabajo de verificación realizado sobre el sistema.

Inicialmente, se analizó que el sistema no bloquea con un modelo con cuatro usuarios. pero el factor de hash, que idealmente debe de estar por encima de 100, rondaba el valor de 3. De modo que incrementamos el tamaño del espacio de estados estimado de 1000 a 10.000.000. Después de 24 horas se pausó la verificación con este resultado.

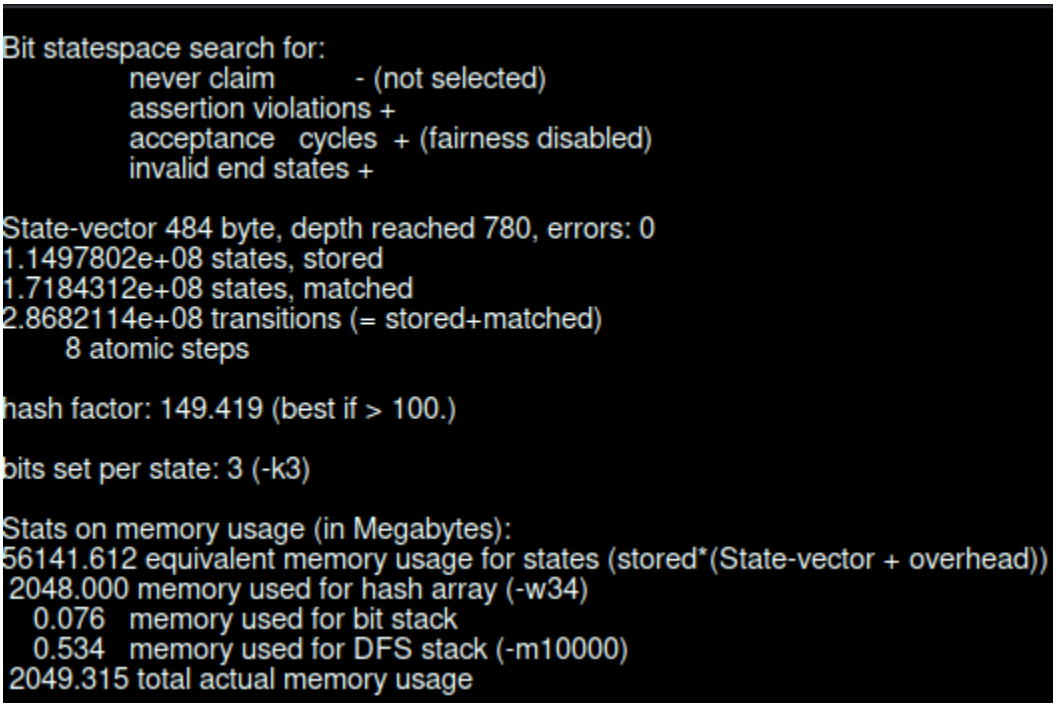


Resultados de la verificación con 3 usuarios concurrentes y cola de 3 espacios

Figura 5.4.1

Si bien no se encontró ningún fallo, tras 78200 segundos y aún con un factor de hash bajo, llegamos a la conclusión de que era necesario simplificar de alguna forma el sistema a representar.

