TEST AUTOMATIZADOS DEL SISTEMA “GÜERTITO”

Basado en Simulink® Requirements™ Y Simulink® Test™

*Versión 1.0*

***GRUPO F:***

JORGE DEIRO FERRE (M19056)

JOSE FIGUEROLA PALACIOS (M19080)

JUAN RUBIO ROMERA (13399)

SERGIO MARTÍNEZ HAMDOUN (M19155)

Ingeniería del Software

Universidad Politécnica de Madrid

6 de enero de 2021

Contenido

[Introducción 1](#_Toc60934442)

[Propósito 1](#_Toc60934443)

[Alcance de los test 1](#_Toc60934444)

[Realización de los requisitos en Simulink® Requirements™ 1](#_Toc60934445)

[Desarrollo 1](#_Toc60934446)

[Selección de los requisitos a evaluar mediante test automatizados 3](#_Toc60934447)

[Requisitos para testear 3](#_Toc60934448)

[Explicación de los test realizados (basados en los requisitos) 4](#_Toc60934449)

[Introducción 4](#_Toc60934450)

[Requisito cruzado con el sistema de puerta automática para gatos 4](#_Toc60934451)

[RFU10 - Identificación única y personal en la aplicación 14](#_Toc60934452)

[Conclusiones 24](#_Toc60934453)

[Apéndices 24](#_Toc60934454)

Historial de Revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Fecha** | **Razón de los Cambios** | **Versión** |
| Versión inicial | 06/01/2021 | Redacción del documento y estructura general | 1.0 |

# Introducción

## Propósito

El objetivo de este documento es doble. Por un lado, se pretende mostrar una parte de la implementación de nuestro sistema ***Güertito***. Esta implementación se realizará utilizando Matlab y Simulink, tanto en forma de diagramas de estados como funciones definidas en Matlab.

El segundo objetivo deriva del primero, puesto que con estas pequeñas implementaciones podremos realizar, y como veremos a continuación, con éxito, diversos test automatizados para mostrar el correcto funcionamiento de las partes que se han implementado.

Cabe destacar que no se ha procedido a la implementación de un modelo completo del sistema. La razón es que de esta manera, se consiguen implementar modelos más precisos y que se asemejen más a la realidad, para que una vez que se hayan desarrollado los test de forma exitosa, proceder a la implementación final en los microcontroladores.

Es importante resaltar que los test surgen para comprobar ciertos requisitos, y sobre estos requisitos se ha realizado la implementación de los modelos y los test automatizados. Por lo tanto este documento está estructurado en dos partes: primero se muestran los requisitos programados con Simulink® Requirements™ de forma general, y posteriormente se escogen algunos requisitos para probar, con sus modelos correspondientes.

## Alcance de los test

Como ya se ha dicho, el objetivo de los test es el derivado de la propia definición de test. Además, es destacable que se han ido cambiando los test para que el resultado sea satisfactorio, por lo tanto se mostrarán los resultados que han salido positivos. El nivel de detalle de los modelos deriva del cumplimiento de los requisitos que se han seleccionado para testear, siendo especialmente importante lo dicho por los requisitos para el desarrollo de los test y su resultado definitivo.

# Realización de los requisitos en Simulink® Requirements™

## Desarrollo

Para poder crear los requisitos se ha seguido un procedimiento similar al visto en la sesión de teoría de la asignatura Ingeniería del software. Se ha creado un proyecto de Matlab y en él, utilizando la herramienta Simulink® Requirements™ se ha creado una tabla donde se han ido introduciendo todos los requisitos que ya se definieron en el documento de entra anterior llamado “*Especificación de Requisitos del Sistema Güertito*”. Se puede ver un ejemplo de la estructura en la *Ilustración 1*, y un ejemplo de requisito desarrollado en la *Ilustración 2:*