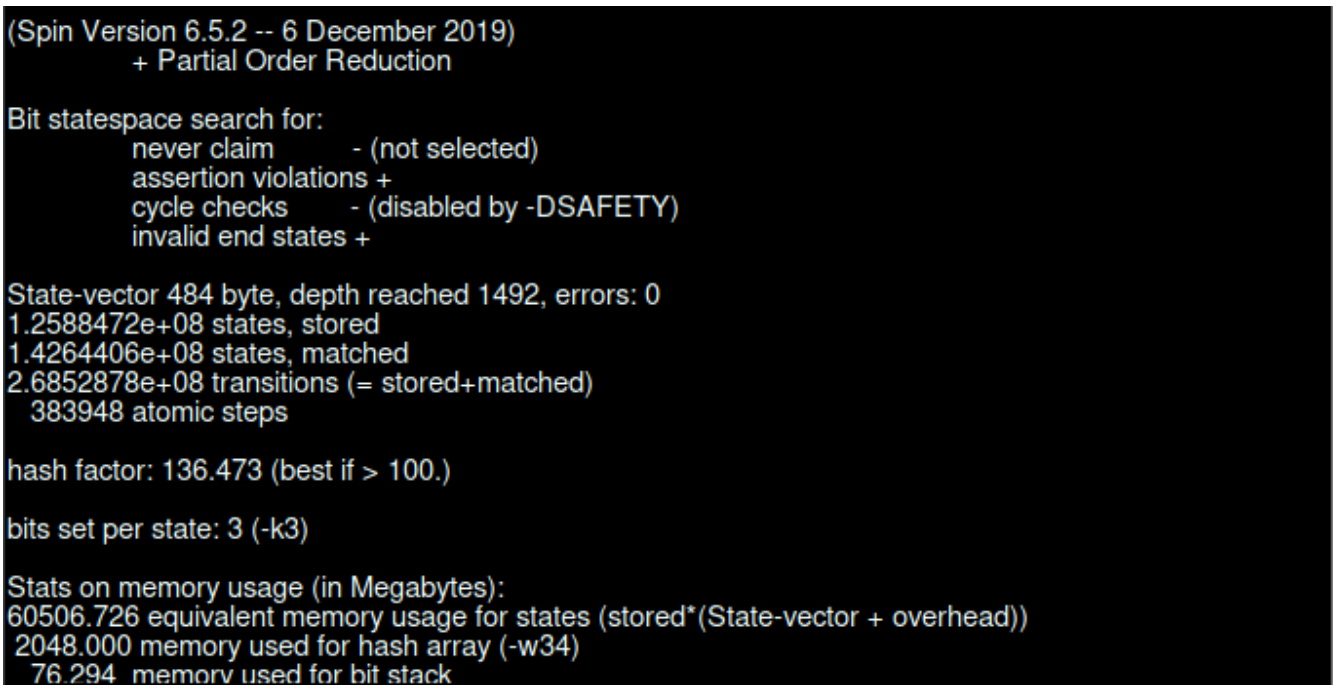
Tras ajustar el sistema para que se den las mismas condiciones que las que aquí queríamos probar, se simplificó el comportamiento de los ejecutores de forma que no se alterara esencialmente el comportamiento del modelo. Con esta simplificación, en un tiempo mucho más razonable y un factor de hash aceptable, el resultado al acabar la verificación fue el siguiente:



Resultados de la verificación con 3 usuarios concurrentes y cola de 2 espacios

Figura 5.4.2

Para comprobar que, en el caso anterior no había ningún problema, se introdujeron esta vez menos usuarios simultáneos para el mismo sistema original y se obtuvieron los resultados satisfactorios que se muestran en la siguiente figura.



Resultados de la verificación con 2 usuarios concurrentes y cola de 3 espacios

Figura 5.4.3

5.5 Prueba de propiedades sobre el modelo

En esta sección, presentamos las propiedades que se han analizado sobre el modelo. Como ya se indicó en la Sección 3.3, las propiedades se especifican en la lógica temporal lineal LTL utilizando la sintaxis que acepta la herramienta SPIN.

1) *ltl p1 { [] (capacidad <= 3 && capacidad >=0) }*

La primera propiedad analizada, p1, es una propiedad de seguridad que especifica que el número de experimentos activos en el sistema nunca será mayor que 3 ni menor que 0. Se ha escogido el valor 3 porque es el tamaño máximo de la estructura FIFO de ejecutores del sistema. Una FIFO más grande crea modelos muy pesados de analizar.

La siguiente figura muestra el diagnóstico de SPIN tras evaluar la propiedad p1. Como puede observarse, el modelo la satisface.

```
spin -a  Tfg5Genesis.pml
ltl p1: [] (((capacidad<=3)) && ((capacidad>=0)))
gcc -DMEMLIM=1024 -O2 -DBITSTATE -DXUSAFE -w -o pan pan.c
./pan -m10000  -k3 -a -w20
Pid: 81449

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Bit statespace search for:
        never claim         + (p1)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 572 byte, depth reached 4239, errors: 0
  572534 states, stored
 1079136 states, matched
 1651670 transitions (= stored+matched)
   41328 atomic steps
```

Evaluación de la propiedad p1 *Figura 5.5.1*

2) *ltl p2 { [] ((estado[0]==0)||(estado[0]==1)||(estado[0]==2)||(estado[0]==3)||(estado[0]==4))}*

La propiedad p2 es también una propiedad de seguridad. En este caso, queremos comprobar que el primer ejecutor (el que tiene id 0) tiene siempre un estado válido. Los estados de los ejecutores se representan en el modelo con los valores del 0 al 4, que representan Init, PreRun, Run, PostRun y Finished, respectivamente. Como en caso de que ocurra un error, el ejecutor correspondiente se libera, hemos considerado que la transición a Errored hace que su estado vuelva al valor 0 liberando su espacio (y por tanto, su espacio en el array de canales en uso también se libere).

Como en el caso anterior, la siguiente figura muestra la respuesta de SPIN tras analizar la propiedad, concluyéndose que se satisface.

```
spin -a  Tfg5Genesis.pml
ltl p2: [] ((((((estado[0]==0)) || ((estado[0]==1))) || ((estado[0]==2))) || ((estado[0]==3))) || ((estado[0]==4))
)
gcc -DMEMLIM=1024 -O2 -DBITSTATE -DXUSAFE -w -o pan pan.c
./pan -m10000  -k3 -a -w20
Pid: 82855

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Bit statespace search for:
        never claim         + (p2)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 572 byte, depth reached 4239, errors: 0
  572534 states, stored
 1079136 states, matched
 1651670 transitions (= stored+matched)
   41328 atomic steps
```

Evaluación de la propiedad p2 *Figura 5.5.2*

3) ltl p3 { [] (capacidad != 3)}

La propiedad p3 es una propiedad existencial que no puede probarse directamente en SPIN, ya que la herramienta asume que las propiedades son universales, es decir, que deben satisfacerse en todos los posibles comportamientos del modelo. En este caso, buscamos alguna ejecución del modelo en la que la variable capacidad tome el valor 3. Para ver si esto posible, la metodología habitual es escribir la propiedad negada, es decir, especificar que en todas las ejecuciones y en todos los estados, la variable capacidad nunca vale 3. Si la herramienta nos devuelve un contraejemplo, eso significa que en alguna de las ejecuciones del sistema se utiliza la capacidad completa del canal. Esta propiedad es interesante porque significa que, en algunas ejecuciones, se hace uso de todos los recursos que disponibles en la plataforma.

A diferencia de los ejemplos anteriores, en la siguiente figura se muestra la respuesta de SPIN, que da un contraejemplo, que es el que prueba que la propiedad deseada (hay alguna ejecución en la que la variable capacidad es 3) se satisface.

```
pan:1: assertion violated  !( !((capacidad!=3))) (at depth 2232)
pan: wrote Tfg5Genesis.pml.trail

(Spin Version 6.5.2 -- 6 December 2019)
Warning: Search not completed
        + Partial Order Reduction

Bit statespace search for:
        never claim         + (p3)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 556 byte, depth reached 2232, errors: 1
    1108 states, stored
     127 states, matched
    1235 transitions (= stored+matched)
      18 atomic steps
```

Evaluación de la propiedad p3 *Figura 5.5.3*

```
capacidad  =  3
estado[0]  =  3
estado[1]  =  3
estado[2]  =  0
recuento  =  3
uso[0]  =  1
uso[1]  =  1
uso[2]  =  1
```