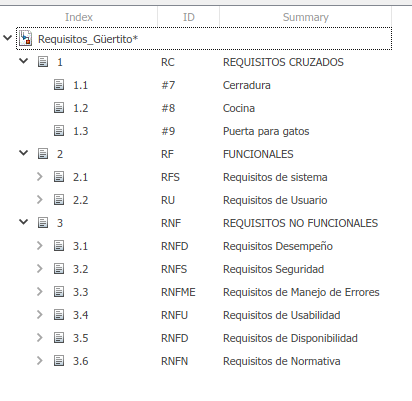


Ilustración 1: Estructura de los requisitos del sistema Güertito

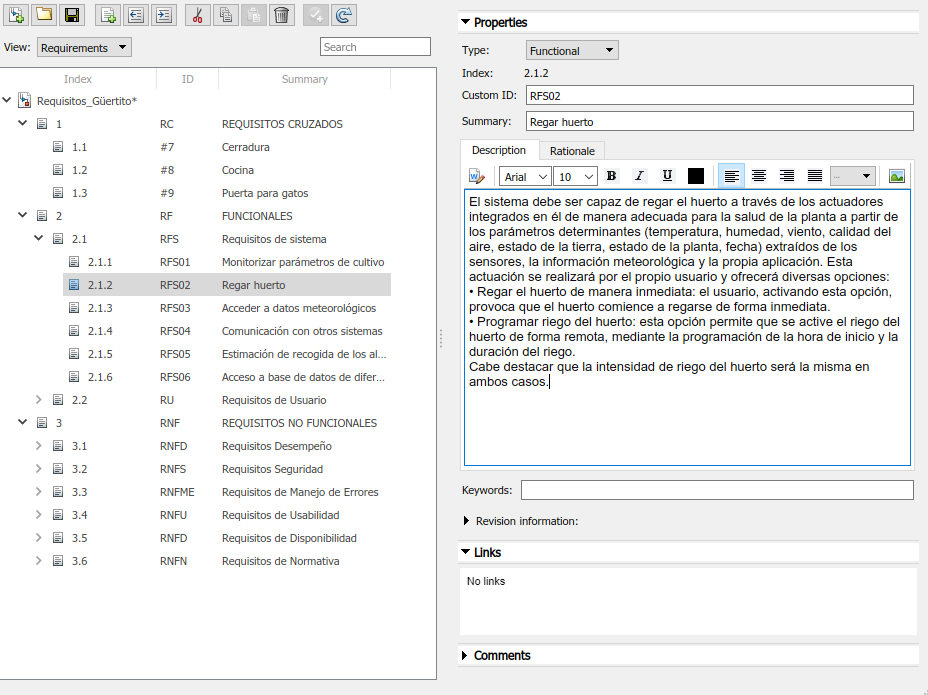


Ilustración 2: Ejemplo de requisito implementado del sistema Güertito

Una vez que se han implementado los requisitos, se ha procedido a seleccionar aquellos a evaluar, con la implementación de los modelos correspondientes.

# Selección de los requisitos a evaluar mediante test automatizados

## Requisitos para testear

Tabla 1: Tabla con los requisitos probados AÑADIR

|  |  |  |
| --- | --- | --- |
| CÓDIGO | TÍTULO | REQUISITO |
| Requisito cruzado | Puerta para gatos | La aplicación de la gatera se conectará por una API a los datos recopilados por el sistema del huerto acerca de la meteorología, y avisará en caso de precipitaciones para poder programar acciones como cerrar la gatera para evitar que el gato salga de la vivienda. |
| RFU10 | Identificación única y personal en la aplicación | El sistema debe proporcionar un usuario único y personal a cada persona que utilice la aplicación, debiéndose verificar la verdadera identidad de la persona que está usando un determinado usuario. El login en la aplicación se realizará mediante un sistema de verificación en dos pasos, estableciendo una clave única para cada usuario y una verificación vía SMS o correo electrónico. Como precondición se establece que el usuario esté dado de alta en el sistema y haya realizado dicha verificación. Una vez dado de alta se le asignar se podrá asignar un rol al usuario tal y como se indica en el requisito RFU11. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

\*Los códigos de los requisitos vienen explicados al final del documento, en el apéndice A: Glosario

Para cada requisito se ha desarrollado un modelo de implementación. Además, se ha establecido una relación entre los requisitos probados y los test implementados.

Tabla 2: Relación entre los requisitos y los números de test aplicados a cada uno de ellos AÑADIR

|  |  |
| --- | --- |
| REQUISITO | NÚMERO DE TEST |
| Requisito cruzado | 3 |
| RFU10 | 3 |
|  |  |

# Explicación de los test realizados (basados en los requisitos)

## Introducción

Se presentan, por lo tanto, los modelos de implementación de cada requisito, con la explicación de los test que se han llevado a cabo sobre cada modelo. El hecho de incluir varios test dentro de cada modelo se basa en mostrar el comportamiento distinto del sistema ante un cambio de entradas. Todo esto se mostrará por el cambio de las señales de salida.

## Requisito cruzado con el sistema de puerta automática para gatos

El modelo de implementación que conecte nuestro sistema con el sistema de apertura y cierre automático para gatos se basa en que, cuando se detecten precipitaciones cercanas, se genere una señal de alarma que se envíe a dicho sistema. Por lo tanto lo que se ha implementado es una secuencia mediante la cual se accedan a los datos meteorológicos. Si se ha accedido de forma correcta a la información meteorológica, se genera un pulso de salida en la señal “*pronóstico*”. El siguiente paso consiste en acceder a la información de precipitaciones, sólo si se ha accedido de forma correcta al pronóstico temporal. Si se obtiene la información de precipitaciones, se pasa a evaluar si son cercanas o no, parámetro que se determinar por la señal “*lluvia*”, que sigue la siguiente lógica:

*if (lluvia==1)*

*Alarma*

*if (lluvia == 2)*

*No\_alarma*

La señal de alarma o no alarma viene definida por “*mensaje\_gatera”,* que envía esta información codificada al sistema de la puerta automática para gatos, para que posteriormente este sistema actúe en consecuencia de esta información enviada.

Se ha implementado un sistema de fallos, de tal forma que si no hay conexión a internet, se incrementen el número de intentos que de forma automática el sistema acceda a internet para recopilar los datos meteorológicos, y así se mande un mensaje de error a la aplicación del usuario del sistema Güertito, para que lo vuelva a intentar. Esta señal viene implementada como “error”.

Por lo tanto, el sistema modelado derivado de este requisito tiene las siguientes señales de entrada y de salida:

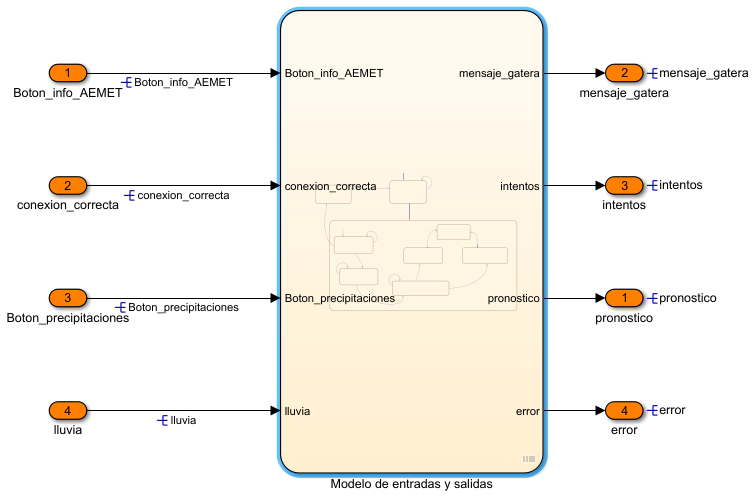


Ilustración 3: Modelo de entradas y salidas

Se ha intentado modelar de alguna forma los botones que el usuario tiene que pulsar para acceder a la información de las precipitaciones, que son “*Boton\_info\_AEMET”* y “*Boton\_precipitaciones”.* Además, se ha creado la señal de “*intentos*” simplemente con el objetivo de monitorizar los intentos que el sistema realiza para acceder de forma correcta a la información del tiempo. Por lo tanto, el modelo que resulta, teniendo en cuenta todo lo anterior, es el siguiente:

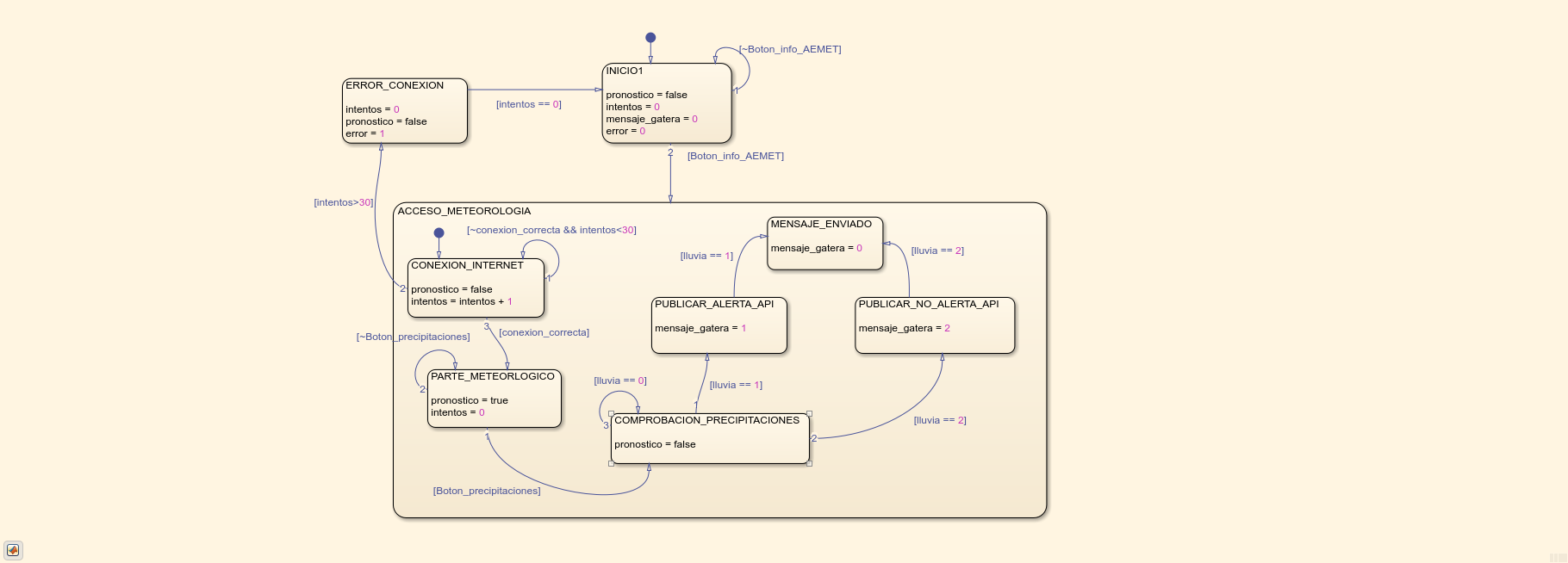


Ilustración 4: Modelo de conexión con el sistema de la puerta de gatos que genera una alerta de lluvias

Sobre este modelo se han realizado los test que se detallan a continuación:

1. **Test1**: Comprobación de se accede correctamente al pronóstico del tiempo, se accede a las precipitaciones, pero estas son los suficientemente lejanas como para no mandar una señal de alarma al sistema de la puerta de la gatera. El flujograma del test realizado es el siguiente, mostrándose los cambios que se desean en las diversas señales de entrada, así como las transiciones con sus descripciones:

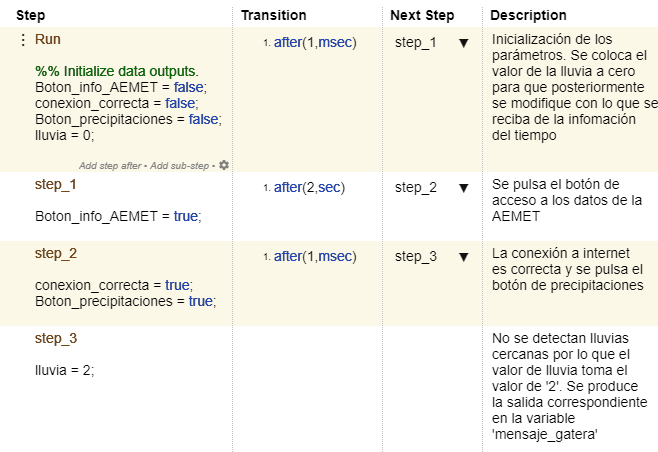


Ilustración 5: Flujograma del test1

En cuanto a la salida, vemos como efectivamente se recibe el pronóstico del tiempo correctamente **(señal azul)** y la señal de lluvia toma el valor de 2 **(señal naranja)**, indicando que no hay lluvias cercanas:

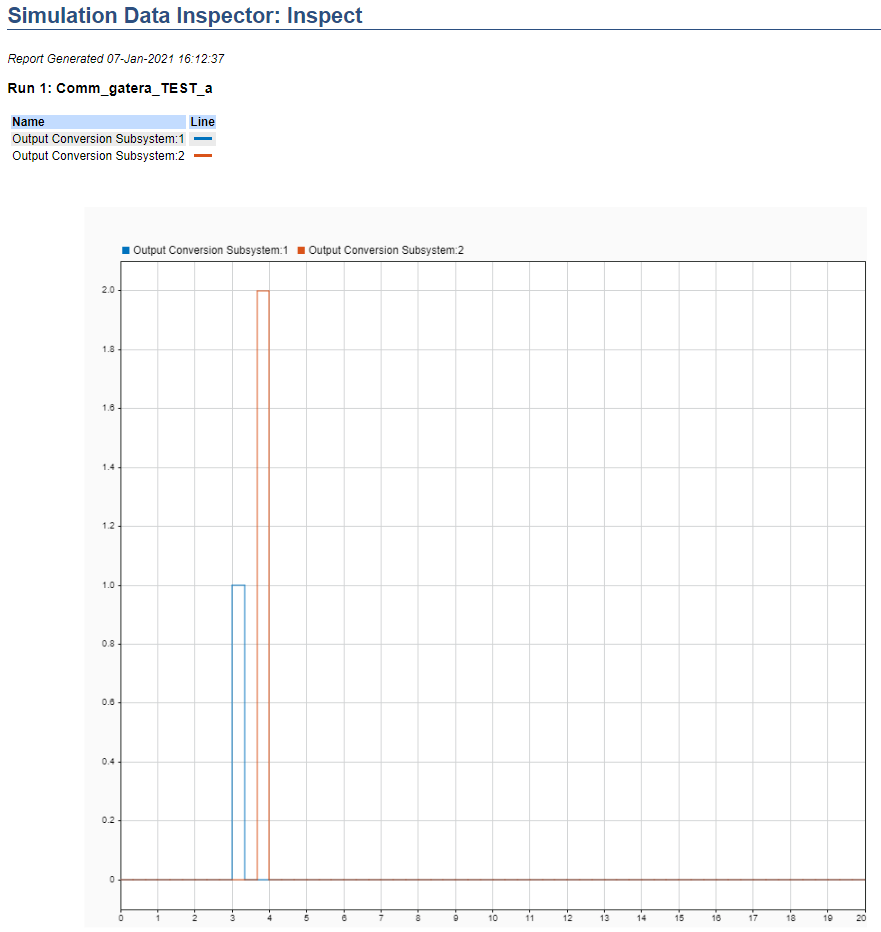


Ilustración 6: Resultado en las señales de salida del Test1

1. **Test2**: se sigue el mismo procedimiento que el caso anterior, pero esta vez sí que hay precipitaciones cercanas **(señal naranja)**, por lo tanto la señal de lluvia adoptará un valor de 1, y mandando la señal de alarma o aviso al sistema de la puerta para gatos. También se comprueba que se recibe de forma satisfactoria el pronóstico **(señal azul)**. El código de implementación es el siguiente:



Ilustración 7: Implementación del código para el Test2

Y por lo tanto, en la salida tendremos un ‘1’ como respuesta al valor de lluvias próximas:

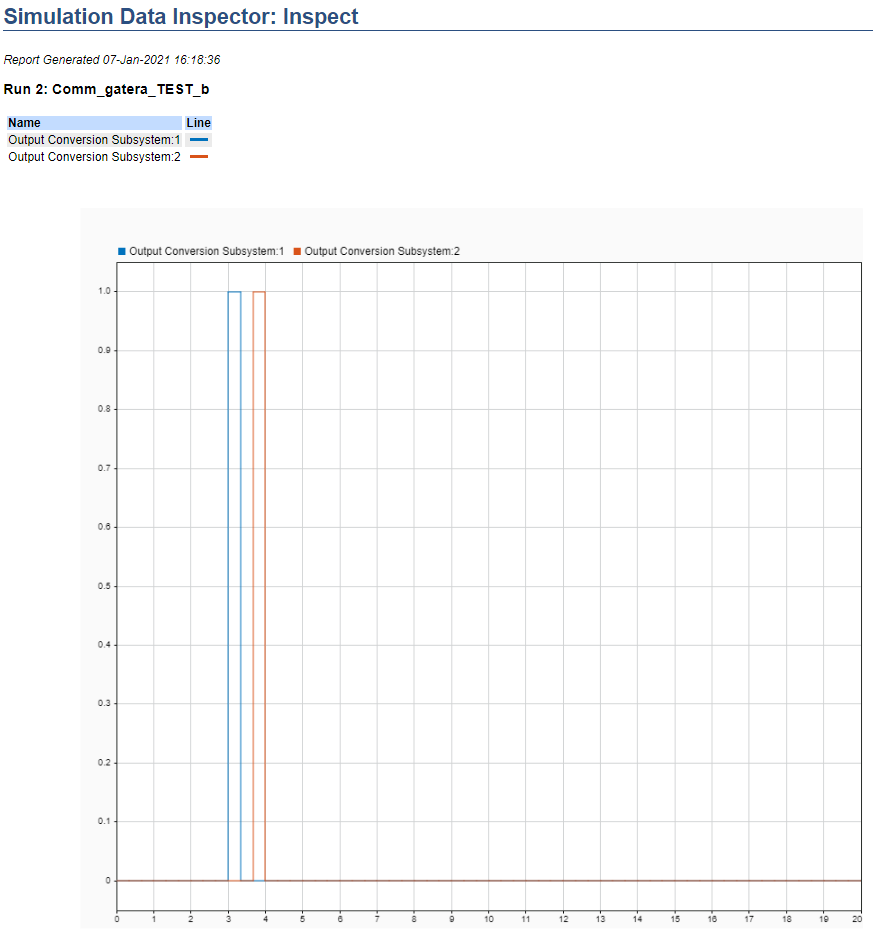


Ilustración 8: Resultado en las señales de salida del Test2

1. **Test3**: Este test es ligeramente diferente a los anteriores, porque ahora lo que se está testeando es que no hay conexión a internet, por lo que la señal de pronóstico será 0 y cuando se alcance un número de intentos determinados, pasará al estado de error **(señal azul)**, se reinician los intentos y se vuelve al inicio de la aplicación, en la que el usuario tenga que volver a darle al botón de obtención de datos meteorológicos.



Ilustración 9: Implementación del Test3

Y el resultado del test es el siguiente:

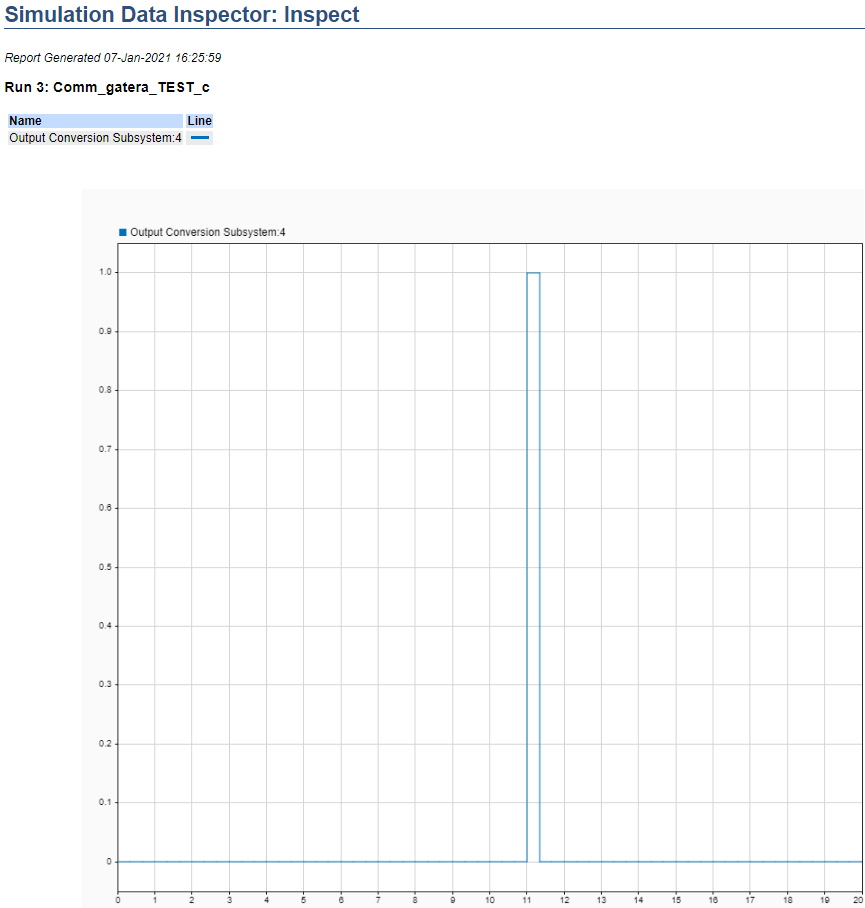


Ilustración 10: Resultados de la señal de salida del Test3

A modo de curiosidad, se muestra la señal de la variable intentos **(señal azul)**, que se ha puesto como salida para vislumbrar que, efectivamente, cuando se sobrepasan los intentos **(señal naranja)** de acceso al tiempo, se activa la señal de error:

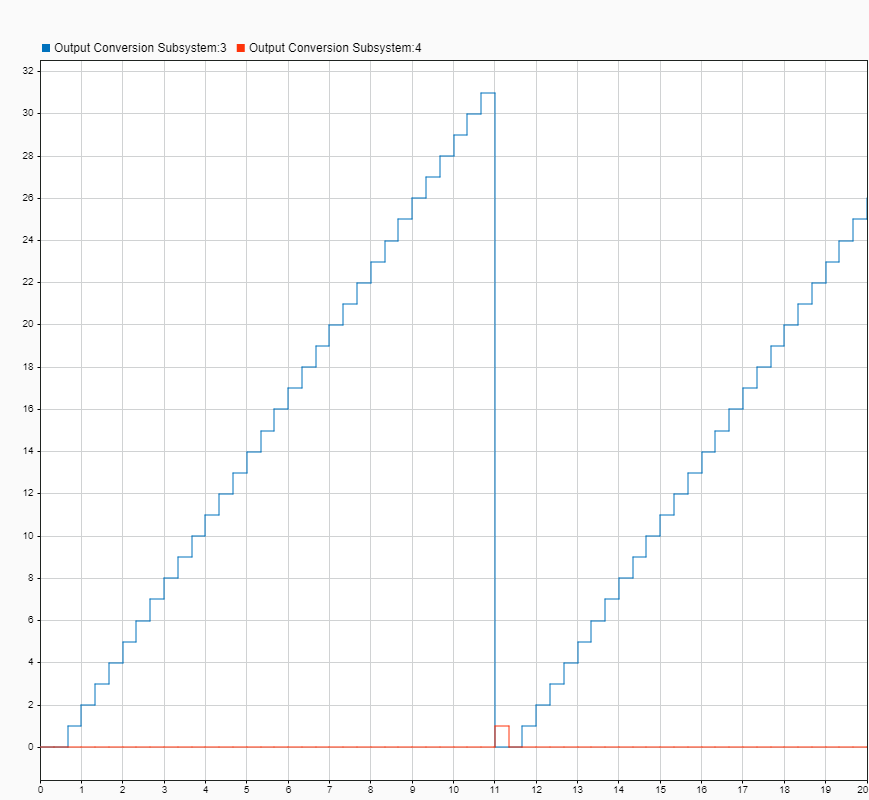


Ilustración 11: resultados de la salida cuando se realiza el Test3

Con esto ya quedaría verificado el requisito cruzado de comunicación con la gatera. Podría haber una mejora a este requisito y a su correspondiente verificación, que sería aumentar el número de casos del tiempo. Por ejemplo, se podría comprobar que, en vez de lluvia, sea un pronóstico de nieve, de vientos fuertes, de temperaturas extremadamente bajas… con el objetivo de que se cierre la puerta de la gatera y el gato no sufra condiciones adversas.

## RFU10 - Identificación única y personal en la aplicación

El modelo de implementación que permita al usuario registrarse y posteriormente realizar el login es algo más complejo que el anterior, pues hay una gran diversidad de casos. En esta primera implementación se han tenido casi todos en cuenta, si bien es cierto que se podría completar con algunos más en versiones posteriores.

Este sistema simplemente detecta lo que desea el futuro usuario de la aplicación. Mediante la variable ‘*Selector*’, se recoge ese deseo, de bien registrarse o logearse.

En caso de que el usuario decida **registrarse** (Selector tiene valor de 1) entonces el usuario pasará por una serie de etapas, representadas en forma de un diagrama de estados, y no se podrá pasar a la siguiente etapa hasta que no complete todas las fases. Se pasará por un estado de introducir correo, otro de introducir contraseña y otro de introducir otros datos complementarios y secundarios.

Después de esto, se pasará a la fase de **login**, de forma obligada. En esta primera fase de registro, se deberá verificar mediante una verificación en dos pasos el nuevo usuario. En el login se siguen los mismos pasos pero sin el estado de introducir esa información secundaria considerada en el registro: estado de introducir correo, introducir contraseña y verificación en dos pasos de ese usuario. Hay que destacar que la verificación en dos pasos se realizará mediante un envío de un correo electrónico al introducido por el usuario momentos antes. Cuando el usuario pasa a la fase de login, la variable Selector cambia de valor, de tal forma que ya no podrá volver a la fase de registro. En este momento la variable ‘*Selector’* valdrá 2.

Además, en afán de hacer de forma más eficiente la implementación, se han utilizado las mismas variables intermedias de los estados de introducir correo, introducir contraseña y verificación en dos pasos, las cuales se reinician a *false* en caso de haber superado el login correctamente.

Por último, en cuanto el usuario ha pasado las fases de registro y login de forma satisfactoria, se vuelve al estado inicial del menú de la app.

Se han considerado una serie de posibles fallos, bien sea debido a la conexión con el servidor, o bien porque el usuario no ha seguido las indicaciones de seguridad a la hora de crear la contraseña. Esto se ha representado, al igual que el requisito anterior, mediante dos variables de ‘*error’* e ‘*intentos*’.

Las entradas utilizadas y las salidas monitorizadas son las siguientes:

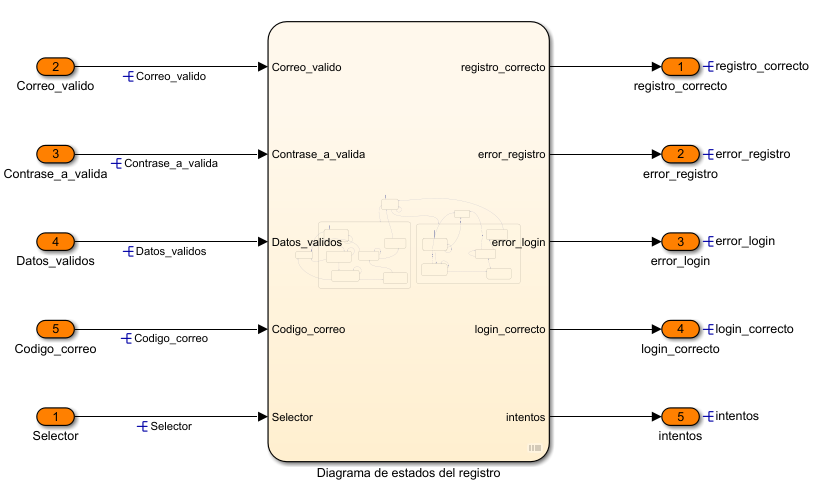


Ilustración 12: entradas y salidas del registro y login

Se ha identificado dos tipos de error bien sea durante la fase de registro o durante la fase de login, por los motivos comentados más arriba. El diagrama de estados queda de la siguiente manera:

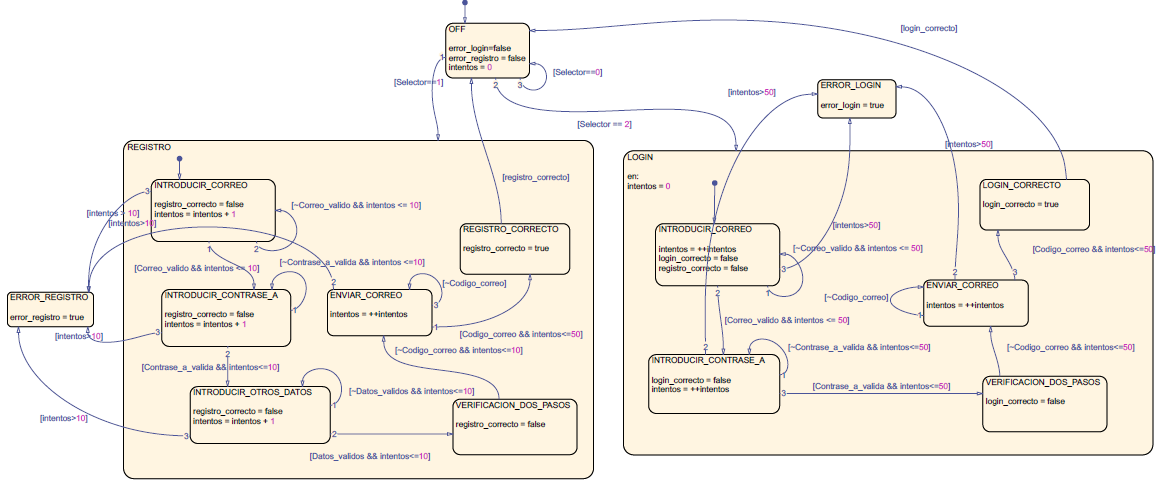


Ilustración 13: Diagrama de estados del registro y login

Sobre este modelo se ha procedido a implementar una serie de test. En concreto, se han implementado 3 test que tienen que ver con el caso de que un usuario se registre y realice el login correctamente, un usuario que solamente realice el login, y un caso de error en el registro.

* **Test1**: con la variable ‘*Selector’* a 0, se procede a registrarse un usuario. Se cambia el valor de la variable ‘*Selector’*, poniéndose a 1, y poco a poco comienza esa secuencia en la cual el usuario se registra:
  + Introducir correo
  + Introducir contraseña
  + Introducir otros datos
  + Verificación en dos pasos
  + Enviar correo y confirmar
  + Registro correcto

Cuando el usuario se ha registrado correctamente, es importante que las señales anteriores que se han ido poniendo a *true*, cambien de estado a *false*, porque a continuación se pasará al login. El login es el mismo proceso descrito antes, solo que aquí no hay una introducción de datos secundarios. Cuando se ha producido el login correcto, entonces la señal Selector cambia a 0, porque ya no es necesario que comience ninguna otra secuencia.

El código de este test es el siguiente:



Ilustración 14: Código del test1

Y como resultado se sacarán las señales de *registro\_correcto* **(señal azul)**, *y login\_correcto* **(señal naranja)***:*

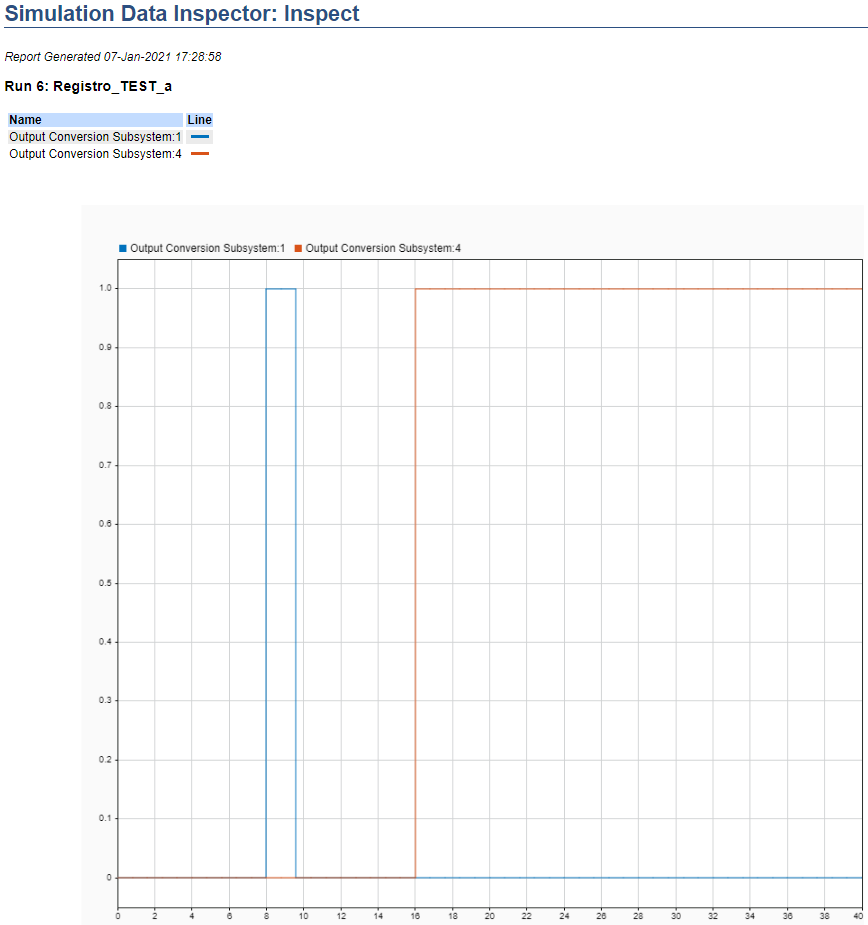
**

Ilustración 15: Resultados del test1

Como podemos observar, nos interesa que la señal *login\_correcto* **(señal naranja)**se mantenga igual a 1, lo que quiere decir que el usuario permanecerá con la sesión iniciada el tiempo que esté activo en la aplicación.

* **Test2:** En este test, la variable Selector es 2, por lo que directamente el usuario pasa al login, ya que en su momento ya se registró, por lo tanto aquí solamente veremos que la variable *login\_correcto* **(señal naranja)** toma el valor de 1, no el registro\_correcto **(señal azul)**:



Ilustración 16: Código el test2

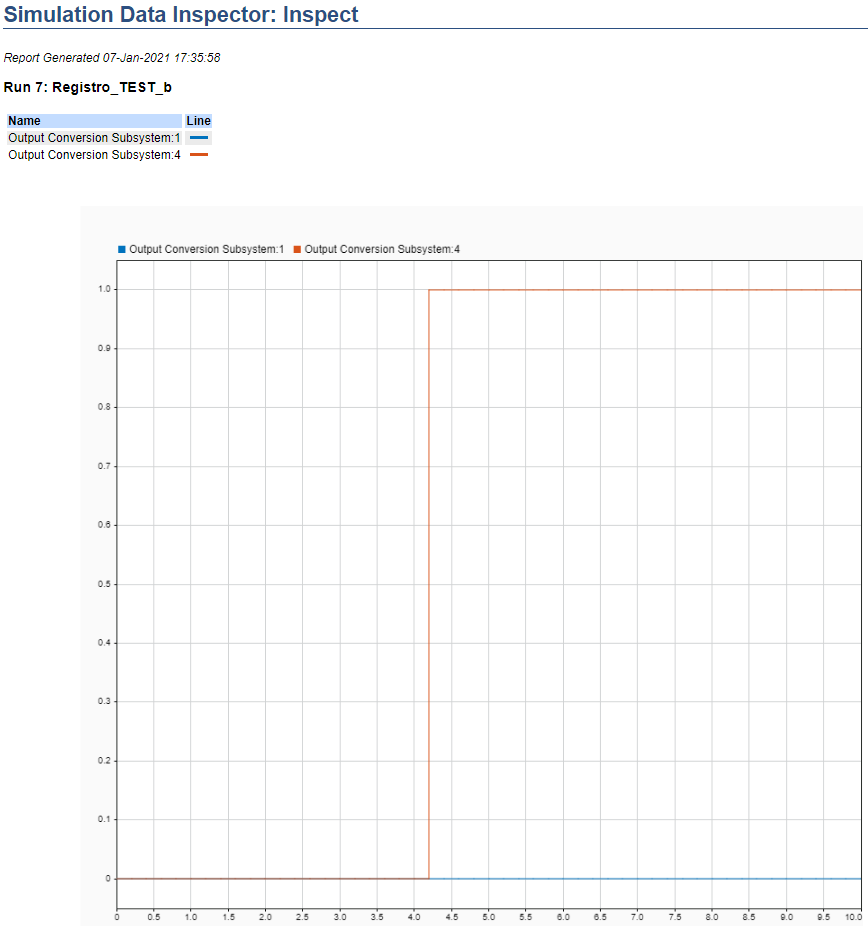


Ilustración 17: Login

* **Test3:** en este caso no se realiza ni el registro ni el login porque se produce un error, haciendo que la variable intentos llegue a su máximo, sacando un 1 por la señale de salida, en este caso de la señal de *error\_registro* **(señal azul)**:

****

Ilustración 18: Test3 de error en el registro

Efectivamente, como vemos, es la contraseña la que no cumple con los requisitos de seguridad, manteniéndose a false todo el rato, hasta que la señal de error se activa, como vemos a continuación:

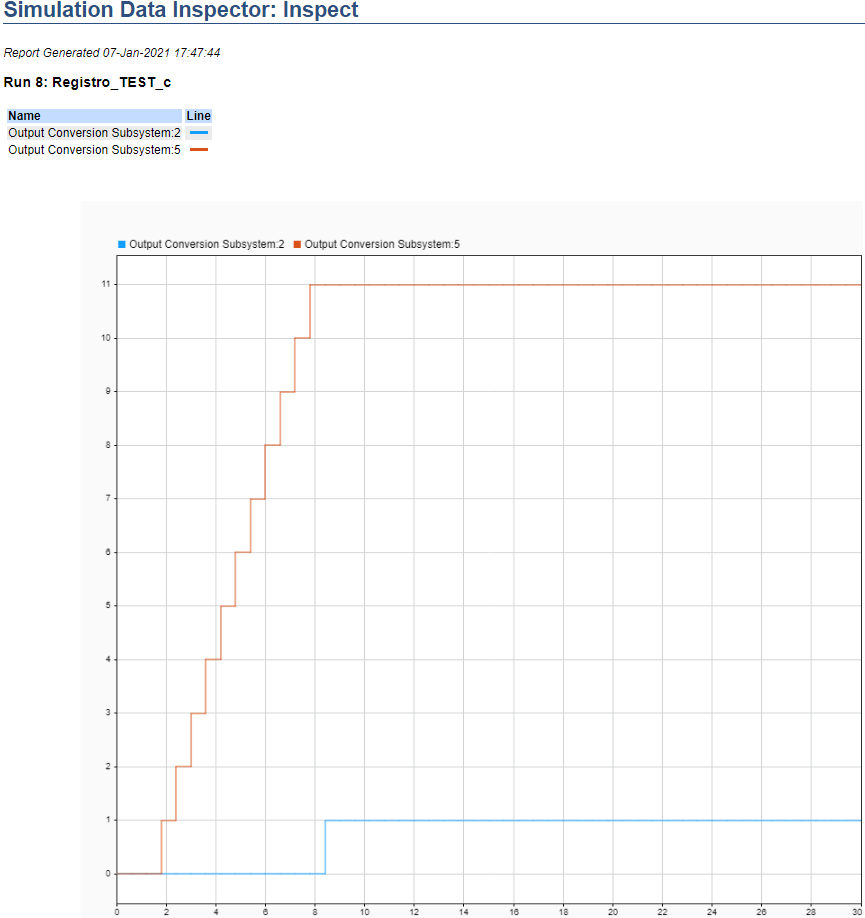


Ilustración 19: Señal de error con los intentos realizados

Como vemos, también se ha incluido la señal de los intentos que se realizan antes de que salte el error.

La implementación de este requisito también ha dejado margen de mejora. Por ejemplo, ha faltado la implementación de otro botón de cancelar cuando el usuario lo requiera, que no sería otra cosa que una señal más de entrada.

# Conclusiones

# Apéndices

**Apéndice A: Glosario**

|  |  |
| --- | --- |
| **RFS** | **Requisito Funcional de Sistema** |
| **RFU** | **Requisito Funcional de Usuario** |
| **RNF** | **Requisito No Funcional** |
| **RNFD** | **Requisito No Funcional Desempeño** |
| **RNFS** | **Requisito No Funcional Seguridad** |
| **RNFME** | **Requisito No Funcional Manejo de Errores** |
| **RNFU** | **Requisito No Funcional de Usabilidad** |
| **RNFDI** | **Requisito No Funcional de Disponibilidad** |
| **RF** | **Requisito Funcional** |
| **RNFN** | **Requisito No Funcional de Normativa** |