Contraejemplo para la propiedad p3

Figura 5.5.4

ltl p4 { [] !(estado[0]==3 && estado[1]==3 && estado[2]==3)}

La propiedad p4, al igual que la propiedad p3, es una existencial. Describe el hecho de que el sistema soporta situaciones de verdadera concurrencia en la que los tres ejecutores progresan simultáneamente. Como en el caso de p3, la fórmula se define negando lo que se quiere ver si es posible, es decir, que en ningún estado los 3 ejecutores están en el estado PostRun a la vez. De

ese modo, si la herramienta devuelve un contraejemplo, podemos estar seguros de que tres usuarios pueden estar ejecutando experimentos de forma concurrente.

```
spin -a  Tfg5Genesis.pml
ltl p4: [] (! ((((estado[0]==3)) && ((estado[1]==3))) && ((estado[2]==3))))
gcc -DMEMLIM=2024 -O2 -DBITSTATE -DXUSAFE -w -o pan pan.c
./pan -m100000  -k3 -a -w20
Pid: 154220
pan:1: assertion violated  !( !( !((((estado[0]==3)&&(estado[1]==3))&&(estado[2]==3))))) (at depth 2275)
pan: wrote Tfg5Genesis.pml.trail

(Spin Version 6.5.2 -- 6 December 2019)
Warning: Search not completed
        + Partial Order Reduction

Bit statespace search for:
        never claim         + (p4)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 524 byte, depth reached 2275, errors: 1
    1131 states, stored
     126 states, matched
    1257 transitions (= stored+matched)
      15 atomic steps
```

Evaluación de la propiedad p4 *Figura 5.5.5*

```
capacidad  =  3
estado[0]  =  3
estado[1]  =  3
estado[2]  =  3
recuento  =  2
uso[0]  =  1
uso[1]  =  1
uso[2]  =  1
```

Contraejemplo propiedad p4

Figura 5.5.6

ltl p5 { [](estado[0]==0 U (estado[0]==1) || estado[0]==1 U (estado[0]==2 || estado[0]==0)|| estado[0]==2 U (estado[0]==3 || estado[0]==0)||estado[0]==3 U (estado[0]==4 || estado[0]==0)||estado[0]==4 U estado[0]==0)}

Finalmente, la propiedad p5 describe si los cambios de estados de los ejecutores se realizan de forma correcta. Así, las únicas transiciones posibles para un ejecutor son, o bien avanzar, es decir, de 0 a 1, de 1 a 2 y así hasta llegar al estado Finished representado por el valor 4, o bien que se produzca un error y que vuelva al 0. Así pues, la fórmula establece que un estado cualquiera deberá seguir siendo ese mismo hasta, o bien pasar a ser su estado siguiente correcto, o bien tener un error y volver al 0.

```
spin -a  Tfg5Genesis.pml
ltl p5: [] (((((((estado[0]==0)) U ((estado[0]==1))) || (((estado[0]==1)) U (((estado[0]==2)) || ((estado[0]==0
))))) || (((estado[0]==2)) U (((estado[0]==3)) || ((estado[0]==0))))) || (((estado[0]==3)) U (((estado[0]==4))
|| ((estado[0]==0))))) || (((estado[0]==4)) U ((estado[0]==0))))
gcc -DMEMLIM=2024 -O2 -DBITSTATE -DXUSAFE -w -o pan pan.c
./pan -m100000  -k3 -a -w20
Pid: 156096

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Bit statespace search for:
        never claim         + (p5)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 508 byte, depth reached 2558, errors: 0
  532292 states, stored
  767213 states, matched
 1299505 transitions (= stored+matched)
   15207 atomic steps
```

Evaluación de la propiedad p5 *Figura 5.5.7*

6

6.Conclusiones y líneas futuras

En este trabajo se ha modelado y analizado el sistema que gestiona la realización de los experimentos en la plataforma 5Genesis. El modelo se ha construido utilizando el lenguaje Promela, para que posteriormente pueda ser analizado por el model checker SPIN. Las propiedades analizadas se han especificado utilizando la lógica temporal LTL.

A pesar de la complejidad del sistema, que contiene muchos procesos que interactúan de forma concurrente enviándose mensajes, a través de canales, el proyecto ha demostrado la capacidad de las herramientas de model checking para verificar propiedades complejas sobre modelos complejos. En cualquier caso, como es bien sabido, debido a la limitación de memoria ha sido necesario acotar el número de usuarios que utilizan simultáneamente la herramienta.

Como trabajo futuro, se plantea el uso de instrucciones incorporadas en las últimas versiones de Promela para incrustar código C. Este código embebido tiene la ventaja de que, aunque el model checker lo ejecuta no tiene por qué guardar su estado en el vector de estados, lo que amplía considerablemente el tamaño de los modelos que se pueden analizar.

Otra línea futura de trabajo es utilizar los autómatas temporizados, que son autómatas que contienen relojes que evolucionan de manera síncrona con el tiempo, y herramientas como Uppaal [9] para incorporar el tiempo en el modelo, de manera que puedan probarse propiedades que tienen que ver con el tiempo.

Este trabajo ha sido sin duda un desafío, por el uso de un lenguaje de menos conocido y muy distinto a los que se estudian en nuestro grado. No se suele tener en cuenta, la cantidad de recursos a la que se puede acceder para aprender con detalle una nueva tecnología, de manera que nos ayude a la solución de problemas. Muchas veces no somos conscientes de lo afortunados que somos por el hecho de poder consultar problemas similares que otras personas tuvieron antes que nosotros.

I

Bibliografía

[1] Ben Ari, M. *Principles of the Spin Model Checker.* Israel, 2008.

[2] A. Díaz Zayas, B. García, P. Merino, A. Brunstrom, G. Caso, M. Emmelmann, R. Shrestha, Mikhail Smirnov, A. M. C Bosneag, E. Aumayr, J.O. Fajardo, E. Atxutegi, V. Koumaras, G. Theodoropoulos, I. Pretel, A. Pineda, Jesús Gutiérrez, D. Tsolkas, G. Xilouris, H. Koumaras, T. Anagnostopoulos, G. Gardikis. "Initial planning of tests and experimentation", H2020, D2.3, 2019.

[3] A. Díaz Zayas, Universidad de Málaga, L.M. Ericsson Limited, Cosmote, National center for scientific research "Demokritos", Athonet SRL, Eurocom, Karlstads universitet), Simula metropolitan center for digital engineering, Atos Spain SLC, Space Hellas (Cyprus) Ltd., Fraunhofer gesellschaft zur foerderung der angewandten forschung E.V, IHP GmBH, Avanti

hylas 2 Cyprus limited, RunEL, Telefonica I+D, Fogus, Infolysis. "Final report on facility design and experimentation planning", H2020, D2.4, 2020.

[4] A. Díaz Zayas, Universidad de Málaga, Telefónica Investigación y Desarrollo, Fogus Innovations & Services P.C. "Experiment and Lifecycle Manager (Release A)" H2020, D3.15, 2019

[5] [Online] (2012) http://spinroot.com/spin/Man

[6] Holzmann, G, J. *The Spin Model Checker: Primer and Reference Manual.* 2004

[7] Vega, G. (2020, 5 de marzo) El 91,5% de los internautas ya accede a Internet a través del móvil [Online] Disponible: https://elpais.com/tecnologia

[8] Mate, R. (2008, 25 de febrero) Las VideoLlamadas: Muy poco asequibles sin ninguna duda. [Online] Disponible: https://economiza.com/

[9] Behrmann G., David A., Larsen K.G. (2004) A Tutorial on Uppaal. In: Bernardo M., Corradini F. (eds) Formal Methods for the Design of Real-Time Systems. SFM-RT 2004. Lecture Notes in Computer Science, vol 3185. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30080-9_7

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA