ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERIA DEL SOFTWARE

DOCUMENTOS INTERACTIVOS EXCEL PARA SIMULADORES

INTERACTIVE EXCEL DOCUMENTS FOR SIMULATORS

Realizado por
Andrés Fornells - O'Connor
Tutorizado por
Daniel Garrido Márquez
Departamento
Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Julio de 2014

Fecha defensa:
El Secretario del Tribunal

Resumen: La tecnología conocida como Microsoft Visual Studio Tools for Office (VSTO) ofrece la posibilidad de integrar Office con la plataforma .NET de forma que podamos implementar aplicaciones .NET con la apariencia de documentos de Microsoft Office.

La utilización de la tecnología VSTO con hojas de cálculo para la creación de herramientas de simulación es un campo interesante por la familiaridad que las hojas de cálculo ofrecen a cualquier usuario. Cuando se desarrolla un complemento para un simulador, dicho complemento suele ser muy específico de un simulador concreto, por lo que los elementos desarrollados en estos proyectos no es posible reutilizarlos con otro simulador. Esta es la motivación que lleva a crear este proyecto: facilitar la creación de extensiones para Excel adaptables a distintos tipos de simulador, de manera que sea posible reutilizar las extensiones y sus elementos constituyentes.

Para eso, se ha desarrollado un framework para la creación de extensiones Excel con VSTO que puedan ser fácilmente adaptables a diferentes tipos de simuladores. Los principales puntos que toca este framework son:
- Elementos comunes a todo proyecto de simuladores. Se han ofrecido precargados un conjunto de elementos que son comunes en el desarrollo de estas aplicaciones, de manera que no sea necesario tener que implementarlos.
- Definición de elementos de simulación, lo que se ha denominado en el framework controles. Se ha buscado reducir el coste de desarrollo y maximizar la reutilización.
- Comunicación con los simuladores. Se ha definido e implementado una interfaz que permite la comunicación de las hojas Excel con los posibles motores de simulación. Se ha ofrecido esta interfaz en la interfaz ISimulatorService y se ha ofrecido también un cliente para comunicar con los simuladores de esta interfaz.

Palabras claves:complemento, csharp, documentos interactivos, Excel, simuladores

Abstract: The technology known as Microsoft Visual Studio Tools for Office (VSTO) offers the possibility of integrating Office with the .NET framework, allowing us to develop .NET applications with the look and feel of Microsoft Office documents.

The development of simulation tools using the VSTO technology for Excel is an interesting field because of the familiarity users have with Excel documents. When an Excel add-in is developed for a simulator, the add-in is generally very specific to a particular simulator, and the elements developed in these projects can't be reused with other simulators. This is the motivation for creating this project: to facilitate the creation of Excel add-ins that are adaptable to different types of simulators, making it possible to reuse the add-ins and its elements.

For that, I have developed a framework for creating Excel add-ins with VSTO that can be easily adapted to different types of simulators.

The main points covered by this framework are:

- Common elements to all projects for simulators. A series of elements common to all Excel applications for simulators have been included in the framework, so that it is not necessary for the developer to have to implement these himself.

- Development of simulator elements. These elements have been called controls in this framework. The main goal here has been to reduce the development cost and to maximize reuse.

- Communication with simulators. An interface has been defined and implemented that enables communication of Excel books with potential simulation engines. The interface used is specified in the file ISimulatorService. A client that can be used to communicate with simulators that follow this interface has been included in the project.

Keywords: add-in, csharp, Excel, interactive documents, simulators

Índice

1. Introducción

1.1. Motivación del TFG

La tecnología conocida como Microsoft Visual Studio Tools for Office (VSTO) ofrece la posibilidad de integrar Office con la plataforma .NET de forma que podamos implementar aplicaciones .NET con la apariencia de documentos de Microsoft Office.

La utilización de la tecnología VSTO con hojas de cálculo para la creación de herramientas de simulación, es un campo interesante por la familiaridad que las hojas de cálculo ofrecen a cualquier usuario. Cuando se desarrolla un complemento para un simulador, éste suelen ser muy específico de un simulador concreto, por lo que los elementos desarrollados en estos proyectos no es posible reutilizarlos con otro simulador. Esta es la motivación que lleva a crear este proyecto: facilitar la creación de extensiones para Excel adaptables a distintos tipos de simulador, donde los elementos desarrollados puedan traerse de un proyecto a otro.

1.2. Objetivos del TFG

El principal objetivo del TFG es facilitar la creación de extensiones para Excel adaptables a distintos tipos de simulador y que los elementos constituyentes del complemento sean reutilizables de un proyecto a otro. Este objetivo puede descomponerse en 3 subobjetivos:

- Desarrollar los elementos comunes a todo proyecto para simuladores, de modo que no sea necesario para el desarrollador Excel tener que implementarlos él cada vez que cree un nuevo proyecto para simuladores. Por ejemplo, todo proyecto para simuladores tiene un modo ejecución y un modo edición y tiene un panel para cambiar de un modo a otro. Este panel se podría incluir en el framework.
- Ofrecer un mecanismo estándar para el desarrollo de elementos de simulación (controles), además de un conjunto de herramientas para simplificar el desarrollo de estos. El objetivo principal aquí es hacer que estos controles sean reutilizables, tanto de un proyecto a otro, como de un simulador a otro (este último punto es compartido con el último objetivo).
- Ofrecer una interfaz genérica de comunicación con los simuladores. Esta interfaz sería implementada por los simuladores, de modo que sería posible utilizar los mismos controles con distintos simuladores, o incluso utilizar un proyecto completo con un simulador distinto.

1.3. Estructura de la memoria

La memoria se estructurará de acuerdo a las fases del desarrollo. Es decir, se definen los stakeholders, requisitos, casos de uso y diseño e implementación. A continuación, se encuentran las pruebas, el resultado del proyecto (templates e interfaz genérica) y la conclusión.

Finalmente, se encuentran las referencias bibliográficas y los anexos técnicos (manual de uso, ejemplo de aplicación y pruebas en detalle).

1.4. Términos importantes

Para especificar la definición de los diferentes apartados, se llamará:

- **Usuario Excel** a todo aquel que utilice un documento Excel generado con el framework creado en este proyecto.
- **Desarrollador Excel** a todo aquel que utilice el framework creado en este proyecto para crear un complemento Excel.

2. Estado del arte y tecnologías utilizadas

2.1. Estado del arte

Como se ha explicado antes, actualmente el desarrollo de complementos Excel para simuladores produce aplicaciones imposibles de reutilizar con simuladores distintos al usado para dicho complemento, y que además contienen un conjunto de elementos que no son reutilizables, es decir, que ni se pueden traer de un proyecto a otro, ni mucho menos se pueden utilizar con otros simuladores.

Esta es la situación actual del desarrollo de complementos para simuladores, y conseguir resolver este problema es el objetivo de este proyecto.

2.2. Tecnologías principales

Para el desarrollo de este framework se utiliza .NET 4.0. Se utiliza el lenguaje c#. Este desarrollo se realiza con Visual Studio 2012 junto con el paquete Visual Studio Tools for Office (VSTO) y con Microsoft Excel 2013 (sin embargo, las plantillas finales generadas también se han creado para Microsoft Excel 2010).

Para la comunicación con los simuladores, se utiliza el framework Windows Communication Foundation (WCF).

Para el control de versiones, se utiliza Team Foundation Version Control (TFVC), que es un sistema de control de versiones centralizado de microsoft. Además, se utilizan los servidores gratuitos de Visual Studio Online para alojar el código. Visual Studio se integra perfectamente con esta página web y con TFVC, lo que hace que sea muy fácil configurar el proyecto para que las diferentes versiones se alojen en el servidor.

Se han utilizado etiquedas <<summary>> para especificar los métodos, estas etiquetas permiten que el intellisense de visual studio pueda mostrar las definiciones de los métodos mientras se escribe código. Para generar la documentación (en html y latex) de las clases a partir de estas etiquetas, se ha utilizado Doxygen.

Finalmente, para realizar los diagramas UML se ha utilizado Enterprise Architect.

2.2.1. VSTO

VSTO es un conjunto de herramientas que permiten desarrollar complementos para Office. En este proyecto en particular, se desarrolla un complemento de nivel de documento para Excel.

Los complementos de nivel de documento son uno de los 2 tipos de complementos que se pueden desarrollar. Los complementos de nivel de documento son complementos que solo trabajan sobre un único libro. Siempre que se desarrolla una aplicación Excel para simuladores, lo que se busca es al final crear un documento que tenga en la barra Ribbon las herramientas para ese simulador específico y se puedan crear los diferentes elementos en ese documento en particular.

Los otros tipos de complementos, lo que permiten es crear complementos que instala el usuario y que pueden correr en cualquier documento Excel. Estos no son los que nos interesan, ya que son más complicados de desarrollar, tienen muchos más problemas, y además no son los tipos de complementos usados para este tipo de proyectos, por lo que no tendría ningún sentido hacer las plantillas de este tipo.

Cuando se desarrolla un complemento de nivel de documento, este permite cambiar la barra Ribbon, cambiar el actions pane, y realizar cambios en el documento como, por ejemplo, modificar una celda.

Los elementos principales que se manejan en este tipo de aplicaciones son:

- ThisWorkbook. Representa el libro Excel. Es utilizado para suscribirse a eventos como el de creación de nueva hoja o click derecho. También es en esta clase en la que se implementa la actividad principal del proyecto, ya que aquí es donde se ejecuta el método de inicio (apertura del documento) y el de finalización (cierre del documento). Por último, la instancia única de ThisWorkbook se utiliza para acceder a las hojas del libro, entre otros elementos.
- Clases que heredan de Worksheet. Estas clases representan una hoja Excel. Si se quiere hacer alguna ejecución específica a una hoja o se quiere modificar una hoja, se debe escribir código en la hoja o acceder a esta.
- Objetos Range, que representan un conjunto de celdas de una hoja Excel. También graficos, y otros controles, como por ejemplo botones.
- La clase Globals, que permite acceder a ThisWorkbook y a las hojas creadas desde el editor de visual studio.
- El objeto único Application, que permite hacer pequeñas cosas como cambiar el título o realizar operaciones de intersección de objetos Range.

2.2.2. WCF

WCF es un conjunto de tecnologías orientadas al manejo de servicios. WCF puede ser utilizado para desarrollar servicios. Lo que se hace es establecer un contrato de comunicación (un ServiceContract) e implementar el código del servicio de acuerdo al contrato.

WCF también es utilizado para comunicar con servicios. Lo que se hace es añadir una referencia a un servicio, y visual studio genera automáticamente lo que se denomina un cliente WCF. Este objeto cliente contiene ya los métodos que contiene el servicio, facilitando el envío de los diferentes mensajes.

Por último, hay un archivo muy importante que se llama app.config. En este archivo, se encuentran las configuraciones para la comunicación. Se encuentran cosas como la dirección del servicio, o el timeout (tiempo de espera) en los mensajes al servicio.

3. Stakeholders

Principalmente se tienen los Desarrolladores Excel y los Usuarios Excel. Los desarrolladores son los que utilizarán el framework para el desarrollo de complementos Excel para simuladores, y los usuarios son los que usarán los complementos generados por los desarrolladores.

Estos últimos también son afectados por este proyecto, ya que las características del framework afectarán a la calidad de los complementos desarrollados y al tiempo que se tarde en desarrollarlos.

4. Requisitos

A continuación se definen los requisitos funcionales y no funcionales del proyecto.

4.1. Requisitos funcionales

Se organizarán los requisitos según los 3 grupos de funcionalidad principales de la aplicación. Cada uno de estos grupos se asocia a uno de los 3 objetivos principales del proyecto. La razón por la que se ha decidido organizar los requisitos de esta manera es porque estos grupos de funcionalidad son bastante independientes y permiten realizar una división clara de los requisitos.

Los requisitos se encuentran ordenados, dentro de cada categoría, por orden de importancia (primero los que son más importantes o deben implementarse antes).

4.1.1. Elementos comunes

Este conjunto de requisitos está relacionado con el objetivo de no tener que reprogramar los elementos comunes a toda aplicación Excel para simuladores (cortar, pegar, eliminar…), sino poder tenerlos ya implementados de un proyecto a otro. La lista de requisitos para esta categoría es:

RF1. El Usuario Excel podrá cambiar de modo entre modo ejecución y modo edición (con introducción de contraseña en caso de que se requiera).
RF2. El Usuario Excel podrá copiar, pegar, cortar y eliminar controles (cuando esté en modo edición).
RF3. El Desarrollador Excel podrá definir si hace falta contraseña para cambiar a modo edición y, en caso de que haga falta contraseña, cual es esta contraseña.
RF4. El Desarrollador Excel podrá incorporar paneles de introducción de datos del simulador sin un elevado coste de programación.
RF5. El Usuario Excel podrá introducir los datos del simulador (dirección, etc…) en caso de que el programador haya decidido que este paso esté disponible.
RF6. El Usuario Excel podrá consultar información valiosa sobre la aplicación o el/los simuladores, como son, por ejemplo, el nombre de la aplicación y el estado del simulador.

4.1.2. Desarrollo de controles

Este conjunto de requisitos está relacionado con el objetivo de poder crear controles de manera sencilla y reutilizable. Los controles aquí hacen referencia a elementos de programación que realicen tareas como actualizar una casilla o mostrar un gráfico. La lista de requisitos para esta categoría es:

RF7. El Desarrollador Excel podrá construir controles que se mantengan entre sesiones (prácticamente no se alteran por el cierre de la aplicación) sin un elevado coste de desarrollo.
RF8. El Desarrollador Excel podrá consultar el modo actual (de ejecución o de edición) de manera programática sin un elevado coste de desarrollo.
RF9. El Desarrollador Excel podrá hacer que una serie de datos sea persistente entre sesiones de Excel (es decir, tras cerrar y abrir un documento Excel) sin un elevado coste de desarrollo.
RF10. El Desarrollador Excel podrá realizar tareas periódicas que persistan entre sesiones sin un elevado coste de desarrollo.

4.1.3. Comunicación con simuladores

Este conjunto de requisitos está relacionado con el objetivo de tener una interfaz común de comunicación con los simuladores, que facilite la integración de proyectos o controles con simuladores distintos al simulador para el que fueron diseñados originalmente. También se incluye un requisito relacionado con la consulta del estado del simulador (el RF11). La lista de requisitos para esta categoría es:

RF11. El Desarrollador Excel podrá consultar el estado de los simuladores de manera programática sin un elevado coste de desarrollo.

RF12. El Desarrollador Excel podrá realizar una serie de operaciones básicas comunes a todo simulador (arrancar el simulador, detenerlo,…) de manera programática a través de la utilización de una interfaz genérica de simuladores ofrecida con la herramienta (siempre y cuando el simulador implemente esta interfaz).

RF13. El Desarrollador Excel podrá realizar lecturas de una variable de un simulador de manera programática sin un elevado coste de desarrollo programática a través de la utilización de una interfaz genérica de simuladores ofrecida con la herramienta (siempre y cuando el simulador implemente esta interfaz)

4.2. Requisitos no funcionales

En este proyecto en particular, algunos requisitos no funcionales tienen una gran importancia, esto es debido a que muchas de las cosas que buscan los interesados son cosas fuertemente relacionadas con requisitos no funcionales, en particular, la reutilización y la interoperabilidad. Los requisitos no funcionales principales son:

RNF1. El programa se desarrollará utilizando VSTO (Visual Studio Tools for Office), y se implementará como un complemento Excel de nivel de documento.

RNF2. El programa será desarrollado en c#.

RNF3. El desarrollador de documentos Excel podrá reutilizar los controles de una aplicación Excel a otra en su totalidad y sin un elevado coste de desarrollo si estos controles han sido desarrollados de la manera recomendada en el manual de uso y usando el framework desarrollado en este proyecto.

RNF4. El desarrollador de documentos Excel podrá reutilizar partes de los controles (por ejemplo, las ventanas solamente) sin un elevado coste de desarrollo si estos controles han sido desarrollados de la manera recomendada en el manual de uso y usando el framework desarrollado en este proyecto.

RNF5. El desarrollador podrá intercambiar cualquier simulador por otro siempre y cuando este simulador cumpla con las especificaciones de interfaz de comunicación dadas en este proyecto y el proyecto no requiera del simulador características adicionales a las que exige la interfaz genérica.

RNF6. Existirá una documentación clara de cómo utilizar la herramienta.

RNF7. La aplicación estará en castellano.

5. Casos de uso

A continuación se explicarán primero cuales son los actores que participan en los casos de uso, y seguidamente se mostrará el diagrama de casos de uso del proyecto y la descripción detallada de cada uno de los casos de uso que aparece en este diagrama.

5.1. Actores

Los actores son:

- Desarrollador Excel: todo aquel que utilice el framework desarrollado en este TFG con el fin de crear complementos Excel de nivel de documento.
- Usuario Excel: todo aquel que utilice documentos Excel generados por los desarrolladores Excel.

5.2. Diagrama de casos de uso

En la siguiente imagen se representan los actores asociados a los casos de uso en los que participan.

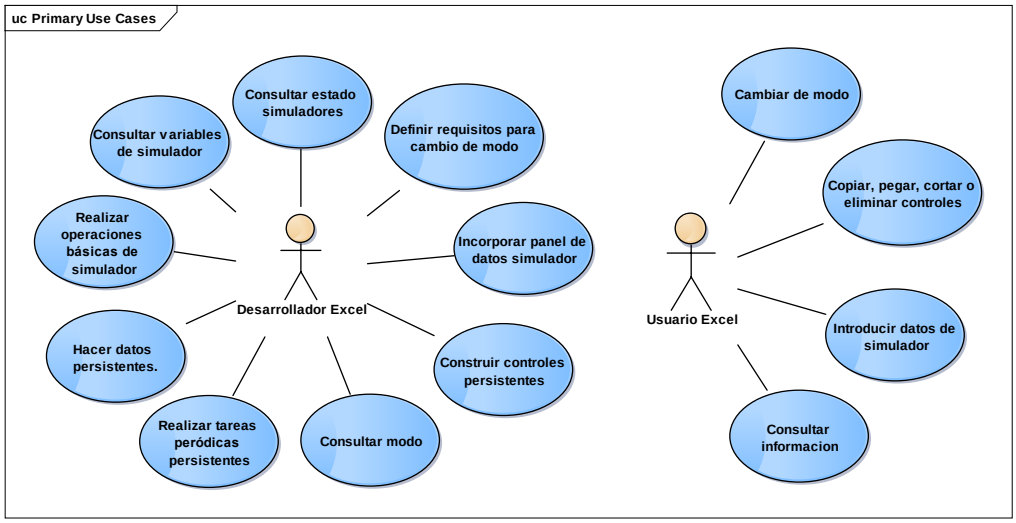

Figura 1. Imagen del diagrama de casos de uso

5.3. Descripción de los casos de uso

A continuación se detallan los casos de uso:

CU1. Cambiar de modo	
Descripción	El usuario de documentos Excel cambia de modo de uso en un documento Excel. Este cambio puede ser a modo edición o a modo ejecución.
Pre-condición	El usuario de documentos Excel debe tener abierto un documento Excel que haya sido desarrollado con la herramienta.
Post-condición	En dicho documento se produce un cambio de modo de uso. Un cambio de modo significa que algunas opciones se ocultarán (o aparecerán), y algunos elementos Excel cambiarán su comportamiento.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF1
Escenario principal	El Usuario Excel ha abierto la barra de opciones de la aplicación (barra Ribbon) en un documento Excel creado con la herramienta. 1. El Usuario Excel está en modo edición y quiere cambiar de modo en dicho documento. 2. El Usuario Excel pulsa en el botón cambiar de modo. 3. El sistema cambia de modo a modo ejecución, oculta los botones exclusivos al modo edición y establece el flag de modo a modo ejecución (lo cual cambiará el comportamiento de algunos elementos Excel basados en este flag).
Escenario alternativo	**Escenario alternativo al principal** 1b. El Usuario Excel está en modo ejecución y quiere cambiar de modo en dicho documento. 2b. El usuario Excel pulsa el botón cambiar de modo 3b. El sistema muestra un panel solicitando la contraseña si el desarrollador indicó que se necesitaba una (sino, pasa directamente al paso 5b). 4b. El Usuario Excel introduce la contraseña correcta. 5b. El sistema cambia de modo a modo edición, con los correspondientes cambios en los botones y con el cambio del flag de modo a modo edición. **Escenario alternativo al 4b.** 4c. El Usuario introduce la contraseña incorrecta. 5c. El sistema cierra el panel de introducción de la contraseña y muestra un panel informando de que la contraseña es incorrecta. 6c. El Usuario Excel cierra el mensaje de contraseña incorrecta.

CU2. Copiar, pegar, cortar o eliminar controles	
Descripción	El Usuario Excel realiza la operación de copiar, pegar, cortar o eliminar un determinado control. Estos comandos no solo copian lo visual, copian el comportamiento dinámico y en principio solo se realizan sobre controles que actúan sobre celdas (no sobre controles formados por un objeto WPF, por ejemplo).
Pre-condición	El usuario de documentos Excel debe tener abierto un documento Excel que haya sido desarrollado con la herramienta y estar usándolo en modo edición.
Post-condición	Se produce la acción especial que corresponda según la operación (copiado, pegado, cortado o eliminación de controles).
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF2
Escenario principal	**Escenario principal (copiar)** El Usuario Excel se encuentra en modo edición en un documento Excel creado con la herramienta. 1. El Usuario Excel selecciona unas celdas y pulsa click derecho sobre éstas. 2. El sistema muestra un panel contextual. 3. El Usuario Excel pulsa copiar celdas dentro del panel contextual (dentro de la zona de acciones especiales). 4. El sistema guarda en la zona de copiado especial del programa los datos del control eliminando los datos anteriores de esta zona (si hay) y marca que los datos pertenecen a una operación de copiar.
Escenario alternativo	**Escenario alternativo al principal (cortar)** 3b. El Usuario Excel pulsa cortar celdas dentro del panel contextual (dentro de la zona de acciones especiales). 4b. El sistema guarda en la zona de copiado especial del programa los datos del control eliminando los datos anteriores guardados (si hay) y marca que los datos pertenecen a una operación de cortar. 5b. El sistema elimina el contenido de las celdas seleccionadas. **Escenario alternativo al principal (eliminar)** 3c. El Usuario Excel pulsa eliminar celdas dentro del panel contextual (dentro de la zona de acciones especiales). 4c. El sistema elimina el control del documento Excel. **Escenario alternativo al principal (pegar)** 1d. El Usuario Excel selecciona una zona para pegar los controles y hace click derecho. 2d. El sistema muestra un panel contextual. 3d. El Usuario Excel pulsa pegar celdas dentro del panel contextual (dentro de la zona de acciones especiales). 4d. Si hay algún dato en la zona de copia, se pega la información en la zona seleccionada (sino no se hace nada).

CU3. Definir requisitos para cambio de modo	
Descripción	El Desarrollador Excel define si hace falta contraseña para cambiar de modo y cuál es esta contraseña.
Pre-condición	El Desarrollador Excel debe estar trabajando en Visual Studio sobre un proyecto que se haya creado usando una de las plantillas proporcionadas para usar la herramienta, o haber copiado el código de la herramienta a mano a su proyecto de Visual Studio.
Post-condición	Se producen cambios en la configuración de la contraseña para el cambio de modo de acuerdo a lo introducido por el desarrollador.
Prioridad	Media
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF3
Escenario principal	El Desarrollador Excel tiene abierto un proyecto Visual Studio con todos los elementos necesarios para usar la herramienta cargados. 1. El Desarrollador Excel entra en la región de código donde se solicita el cambio de modo. 2. El Desarrollador Excel cambia los parámetros para el cambio de modo de manera que reciba como parámetros la contraseña que se quiere para este cambio de modo. 3. El Desarrollador Excel guarda el proyecto Visual Studio.
Escenario alternativo	**Escenario alternativo al principal** 2b. El Desarrollador Excel cambia los parámetros para el cambio de modo de manera que reciba no reciba como parámetros ninguna contraseña para cambio de modo. 3b. El Desarrollador Excel guarda el proyecto Visual Studio.

CU4. Incorporar panel de datos simulador	
Descripción	El Desarrollador Excel incorpora un panel de datos de simulador a su proyecto Visual Studio para simuladores.
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Se incorpora un panel de datos de simulador a la aplicación.
Prioridad	Media
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF4
Escenario principal	El Desarrollador Excel tiene abierto un proyecto Visual Studio con todos los elementos necesarios para usar la herramienta cargados. 1. El Desarrollador Excel se posiciona en la zona donde quiere que aparezca el panel (generalmente el arranque de la aplicación Excel). 2. El Desarrollador Excel instancia y lanza la ventana de datos de simulación (pasando datos del simulador por defecto y los modos). 3. El Desarrollador Excel obtiene y usa los datos recogidos por la ventana.

CU5. Introducir datos de simulador	
Descripción	El Usuario Excel es preguntado por los datos de simulación, e introduce los datos de localización del simulador (IP, etc…).
Pre-condición	El Usuario Excel debe tener abierto un documento Excel que haya sido desarrollado con la herramienta y haber hecho algo que haya hecho que se muestre la ventana (generalmente abrir el documento).
Post-condición	La ventana se cierra y el sistema recibe los datos introducidos por el usuario (en caso de no haber cancelado).
Prioridad	Media
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF5
Escenario principal	El Usuario Excel es presentado el panel de introducción de datos del simulador (con una serie de opciones rellenadas con valores por defecto) 1. El Usuario Excel introduce la dirección IP y el puerto. 2. El Usuario Excel pulsa aceptar. 3. El sistema recibe los datos del panel y recibe que se pulsó aceptar la ventana.
Escenario alternativo	**Escenario alternativo al principal** 2b. El Usuario Excel pulsa cancel (o pulsa la x de la ventana). 3b. El sistema recibe los datos que fueron rellenados, pero recibe que se pulsó cancelar, por lo que ignora los datos.

CU6. Consultar información	
Descripción	El Usuario Excel consulta información sobre el documento Excel y elementos relacionados con éste (como los simuladores).
Pre-condición	El Usuario Excel debe tener abierto un documento Excel que haya sido desarrollado con la herramienta.
Post-condición	No se produce ningún cambio en el sistema.
Prioridad	Baja
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF6
Escenario principal	El Usuario Excel tiene abierto un documento Excel desarrollado con la herramienta. 1. El panel está oculto, por lo que el Usuario Excel pulsa el botón de mostrar panel de información. 2. El sistema muestra el panel con datos como el nombre de la aplicación Excel y el estado del simulador (en el action panel de Excel). 3. El Usuario Excel pulsa el botón para ocultar el panel de información.

CU7. Construir controles persistentes	
Descripción	El Desarrollador Excel construye un control que se mantiene entre sesiones Excel.
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Existe un nuevo control persistente en el proyecto Visual Studio.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF7
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel añade un botón a la barra Ribbon para su nuevo control (es un ejemplo). 2. El Desarrollador Excel desarrolla las ventanas WindowsForms que ve necesarias. 3. El Desarrollador Excel implementa la lógica para modo edición y para modo ejecución, basándose en las clases y métodos de la herramienta. 4. El Desarrollador Excel implementa las interfaces correspondiente a controles persistentes (que se encontrarán en la herramienta) implementando los métodos que permiten la reconstrucción del objeto y de ahí la persistencia.

CU8. Consultar modo	
Descripción	El Desarrollador Excel consulta programáticamente el modo en el que se encuentra el usuario.
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Se obtiene el modo (ejecución o edición).
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF8
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel se posiciona en la zona donde quiere introducir el código dependiente del modo. 2. El Desarrollador Excel llama al método para consultar el modo de uso. 3. El Desarrollador Excel utiliza el valor devuelto, por ejemplo, para ejecutar un código en modo edición y otro en modo ejecución (un if-else).

CU9. Hacer datos persistentes	
Descripción	El Desarrollador Excel indica que una serie de datos son persistentes y que su valor debe conservarse entre sesiones (tras el cierre y apertura).
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Se incorporan datos persistentes al proyecto VS.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF9
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel se posiciona en la zona del código donde tiene los datos o donde quiere indicar que son persistentes. 2. El Desarrollador Excel indica los datos que desea que sean persistentes entre sesiones usando los métodos correspondientes de la herramienta.

CU10. Realizar tareas periódicas persistentes	
Descripción	El Desarrollador Excel implementa una tarea periódica que persiste entre sesiones (tras cerrar el documento Excel).
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Se incorpora la tarea periódica persistente al proyecto VS.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF10
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel se posiciona en la zona donde quiere introducir el código de la tarea periódica. 2. El Desarrollador Excel informa de la tarea periódica usando los métodos correspondientes de la herramienta e indica qué datos deben mantenerse entre sesiones. 3. El Desarrollador Excel implementa la tarea periódica y registra un método (por ejemplo) al evento de eliminación de hoja. 4. En el método de eliminación de hoja, cuando se elimina la hoja de interés, se llama al método correspondiente de la herramienta para indicar que la tarea periódica debe ser eliminada.
Escenario alternativo	**Escenario alternativo al principal** 4b. El Desarrollador Excel introduce código que cambia algunos campos por esta eliminación de una hoja, pero no detiene la tarea periódica (por ejemplo, un sistema de monitorización deja de monitorizar las variables que se encontraban en esa hoja, pero continua monitorizando las demás)

CU11. Consultar estado simuladores	
Descripción	El Desarrollador Excel consulta el estado de un simulador (si es posible conectar con éste o no). Este estado no será necesariamente el del momento exacto en el que se haga la petición (el estado es consultado por el sistema de manera periódica).
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta.
Post-condición	Se obtiene el estado del simulador al ejecutarse el código.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF11
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel registra el simulador para monitorización (por ejemplo, en el arranque del proyecto). 2. El Desarrollador Excel se posiciona en la zona donde quiere consultar el estado del simulador. 3. El Desarrollador Excel llama al método para consultar el estado del simulador para el que quiere información. 4. El Desarrollador Excel utiliza el valor devuelvo, por ejemplo, para consultar una variable del simulador solo si es posible conectar con éste.
Escenario alternativo	**Escenario alternativo al principal** 3b. El Desarrollador Excel indica que quiere el valor actual (utilizando un método distinto), y no con un retraso de unos ms (esto implica que cuando se ejecute el código, el programa se quedará atascado unos ms hasta que se obtenga un valor reciente). 4b. Lo mismo que en el paso 4.

CU12. Realizar operaciones básicas de simulador	
Descripción	El Desarrollador Excel realiza una serie de operaciones básicas comunes a todo simulador (arrancar el simulador, detenerlo,…) de manera programática utilizando la interfaz genérica de simuladores.
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta y el simulador debe implementar la interfaz genérica.
Post-condición	Cuando el código de la región en la que se incorpora ejecute, se ejecutará la operación básica implementada en código (arrancará el simulador, se detendrá, etc…).
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF12
Escenario principal	**Escenario principal (arranque)** El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel se posiciona en la zona del código donde quiere arrancar el simulador. 2. El Desarrollador Excel consulta el estado del simulador (CU11). 3. El Desarrollador Excel solicita que arranque el simulador (a través de la interfaz genérica).
Escenario alternativo	**Escenario alternativo al principal** 3b. El Desarrollador Excel solicita que se detenga el simulador.

CU13. Consultar variables de simulador	
Descripción	El Desarrollador Excel consulta de manera programática el valor de una variable del simulador utilizando para ello un cliente de servicios y la interfaz genérica de simuladores.
Pre-condición	El Desarrollador Excel debe estar trabajando en VS sobre un proyecto que tenga cargados los elementos necesarios para usar la herramienta y el simulador debe implementar la interfaz genérica.
Post-condición	Se obtiene el valor de una de las variables del simulador al ejecutarse el código.
Prioridad	Alta
Autor(es)	Andrés Fornells O'Connor
Versión	1.0
Requisitos asociados	RF13
Escenario principal	El Desarrollador Excel tiene abierto el proyecto. 1. El Desarrollador Excel se posiciona en la zona del código donde quiere introducir el código que consulta una variable del simulador. 2. El Desarrollador Excel consulta el estado del simulador (CU11). 3. El Desarrollador Excel consulta el valor de la variable del simulador (a través de la interfaz genérica) solo si es posible conectar con el simulador (un simple if). 4. El Desarrollador Excel implementa algún código que trabaja sobre el valor devuelto.

6. Diseño e implementación

En esta sección se describirán los elementos principales del diseño de la aplicación. Se incluyen imágenes de los diferentes diagramas usados para diseñar el sistema.

6.1. Estereotipos propios

Para indicar que un campo es persistente (al serializar instancias de la clase, el valor del campo es incluido), se utilizará el estereotipo <<persistente>>. Si un campo debe ser ignorado y no persistir, entonces simplemente no se marca con este estereotipo.

Para indicar que una clase implementa el patrón Singleton, se utilizará el estereotipo <<Singleton>>, para no tener que incluir en el diseño una y otra vez un campo estático Instance. La implementación de Singleton usada en esta API y que se recomienda que usen los desarrolladores Excel viene en secciones posteriores.

6.2. Propiedades

En c# hay una cosa que se llama propiedades. Son como campos de clases normales, pero la diferencia es que tienen un get y un set. Cuando alguien consulta la propiedad, se ejecuta el get y cuando alguien modifica la propiedad, se ejecuta el set.

Existen muchas maneras diferentes de representar una propiedad en UML. En los modelos de este proyecto se ha optado por representarlas como un campo privado (que comienza con minúscula o barra baja), y un método get y/o set asociado a dicho campo según sea de lectura, de escritura o de lectura/escritura. Por ejemplo para una propiedad llamada Password de lectura pública y escritura privada, habría un campo password privado, un método GetPassword público y un método SetPassword privado.

Esto se ha hecho así para favorecer que el modelo sea más portable y porque otros modos de representación no permiten indicar si las propiedades son de lectura o de lectura escritura ni si esta lectura/escritura es pública o privada. Existen propiedades que no almacenan datos, sino que solo devuelven una transformación de otro campo, para estas propiedades generalmente se ha representado con el get y/o el set, pero sin un campo privado asociado.

6.3. Almacenamiento de datos persistentes

Para implementar casi todas las características de Excel, es necesario poder almacenar datos que se mantengan tras cerrar el documento. En una aplicación de escritorio, esto se haría almacenando los datos en una serie de archivos, y después cargando estos archivos al abrir la aplicación. Sin embargo, en la programación en Excel esto no es tan fácil, ya que todo ha de almacenarse dentro del Excel mismo de alguna manera.

Las APIs de Excel ofrecen de 3 maneras principales de guardar:

- Usando el atributo Cached. Poniendo el atributo Cached sobre un campo o propiedad de ThisWorkbook.cs (la clase libro) o alguna clase hoja (Hoja1.cs, etc…) se consigue que el campo o propiedad sea persistente.
- Utilizando Custom Xml Parts. Consisten en serializar grandes trozos de información a Xml e insertarlos dentro del documento Excel al guardar en una zona reservada para esto.

- Utilizando hojas ocultas para guardar los datos. Esta solución tiene bastantes problemas a la hora de implementar.

Si se usan hojas ocultas, lo que se hace es guardar los datos persistentes como strings dentro de las casillas de la hoja. Para guardar los datos como strings, se deben serializar, y esta serialización debe hacerse a mano con alguno de los sistemas de serialización. Existen un límite de caracteres para una celda, y esto obliga a tener que dividir los datos.

Cached tiene un límite de memoria de hasta 10MB por campo/propiedad y hasta 100MB en total, y es muy simple de utilizar (simplemente se pone una etiqueta y ya está).

Custom Xml Parts no tiene los mismos problemas de memoria, pero el guardado y cargado debe hacerse a mano, y tiene algunos problemas más con el polimorfismo que Cached. Además, debido a que es bastante más complicado, implica crear una serie de clases de ayuda.

Por tanto, las hojas ocultas son muy complicadas y problemáticas, Custom Xml Parts no tiene problemas de límites de memoria, pero requiere mucho más trabajo para conseguir lo mismo que Cached. Cached es el más fácil de utilizar, y es muy extraño que sus límites de memoria se superen (tendría que ser una macro de dibujo o algo por el estilo). Por eso, para la implementación de esta API y como estrategia recomendada para el manual de uso se va a utilizar Cached.

6.3.1. Almacenamiento de instancias

Para almacenar un objeto de manera persistente usando Cached, el objeto debe almacenarse en algún campo en una hoja o libro Excel marcando el campo con el atributo Cached.

Para todo objeto marcado con el atributo Cached, lo único que persistirá para dicho objeto son los campos y propiedades públicos que no estén marcados con el atributo XmlIgnore.

Para conseguir que un campo o propiedad público no sea persistente, se debe marcar con la etiqueta XmlIgnore. En caso contrario, su valor se mantendrá tras el cierre y apertura del documento. Para los campos o propiedades privados, como estos simplemente se ignoran, la única forma de hacer que sean persistentes es ponerlos como públicos. Para solucionar el problema de que otras clases puedan acceder a estos campos y propiedades, lo que se hará en este proyecto (y se recomendará para el manual de uso) es ponerles un nombre que empiece por "_". Cuando un programador vea que empieza por "_", sabrá que no debe acceder a éste.

6.3.2. Clases estáticas

Con cached (y con Custom Xml Parts), solo se pueden marcar objetos como persistentes, no se puede marcar que los campos de una clase estática lo sean. Para solucionar este problema, y permitir que las clases estáticas puedan tener campos persistentes, se utilizará el patrón singleton. En toda la API se usará el patrón Singleton en exactamente la manera que se describe a continuación, y se recomienda a los desarrolladores Excel que lo implementen de esta manera, ya que así será más fácil saber cómo se llama la instancia, como manejar la clase, etc…

Se llamará Instance al objeto único del Singleton en la implementación de Singleton recomendada en esta API. Así, el marcado de los datos de una clase llamada MiClase como persistentes en una hoja o libro sería como sigue:

```
[Cached]
public MiClase DatosMiClase {
      get{return MiClase.Instance;}
      set{MiClase.Instance = value;}
}
```

En cuanto a la definición de la clase en sí, este será su aspecto:

```
public class MiClase{
        private static MiClase _instance = new MiClase();
          public static MiClase Instance {get{return _instance;} set{_instance = value;}}

        //Constructor vacío requerido. Es privado para evitar otras instancias.
          private MiClase(){}

        //Propiedades persistentes
        public Tipo1 _Var1{ get; set; }
        public Tipo2 Var2{ get; set; }

        //Propiedades no persistentes
        private Tipo3 Var3{ get; set; }
         [XmlIgnore]
        public Tipo4 Var4{ get; set; }
}
```

Las cosas más importantes que se pueden ver en este ejemplo de código son:

- La instancia única de la clase se llama Instance, este nombre es bastante estándar, y es aplicable a cualquier clase sin crear confusión.
- Hay un constructor vacío, y este es privado.
- Los campos persistentes públicos se dejan sin atributo y los "privados" se ponen con _ en el nombre y se ponen como públicos.
- Los campos no persistentes públicos se dejan con el atributo XmlIgnore, y los privados simplemente se dejan privados.

Todos los métodos y propiedades serán accedidos a través de este campo Instance.

6.3.3. Clases con campos estáticos y no estáticos

Dado que se usa el patrón singleton para las clases estáticas, no es posible poder almacenar tanto los campos estáticos como los campos de las instancias que sean persistentes. Para solucionar este problema en los extraños casos de clases que tienen campos estáticos persistentes además de campos no estáticos persistentes es dividir la clase en 2 clases. Una que contiene la parte estática, y una que contiene la parte no estática.

6.3.4. Uso de otros sistemas

Si un desarrollador necesita usar Custom Xml Parts porque la característica que está usando tiene un enorme uso de memoria, entonces, al igual que usaba el sistema nativo de Cached, puede usar sin demasiados problemas el sistema de Custom Xml Parts sin perder por ello características, ya que Cached en sí se usa de la manera nativa. Además, el esquema del Singleton sería muy similar, solo que usando los atributos de la tecnología que se usa para Custom Xml Parts.

6.3.5. Problemas

Hay algún que otro problema con algunos tipos de c#, que no permiten serializacion. Cuando se incluyan en el diseño clases de las que aparecen aquí y estas deban almacenarse de manera persistente, realmente se estará usando la versión serializable de estas, pero esto no se indicará en el diseño para no complicar demasiado el diseño. Entre ellos, los que me he encontrado que hacen que una clase no pueda serializarse con Cached son:

- Tuple. Es muy útil para guardar conjuntos de diferentes tipos. Para poder seguir usando Tuple, se definirá una clase TupleSerializable.
- List, y Hashset. Se definen las clases ListSerializable y HashSetSerializable para poder guardar las listas y hashsets.
- Dictionary. Se define la clase DictionarySerializable para poder guardar diccionarios.

En el caso de los HashSet y los Diccionarios, debido a que es imposible serializar una interfaz utilizando la tecnología de Cached (y la Custom Xml Parts), se ofrecen 2 versiones, una que ignora completamente el Comparer, y otra que recibe explícitamente el tipo del Comparer utilizado (haciendo así posible la serialización).

Estas clases no tienen grandes decisiones de diseño, simplemente utilizan algún tipo que si sea serializable internamente y ofrecen un mecanismo para hacer un casting del tipo serializable al tipo original. Por esta razón, no se incluye un diseño de estas clases.

Además de estos tipos, tampoco son serializables la mayoría de tipos de VSTO (Range, Worksheet, etc…). La serialización de estos tipos se discute en la sección de controles persistentes.

6.4. Cambio de modo

El usuario debe poder cambiar de modo (entre los modos edición y ejecución). Además, el desarrollador debe poder especificar los datos de contraseña para cambio de modo a modo edición.

La idea es que el desarrollador pueda indicar si se requiere contraseña para el cambio a modo edición, y cuál es tanto desde el editor de visual studio como desde el código. Si estos datos se cambian durante la ejecución del programa, deberían persistir, para dar más opciones al desarrollador.

Además, si el usuario cambia de modo y cierra el documento, lo normal es que el documento siga en el mismo modo cuando lo vuelva a abrir.

Las variables utilizadas para esto se encuentran en la clase SimuGlobals. Como tiene campos persistentes, y se quiere que la clase sea accesible de manera global, se implementa el patrón

singleton. Para la parte de las contraseñas, se utilizan las propiedades RequierePasswordEdicion y PasswordEdicion, y para guardar en qué modo se está, se utiliza la propiedad ModoActual. Para el modo se utiliza un enumerado (ModoUso) que puede tomar los valores EJECUCION y EDICION.

Las 2 primeras propiedades se quiere que las pueda establecer el desarrollador Excel desde visual studio. Para esto, los 2 valores vendrán incluidos en un documento .resx llamado ParametrosGenerales.resx, y el sistema los leerá de dicho documento en caso de no haber sido modificados por el código.

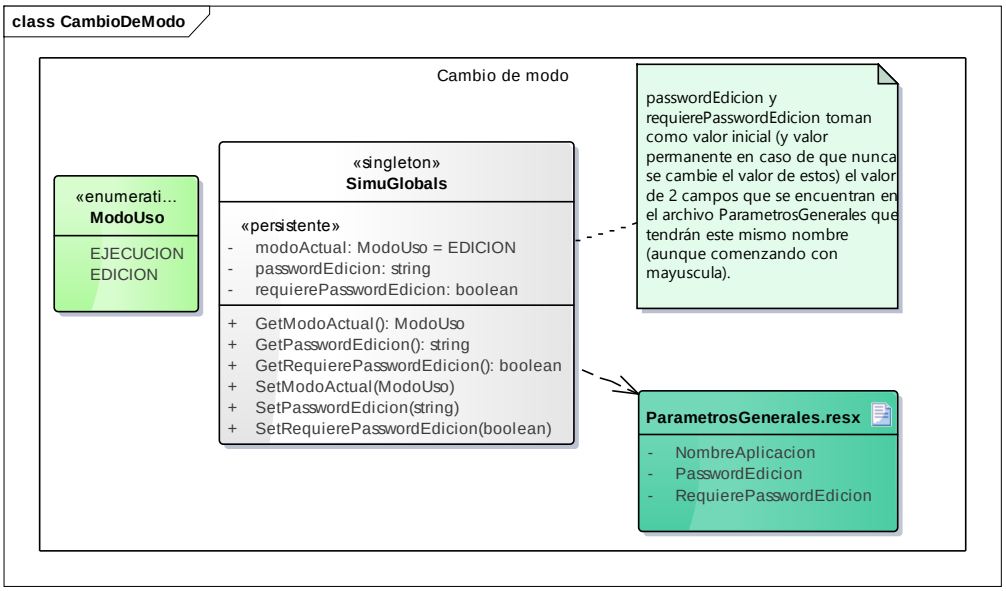

Figura 2. Diagrama de clases de esta parte de la arquitectura

6.4.1. Panel Ribbon de cambio de modo

Para que el usuario pueda cambiar de modo, se ofrece en la barra Ribbon un botón para cambio de modo. En el cambio de modo a modo edición, se ofrece una ventana Windows Forms que permite introducir la contraseña para cambio de modo.

6.5. Copiar, pegar, cortar o eliminar controles basados en celdas

El Usuario Excel debe poder copiar, pegar, cortar y eliminar controles Excel basados en celdas Excel (sobre las celdas seleccionadas). El Usuario Excel debe estar en modo edición para poder hacer esto.

Las 4 operaciones básicas se pueden realmente simplificar en 3: copiar, pegar y eliminar. En esta API, la operación de cortar se implementará como un copiado seguido inmediatamente de la eliminación de las celdas copiadas.

Cuando se hace una copia, como en todos los sistemas de copia, los datos anteriormente copiados se borran. El copiado en esta API está pensado para que sea posible eliminar el control y seguir pudiendo pegar los datos en alguna celda, es decir, que los datos copiados no deben depender de que el control siga existiendo (sino sería imposible realizar el cortado).

La finalidad de copiar es que después puedan reconstruirse los datos al pegar. Por tanto, se deben almacenar tanto la posición de las celdas (se usará como referencia la equina superior izquierda del mínimo rectángulo que rodea el conjunto) como el contenido de los controles asociados a dichas celdas.

Como el contenido depende de cada control, es éste el único que sabe cómo copiarlo y el que debe realizar la copia. Del mismo modo, este es el único que sabe eliminar el control. Además, es imposible para la API saber cuándo un control está asociado a una celda por sí solo, por eso, debe ser el control (o el gestor de este control) el que avise de que tiene un control asociado a una celda. Para la gestión de las operaciones se usará la interfaz IControladorOpCeldas.

Al crearse el control, para ser avisado, éste deberá suscribir un objeto IControladoOpCeldas a las celdas sobre las que trabaja. Esto se hace a través de la clase SistemaOpCeldas, que ofrece 4 métodos estáticos para esto, AddControladorOpCeldas, SetCeldasAsociadas, GetCeldasAsociadas y RemoveControladorOpCeldas, que permiten asociar un controlador a un conjunto de celdas (de manera que si se hace una copia, cortado o eliminación el controlador será avisado) y permiten obtener las celdas asociadas a un controlador y desuscribir un controlador por completo.

La interfaz IControladorOpCelda ofrece los siguientes métodos:

- IDatosComportamientoCelda CopiaComportamiento (Range celdaCopiada, Range seleccionTotal). Este método es en el que se debe realizar la copia. Los datos de la copia se encuentran en el objeto IDatosComportamientoCelda.
- void DestruyeComportamiento (Range celda, out bool desasociaCelda). En este método se realiza la eliminación del control. desasociaCelda representa si se quiere eliminar la celda actual del conjunto de celdas a las que está suscrito el controlador (lo normal es que sí).

Finalmente, al igual que con el copiado y eliminación, el pegado debe implementarlo el desarrollador Excel. Cuando se hace el copiado, el controlador devuelve un objeto IDatosComportamientoCelda, que es usado para realizar el pegado a una determinada celda. Para esto, la interfaz ofrece el método PegaComportamiento(celda). En la implementación el desarrollador debe reconstruir el comportamiento original en la celda pasada como parámetro en base a los datos que éste guardo en el objeto.

6.5.1. Panel contextual: copiado, cortado, pegado y eliminado

Se incorporan las 4 opciones mencionadas en el menú contextual, dentro de una sección de operaciones especiales. Los clicks de esos menus se enlazan con los métodos estáticos CopiaCeldasEsp, PegaCeldasEsp, CortaCeldasEsp y EliminaCeldasEsp de la clase SistemaOpCeldas. En la copia y eliminación se recorren todos los controladores y a los que estén asociados a las celdas de la operación, se les envía un mensaje. En el pegado, se cogen los objetos IDatosComportamietnoCelda y se llama al método de pegado en las celdas que correspondan.

Estas 4 opciones del menú contextual y la sección de operaciones especiales solo aparecerán en el menú contextual cuando se esté en modo edición.

6.5.2. Diagrama final

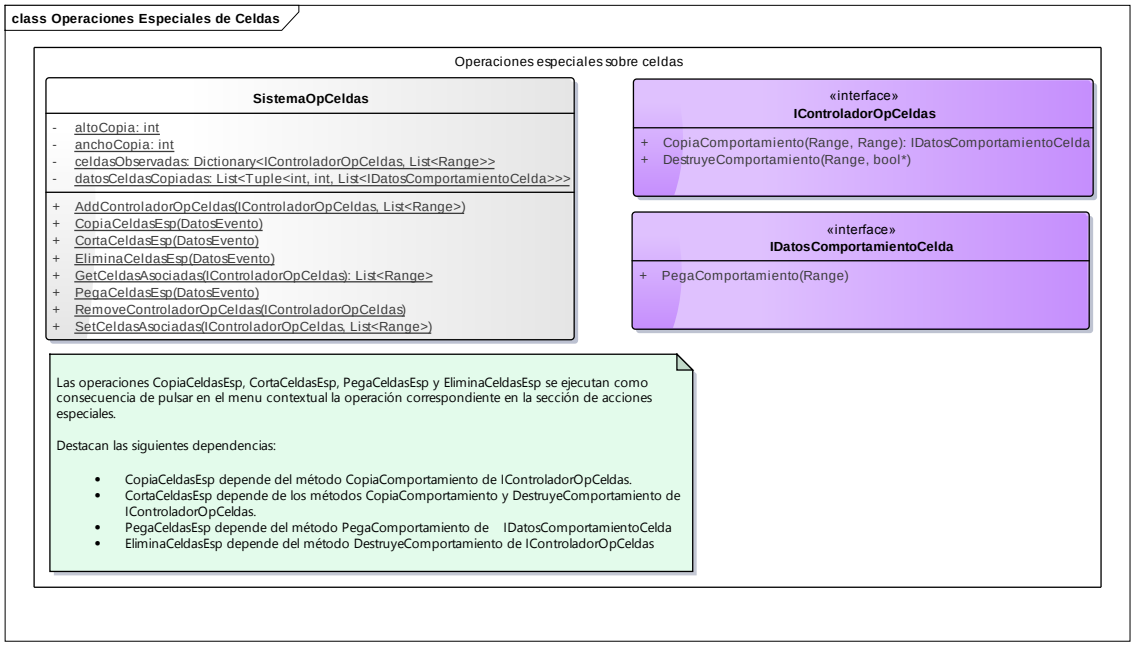

Figura 3. Diagrama de clases del sistema de acciones especiales sobre celdas.

Cabe destacar adicionalmente del diagrama que internamente la clase SistemaOpCeldas almacena el alto y ancho de la copia (para comprobar al intentar pegar que no se salga de la hoja), tiene un diccionario que asocia los controladores a las celdas a las que está suscrito el controlador. También hay una lista de tuplas que representan la posición relativa del objeto IDatosComportamientoCelda y el objeto en sí (de manera que se puedan pegar en su sitio correcto). Finalmente, en todas las operaciones se manejan listas de objetos Range, esto es porque es posible que un controlador esté suscrito a celdas de diferentes hojas.

6.6. Nombre de la aplicación

Es común que el desarrollador Excel quiera que se muestre el nombre de su aplicación Excel dentro de la aplicación. De acuerdo a las aplicaciones de simuladores que he visto, esta información suele venir incluida en un panel de información de la aplicación (que será el que contenga también los datos de los simuladores). Se utilizará el actions pane para mostrar esta información.

Para que el desarrollador pueda modificar este valor, debe incluirse en algún archivo en la aplicación. Por consistencia, este parámetro se incluirá en el archivo ParametrosGenerales.resx. Y será leído directamente de ese archivo. Para esto, Visual Studio incluye la clase ParametrosGenerales, que permite acceder directamente a estos valores.

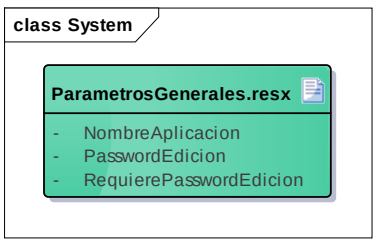

Figura 4. Diagrama mostrando los contenidos del archivo ParametrosGenerales.resx

6.7. Controles persistentes y tareas periódicas

En este proyecto, se llamará controles a los sistemas independientes que realizan cambios en el documento Excel. Por ejemplo, un sistema de monitorización sería un control Excel, al igual que un sistema que permitiese añadir gráficas.

Estas funcionalidades siempre se proporcionan de manera bastante independiente, es decir, que podemos eliminar un sistema, y los demás seguirán funcionando sin problemas (en caso de que ambos sistemas compartan una clase, obviamente esta clase no la podemos eliminar del proyecto).

Se quiere conseguir que el desarrollador Excel pueda construir controles persistentes (es decir, que mantienen datos que se conservan tras el cierre de la aplicación) que sean reutilizables (que se puedan copiar de un proyecto antigua y pegarlos sin demasiado coste de desarrollo). Al mismo tiempo, también se quiere conseguir reducir el coste de desarrollo de estos controles.

Los elementos que comúnmente se encuentran en estos controles son:

- Un botón Ribbon (o en el actions pane o alguna otra parte del documento), que al ser pulsado acciona un código.
- Un código que se ejecuta como consecuencia de pulsar el botón, que crea ventanas WindowsForms, y suele añadir algún comportamiento al documento (una gráfica, una celda que monitoriza una variable, etc…).
- Un gestor de elementos. Este gestor lo que permite es que un conjunto de clases de un tipo puedan persistir al cierre del documento.
- Objetos individuales que son los que realmente realizan la acción y que son gestionados por el gestor.
- Componentes individuales. Estos componentes generalmente se ejecutan desde la primera vez que se abre el documento y realizan una serie de tareas sobre el documento a lo largo del programa. Por ejemplo, en vez de tener un gestor y un montón de elementos que escriben el valor de una variable a una casilla, podría haber un componente que hiciese todo el trabajo.
- Clases auxiliares que ayudan a implementar la funcionalidad. Estos elementos lo único que hacen es separar la funcionalidad de los elementos anteriores en clases separadas, pero no añaden nada, por eso, estos elementos no se tendrán en cuenta, ya que la funcionalidad es la misma (no aportan nada al problema).

No todos estos elementos se encuentran en todos los controles, pero generalmente los controles se componen de un subconjunto de los elementos mencionados anteriormente.

Consiguiendo reducir el tiempo de desarrollo de estos elementos, afrontando los problemas de persistencia que tengan y reparando los problemas de reutilización que se presentan en cada uno de ellos, se conseguirá que sea posible construir controles persistentes de manera rápida y reutilizable.

El punto de la reutilización tiene que ver mucho también con el manejo de los simuladores y sus conexiones, pero este tema se tratará en la sección siguiente, no en esta.

6.7.1. Botón del control (generalmente Ribbon) y acción tras la pulsación

Añadir un botón y asociarle un código se realiza con un par de clicks y no tiene ninguna complicación. Reutilizar estos elementos es simple, basta con mirar las características del elemento en otro proyecto, o, en el caso de barras Ribbon, se podría copiar el código XML. Estos elementos generalmente lo único que tienen diferente es el id de la imagen que usan para su representación. Una forma de hacerlos reutilizables sería, por ejemplo, guardar donde sea que se quieran guardar los elementos para la reutilización, un archivo con los nombres de las imágenes utilizadas para cada botón (generalmente imágenes predefinidas por el sistema).

Una vez está asociado el código al botón. Este código suele crear ventanas Windows Forms para la recogida de datos y tras recoger los datos suele ejecutar alguna acción en algún gestor o en algún componente. Las ventanas en sí son muy simples, y realmente no hay nada que se pueda simplificar, todo es específico del control. Sin embargo, tanto las ventanas como el código del click puede ser interesante reutilizarlos. Para esto, basta con ofrecer un mecanismo estándar. La forma en que se propone que se haga esto es la siguiente:

- Las ventanas Windows Forms, si tienen un botón de aceptar y uno de cancelar, el botón de cancelar tendrá como DialogResult el valor Cancel, y el de aceptar tendrá como valor de DialogResult el valor OK (esto se cambia en el editor). De este modo, al mostrar las ventanas con ShowDialog, se podrá usar el valor devuelto para saber si se pulso aceptar (el resultado vale OK), o se canceló o pulso la x (el resultado vale algo distinto a OK).
- Las ventanas Windows Forms generalmente son para especificar las características particulares del elemento que se quiere añadir al Excel. Por ejemplo, si el botón añade una monitorización, se especificaría en la ventana el periodo y la variable a monitorizar. En este caso, lo recomendable es que la ventana no haga más que recoger los datos. Después, desde el código que creo la ventana se sacarían los datos de la ventana. Para poder recogerlos, se sugieren 2 opciones: ofrecer propiedades públicas en la ventana que permitan obtener estos valores, o pasar un objeto en el constructor y que dentro de este objeto se rellenen estos datos recogidos.
- Finalmente, para la acción que se ejecuta al pulsar el botón en sí, lo mejor es envolver el código dentro de una clase, que tenga un único método estático llamado Run. El nombre de la clase, para que sea fácilmente identificable debería ser Operacion[Nombre]. Por ejemplo, para el control de monitorización se tendría una clase llamada OperacionMonitorizar, con un único método estático llamado Run, que crearía las ventanas y crearía una monitorización en base a los datos recogidos. Después el código asociado al botón de monitorizar lo único que haría es OperacionMonitorizar.Run(). Haciendolo muy fácil de reutilizar.

6.7.2. Nomenclatura de Componentes y Gestores

Para que sean fácilmente identificables, se recomienda llamar a los Componentes Componente[Nombre], y a los gestores Gestor[Nombre]. Por ejemplo, ComponenteMonitorizacion.

6.7.3. Inicialización, terminación y persistencia

Al utilizarse Cached (o cualquier otro sistema de persistencia de datos de Excel), a la hora de cargar los datos, el constructor vacío se ejecuta y después se carga el valor de cada una de las propiedades y campos. Eso significa que al ejecutarse el constructor no está cargado el valor de las propiedades ¿Qué ocurre si hay que ejecutar código de inicialización que depende de estas propiedades? También a veces es necesario realizar alguna actividad al cerrarse el documento Excel (como llamar a la operación de cerrar cliente del simulador).

Adicionalmente, todos estos elementos deben poder marcarse como persistentes. Para facilitar la reutilización, estas 3 operaciones (inicialización, finalización y marcado como persistentes) deben realizarse ofrecerse siempre de la misma manera.

Para la inicialización y la finalización, se utiliza la interfaz IInitializable, donde se han incluido 2 métodos, Initialize y Terminate(). Cuando el desarrollador implemente un gestor, componente o elemento gestionado, deberá implementar esta interfaz. En cuanto a la persistencia, para los elementos gestionados la persistencia es manejada por el gestor (se explica más adelante como), y para los demás, se deberá implementar el patrón Singleton que viene en la sección de Almacenamiento de datos persistentes. Después, para hacer los datos persistentes, el usuario de este componente o gestor deberá envolver el objeto Instance de la clase de la manera que se explica en la sección de Almacenamiento de datos persistentes.

Initialize se utiliza para ejecutar el código de inicialización, por ejemplo, para suscribirse al sistema de operaciones de celdas, para suscribirse al evento de cierre de hoja, etc… Y la función Terminate para ejecutar el código de finalización, finalizar el cliente del simulador, etc… Para componentes y gestores, la idea es que estos métodos sean llamados explícitamente en la función de arranque y cierre del documento respectivamente. Por ejemplo, se haría ComponenteMonitorizacion.Instance.Initialize() para inicializar el componente de monitorización.

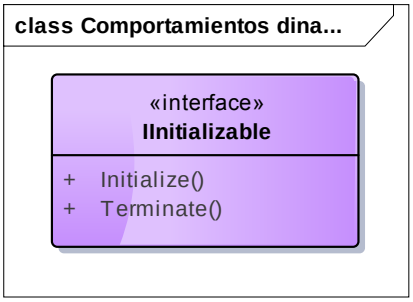

Figura 5. Interfaz IInitializable

6.7.4. Timers

Es muy común que los componentes, gestores y elementos gestionados realicen una actividad periódica. Existen muchos tipos de timers distintos en c#, y los que se deben usar en la programación en Excel son los de tipo System.Windows.Forms.Timer (esto es porque trabajan en la hebra principal). Estos timer requieren un trozo de código considerable, y tienen algunos elementos extraños en su manejo, como tener que utilizar el tag para guardar objetos.

Dado que es común realizar una actividad periódica, y el manejo de timers puede no ser tan simple, se proporciona en la API un sistema de timers mucho más simple, que hace justamente solo lo que se quiere, ejecutar una tarea periódica. Esta funcionalidad se proporciona en la clase SistemaCompDinamicos, que hace uso de System.Windows.Forms.Timer.

Esta clase contiene los métodos estáticos AddTareaPeriodica, que recibe el método a llamar cada periodo ms, el periodo, el objeto pasado en cada llamada al método, y devuelve en un parámetro de salida el id de la tarea (un long). Además, hay un método RemoveTareaPeriodica que elimina la tarea con el id pasado, y IsTareaPeriodica, que devuelve si hay una tarea asociada al id pasado como parámetro.

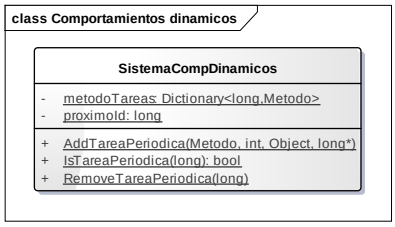

Figura 6. Clase SistemaCompDinamicos. Se encarga de manejar las tareas periódicas.

6.7.5. Manejo de tipos de Excel

Independientemente de si se ejecutan periódicamente, en su ejecución, estos elementos principalmente lo que hacen es interactuar con celdas, gráficos, etc… Y es en estas operaciones donde se va todo el tiempo de desarrollo. Estos elementos tienen muchos problemas, y los desarrolladores tienen que gastar horas resolviendo el problema para cada caso particular.

Para el manejo de estos elementos se proporcionan las clases RangeSerializable, WorksheetSerializable, GraficoFlotanteSerializable y GraficoHojaSerializable, que envuelven las clases Range, Worksheet, graficos incrustados en una hoja y hojas de gráfico, respectivamente.

Estos son los tipos más problemáticos. Los otros (imágenes, etc…), no son tan difíciles de mantener persistentes. Generalmente, ni si quiera se conservan en la hoja, sino que deben crearse cada vez (con lo cual es difícil hacerlos persistentes de manera genérica, pero muy fácil hacerlos persistentes para el caso específico).

6.7.5.1. Problema principal, persistencia

El primero problema es que Worksheet, Range ChartObject, etc… no son serializables. Si uno tiene una propiedad hoja de tipo Worksheet y se marca como pública (y se pone Cached sobre el objeto en ThisWorkbook), al abrir el documento esa propiedad valdrá null (en vez de seguir apuntando a la misma hoja). Tampoco basta con guardar el nombre de la hoja, ya que este puede cambiar. Lo mismo pasa con Range y los demás tipos de Excel, que es imposible conservarlos.

Para todos estos elementos se debe seguir un mecanismo completamente distinto para garantizar que se apunta al mismo objeto tras el cierre y apertura del documento, y estos mecanismos son complicados y llevan muchas horas. Para solucionarlo, RangeSerializable y las demás clases son versiones serializables de los objetos que encapsulan. De modo que si se guarda una propiedad de tipo WorksheetSerializable, al cerrar y abrir el documento esa propiedad seguirá apuntando a la misma hoja. Esto hace que el desarrollador solo tenga que centrarse en el manejo de los objetos en sí, y no en cómo hacerlos persistentes, simplificando enormemente la programación Excel con estos elementos. Estos elementos no reimplementan los métodos de los objetos que encapsulan, sino que ofrecen una propiedad a través de la que se puede acceder al objeto encapsulado.

6.7.5.2. Otros problemas

Estas clases también están diseñadas para resolver otros problemas comunes del manejo de estos tipos, haciendo que la programación sea solo la parte específica de cada control (en vez de horas resolviendo los problemas).

Primero, se soluciona el problema de los castings entre los tipos host ítem (tipos que extienden la funcionalidad de objetos nativos Excel) y los tipos nativos, que suele ser complicada y diferente en cada clase. Para eso, las clases ofrecen diferentes constructores, unos que aceptan la versión nativa y otro que acepta la versión host ítem y realiza el casting correspondiente. Del mismo modo, las clases ofrecen diferentes funciones Set para establecer el objeto envuelto (una para el objeto nativo y otra para el objeto host ítem). La conversión en la otra dirección, de objeto nativo a host ítem, es muy sencilla, y por eso no se ofrece ningún mecanismo. Para esta conversión basta con hacer Globals.Factory.GetVstoObject(objetoNativo) y se obtiene como resultado el objeto host ítem.

Otro problema que surge que también se repara es que el método Equals de la clase Range no compara que los 2 objetos apuntan a las mismas celdas, sino que simplemente compara por referencia. Para solucionar esto, se proporciona la clase MismasCeldasComparer, que extiende IEqualityComparer<RangeSerializable>, y solo devuelve true en el equals si tienen las mismas celdas. Para usarlo se debe pasar como comparer en los diccionarios y hashsets.

El último problema que se soluciona es el de la eliminación de los objetos. Cuando una hoja es eliminada, los objeto Range y Chart asociados lanzan una excepción al acceder a cualquiera de los campos. Para solucionar esto, los objetos RangeSerializable y GraficoHojaSerializable establecen una propiedad Worksheet en el momento de la asignación del objeto envuelto, lo cual permite posteriormente comprobar si un objeto pertenece a una hoja eliminada y suscribirse al evento de hoja eliminada, impidiendo cualquier acceso al objeto eliminado e impidiendo así que salten las excepciones.

6.7.5.3. Diagrama

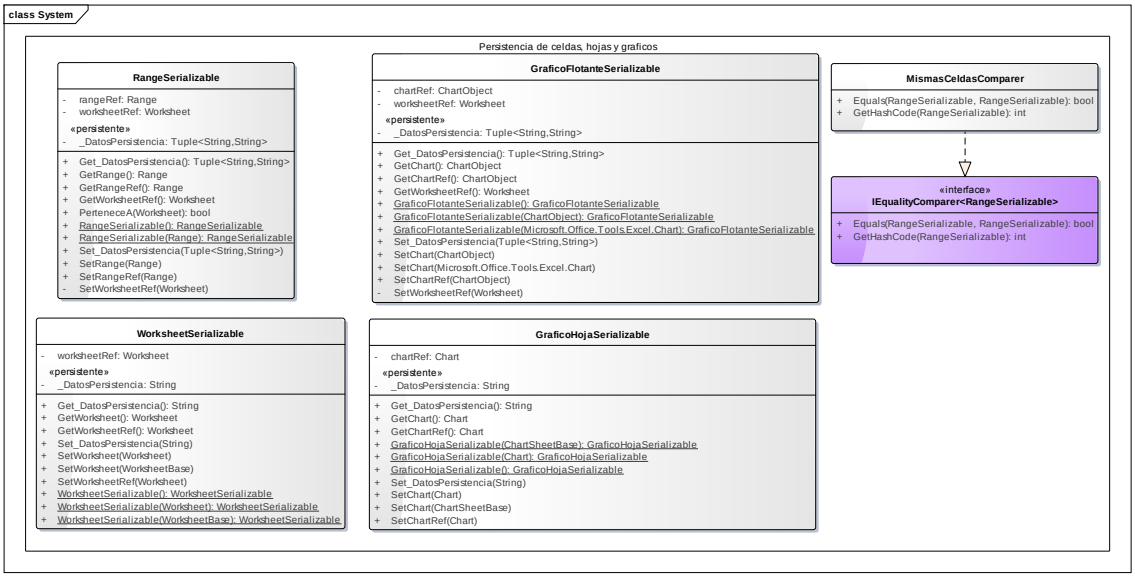

Figura 7. Diagrama con las clases envoltorio de Range, Worksheet y para los charts flotantes y las hojas de gráfico.

6.7.6. Acceso a ThisWorkbook y Application

En la programación de aplicaciones Excel es común necesitar funciones de las ofrecidas por la las instancias únicas de ThisWorkbook y Application. Para acceder al objeto ThisWorkbook es necesario hacer Globals.ThisWorkbook, que es una propiedad cuyo nombre depende del nombre de la clase ThisWorkbook. Si en vez de ser ThisWorkbook es WorkbookPruebas, entonces la propiedad se llama WorkbookPruebas.

Por eso, se proporciona una propiedad WorkbookExcel en SimuGlobals cuyo nombre no cambia al cambiar el nombre de ThisWorkbook. Esto no soluciona totalmente el problema, porque si se cambia el nombre de ThisWorkbook, la clase SimuGlobals no compilará. Sin embargo, si es completamente inevitable cambiarle el nombre de la clase, bastará con cambiar la definición del get de esta propiedad en SimuGlobals en vez de tener que cambiar código en la mitad de las clases del sistema.

También se ofrece una propiedad Application para no tener que ir a través de ThisWorkbook para llegar al objeto Application. Para poder acceder a los objetos de herramientas desde un lugar común.

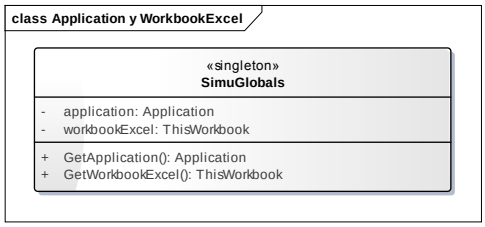

Figura 8. Diagrama mostrando las 2 propiedades mencionadas de SimuGlobals

6.7.7. Gestores de elementos

Los gestores de elementos son los clases que se encargan de que un conjunto de elementos de un mismo tipo sean persistentes. Por ejemplo, podría haber un conjunto de elementos que escriben cada uno el valor de una variable periódicamente en una casilla Excel. Cada uno de estos objetos estaría almacenado en el gestor, el cual se encargaría de llamar a los métodos Initialize y Terminate y de hacerlos persistentes.

Para la implementación de los gestores, se proporciona la clase GestorDeElementos. Esta clase es una clase genérica, y recibe un parámetro tipo T, que indica de que tipo son los objetos que gestiona el gestor (el tipo de los objetos debe implementar IInitializable). El gestor guarda una lista interna, que puede ser accedida a través de la propiedad Elementos y proporciona los métodos AddElemento y RemoveElemento, que añaden y eliminan un elemento al conjunto gestionado respectivamente. También ofrece el método ClearElementos, que vacía el conjunto de elementos gestionados. La lista Elementos es pública, lo cual permite que si el gestor persiste, los objetos gestionados también lo hagan.

Finalmente, GestorDeElementos implementa implementa IInitializable. En el método Initialize llama al método Initialize de los objetos gestionados, y en la función Terminate llama al método Terminate de todos los elementos gestionados.

Por último, se ofrecen 2 versiones de la clase GestorDeElementos, una con un único parámetro tipo, que utiliza el comparador por defecto, y una con 2 parámetros tipo, que recibe un comparador en el constructor, y que utiliza dicho comparador para la lista interna.

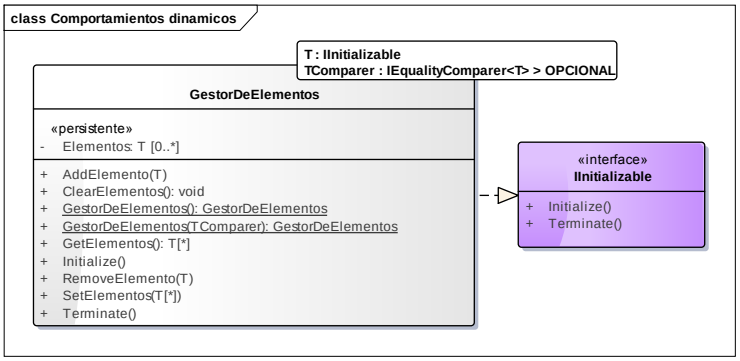

Figura 9. Clase GestorDeElementos.

En cuanto al uso, lo normal es que al crear un Gestor, se incluya una propiedad GestorDeElementos pública que se utilice para gestionar los elementos gestionados. Después, es habitual que estos gestores tengan un método para crear objetos y otra para eliminarlos (por ejemplo, CreaMonitorización). También se puede implementar un gestor heredando la clase GestorDeElementos, pero es más sucio, ya que los métodos internos quedan expuestos.

6.7.8. Eventos especiales y cambio de modo

Para comprobar en qué modo de uso está el Excel actualmente, como ya se ha explicado antes, se utiliza la propiedad ModoActual de SimuGlobals. Esto permite ver antes de ejecutar una acción en qué modo se está. Sin embargo, es posible que se quiere realizar una acción en el momento en que se produce el cambio de modo. Por ejemplo, para cambiar los botones de la barra Ribbon de visibles a no visibles.

Esta es una situación muy habitual, por eso, se proporciona en SimuGlobals un evento de c# llamado ModoCambiado, que es llamado siempre que se cambia de modo. Además, se incluye en ThisWorkbook un método llamado SimuGlobals_ModoCambiado que es llamado cada vez que se produce un cambio de modo y un método ConfiguraSegunModo que es igual, pero que también es llamado en el arranque.

Además de este evento, hay otro evento que habitualmente hace falta pero que Excel no proporciona, es el evento de hoja cerrada. Este evento es implementado en SimuGlobals y también es proporcionado en la clase ThisWorkbook. Lo que se hace es que se guarda una lista de las hojas que estaban abiertas la última vez que se comprobó (propiedad HojasVivas). Después, en los eventos de hoja activada y desactivada (a los que se suscribe SimuGlobals en la función Initialize), se mira cuáles de las hojas que ya no están en el conjunto de abiertas, y se hace una llamada al evento SheetClosed con las hojas cerradas y las abiertas.

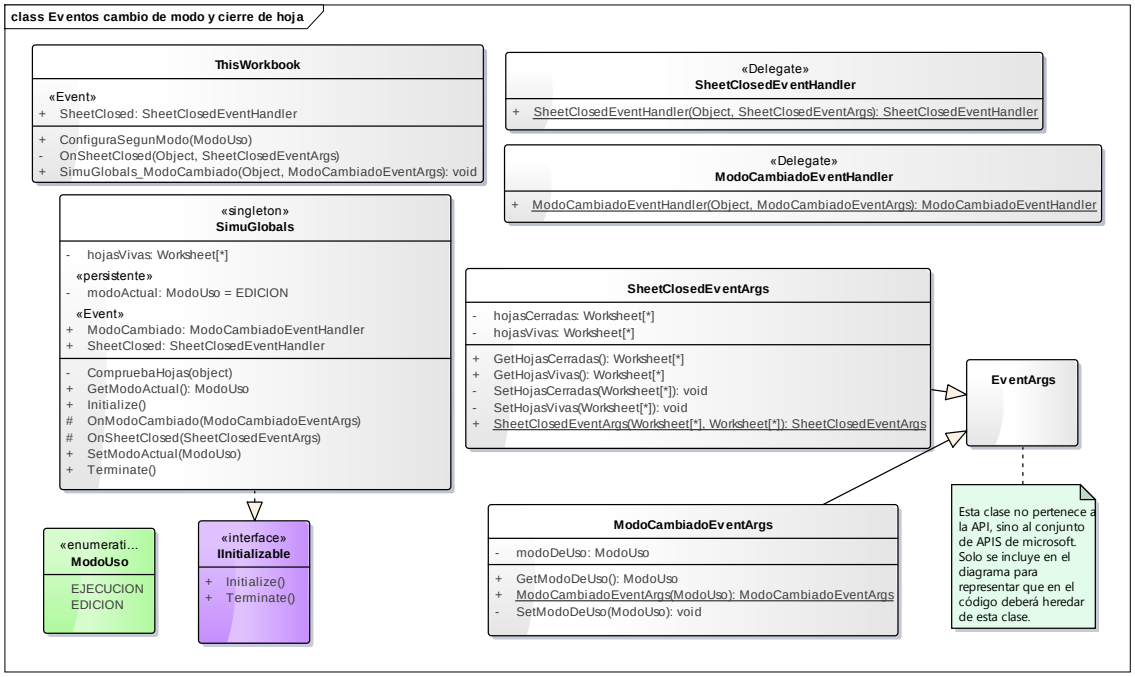

Figura 10. Diagrama de clases de la sección de eventos añadidos en este sistema

6.7.9. Tareas periódicas persistentes

En uno de los casos de uso se hablaba de tareas periódicas persistentes. Esto se puede conseguir fácilmente con los elementos que se han descrito anteriormente. Basta con utilizar una clase para realizar la tarea, por ejemplo un componente. En el componente, en la función initialize se crearía el timer, y se ejecutaría la acción. Los datos serían persistentes, porque como todo componente, se crearía una propiedad que envolviese el Instance marcada con Cached y se llamaría al initialize y terminate, los cuales se encargarían de que en cada arranque se crease el temporizador. Consiguiendo así una tarea periódica persistente.

6.8. Manejo de simuladores

6.8.1. Conexiones en componentes y gestores

A la hora de implementar un componente o gestor, dado que todo el desarrollo de Excel gira entorno a los simuladores, lo normal es que este gestor o componente haga uso de una conexión a un simulador y envíe una serie de mensajes a este.

Para que el componente sea realmente reutilizable, debería ser capaz de trabajar con cualquier configuración de conexión, siempre y cuando esta configuración utilice la misma interfaz para el binding que el cliente usado por el componente o gestor (esto es debido a limitaciones técnicas).

Para conseguir facilitar la especificación de esta conexión, se proporciona una clase llamada PropiedadesDeConexion, que contiene información del binding y del endpoint, que son los 2 elementos principales que definen la configuración de una conexión WCF. Además, la clase ofrece un método CreaClienteWCF<T> que crea un cliente WCF del tipo pasado como parámetro (por ejemplo SimulatorServiceClient) usando los datos de la clase PropiedadesDeConexion.

Para facilitar la configuración de estos componentes, se ofrece una propiedad en SimuGlobals llamada ConfigBaseSimuladores, la cual los componentes y gestores (si están bien implementados) usarán como configuración de conexión en caso de no especificarse explícitamente ninguna. La idea es que en estos componentes y gestores haya una propiedad ConfigSimulador. Si esta propiedad es igual a:

- Null. Entonces se usa la configuración que se encuentra en ConfigBaseSimuladores.
- PropiedadesDeConexion.ClientDefault, entonces usa la configuración por defecto registrada en visual studio para el cliente que utilice la clase
- Otro valor, entonces esta es la configuración que usa para sus tareas.

ConfigBaseSimuladores será inicialmente igual a PropiedadesDeConexion.ClientDefault, mientras que las propiedades ConfigSimulador comenzarán a null. Si el desarrollador cambia ConfigBaseSimuladores, entonces esta configuración se aplicará a todos los componentes y gestores que tengan ConfigSimulador igual a null.

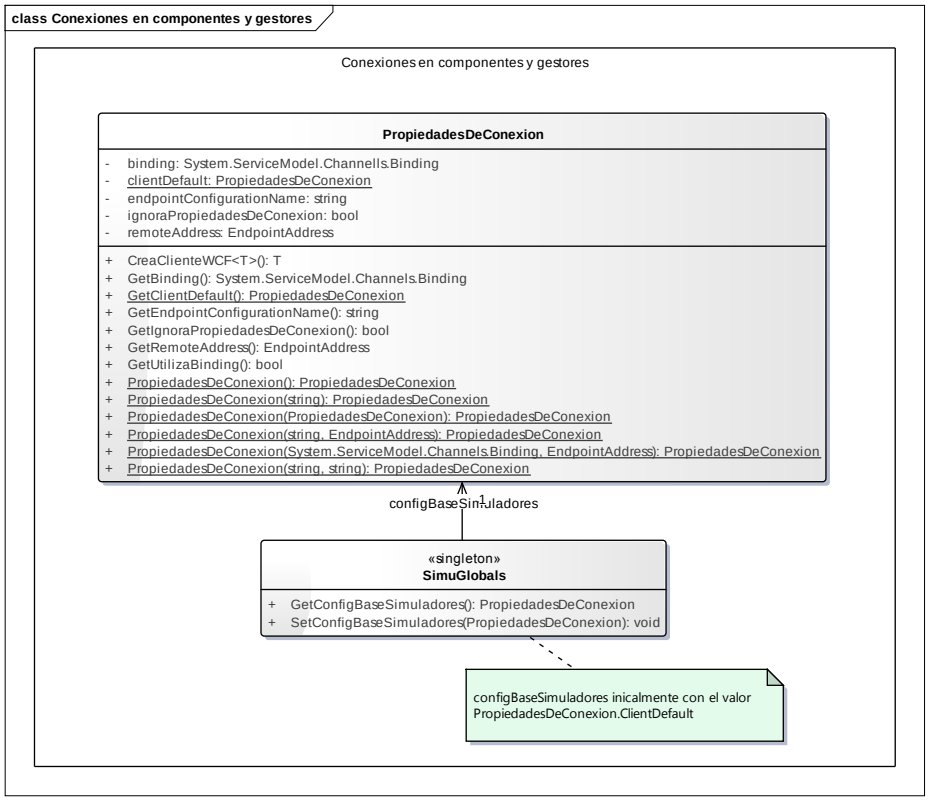

Figura 11. Diagrama de la sección de conexiones en componentes y gestores.

6.8.2. Consulta del estado: accesible o no accesible

Independientemente de los estados propios de cada simulador y aplicación, todo simulador tiene 2 estados que siempre son los mismos, accesible (que es posible enviarle mensajes http básicos, y que responda) y no accesible.

Si una aplicación Excel intenta enviar un mensaje (por ejemplo, para solicitar el valor de una variable) a un simulador no accesible, la aplicación se quedará pillada durante unos milisegundos (intentando acceder al simulador), si hay muchas peticiones, se pillará mucho más tiempo. Por esa razón, y para permitir mostrarlo en el panel de información, es muy importante poder consultar si un simulador es accesible sin que se pille la aplicación.

Esta funcionalidad se proporciona con la clase MonitorDeConexiones, que funciona con cualquier URL y con cualquier cliente WCF. Para permitir que las clases que usan esta clase puedan consultar el estado de manera inmediata, se realizan internamente consultas periódicas del estado de los simuladores. Para permitir que diferentes sistemas puedan utilizar esta clase de manera independiente, cada conexión tiene asociada un contador.

La clase contiene los métodos AddMonitorizacionDeEstado (conexion, int? timeout) y RemoveMonitorizacionDeEstado (conexion), que permiten añadir una monitorización de estado de una conexión y eliminar una monitorización respectivamente. Cada vez que se hace un Add se incrementa en 1 el contador de la conexión, y cada vez que se hace remove se decrementa. Si llega a 0, se deja de observar la conexión. En el primer método el parámetro timeout es opcional, y modifica el timeout asociado a la conexión.

La clase presenta además la propiedad Timeout, que representa el tiempo de espera utilizado en las consultas de conexión en caso de no haberse establecido ninguno explícitamente. El timeout también se puede cambiar con el método SetTimeoutMonitorizacion(conexión, int? timeout), y se puede consultar con el método GetTimeoutMonitorizacion(conexion) un valor de null indica que se usa el valor de la propiedad Timeout.

El método EstaSiendoMonitorizada permite comprobar si la conexión está en la lista de conexiones monitorizadas, y HayDatosDeEstado permite comprobar si se ha hecho ya la primera consulta de estado o aún no hay datos.

Finalmente, para consultar el estado, la clase contiene los métodos GetEstadoConexion e IntentaAccesoAConexion. El primero devuelve el último valor obtenido para la conexión, y en caso de no haber ninguno, se para esperando a que acabe la primera consulta. El segundo ignora los datos contenidos y realiza una consulta de estado desde 0.

Figura 12. Clase MonitorDeConexiones

6.8.3. Panel de estado de los simuladores

Es común que se muestre en las aplicaciones Excel para simuladores el estado de los simuladores de manera visual, por eso, se ofrecerá en el proyecto esta característica de manera que ya no haga falta implementarlo cada vez.

El estado de los simuladores se muestra en una sección del ActionsPane, debajo del título del proyecto, y su contenido es gestionado por la clase ActionsPaneInformacion. Esta clase extiende de UserControl. Es instanciada, y su instancia es añadida al ActionsPane en la función de inicio del libro Excel. Como los objetos de tipo UserControl tienen una cantidad enorme de propiedades y métodos, los métodos propios serían muy difíciles de encontrar. Por eso, esta clases se hará que implemente una nueva interfaz, IInformacionDeEstado que solo contendrá los métodos y propiedades de la API.

Para permitir que se pueda interactuar con el panel desde el código del desarrollador Excel, se ofrece una propiedad pública PanelInformacionDeEstado en ThisWorkbook y en SimuGlobals que devuelve el objeto ActionsPaneInformacion convertido a IInformacionDeEstado. Este panel solo actualiza los estados en modo ejecución, en modo edición muestra el último estado detectado en gris.

6.8.3.1. Especificación y consulta de servicios mostrados

Todas las aplicaciones tienen simuladores distintos, y por tanto los simuladores no pueden estar precargados en el panel y deben ser añadidos explícitamente. Para esto, ActionsPaneInformacion contiene los métodos AddServicio, RemoveServicio, RemoveServicioById, ContainsIdServicio, GetNombre, GetNombreById, GetServicio, GetServicioById y TryGetId, y las propiedades NumServicios y IdsEqualityComparer.

AddServicio permite añadir un servicio al panel, y recibe como parámetros:

- La posición en la que se quiere añadir el elemento en el panel (ordenado de arriba abajo). Por ejemplo, la cero significa que se quiere añadir arriba del todo. La 1, justo debajo del primero, etc…
- Un Object, que representa el id único del elemento (utilizado para la búsqueda de los elementos a través del id). Este parámetro es opcional, pero si se especifica debe ser único.
- Otro string, que representa el nombre del elemento que se mostrará en el panel.
- Por último, o un String o un objeto ClientBase que representen el servicio para el que se quiere que se muestre el estado.

RemoveServicio elimina el servicio que se encuentra en la posición pasada como parámetro, y RemoveServicioById elimina el servicio que tiene como id el id pasado como parámetro.

ContainsIdServicio devuelve true si existe algún servicio en la lista de servicios mostrados cuyo id coincida con el pasado como parámetro.

GetNombre devuelve el nombre del servicio que se encuentra en la posición pasada como parámetro y GetNombreById devuelve el nombre del servicio cuyo id es igual al pasado como parámetro.

GetServicio devuelve un objeto de tipo Object (que será un objeto String o un objeto ClientBase según lo que pasase el usuario en el método Add) que representa el servicio que se encuentra en la posición pasada como parámetro, y GetServicioById hace lo mismo, pero devuelve el servicio que tenga como id el id pasado como parámetro.

TryGetId devuelve el id del servicio encontrado en la posición pasada como parámetro dentro de un parámetro out. En caso de no haber id para esa posición, devuelve false. En caso de sí encontrar id, devuelve true.

Por último, la propiedad NumServicios devuelve el número de servicios mostrados e IdsEqualityComparer permite al desarrollador Excel cambiar el comparador que se usa para hacer el equals sobre los ids y usar uno propio.

En general, el sistema funciona de una manera muy similar a una lista. No hay huecos entre los elementos, y no es posible añadir elementos en una posición menor que 0 o mayor que NumServicios.

6.8.3.2. Especificación de estados

En el panel se muestra el nombre y el estado de los simuladores y servicios que han sido añadidos. Por defecto hay 2 estados, desconectado y accesible. Estos estados los detecta automáticamente el sistema realizando para ello una consulta http a la dirección del servidor.

Es común que haya estados adicionales. De hecho, en la interfaz genérica para simuladores que se describe más adelante se puede ver que hay 2 subestados de accesible, que son corriendo y parado. Los simuladores generalmente seguirán esa interfaz, pero asumir que estos 2 subestados estarán presentes en cualquier servicio haría que el sistema fuese menos flexible, ya que no podría ser usado con servicios de otros tipos.

Para solucionar estos problemas, la clase ofrece los métodos SetControladorDeSubestados (int posicion, bool subestadosDeAccesible, IControladorDeSubestados controlador), SetControladorDeSubestadosById (object id, bool subestadosDeAccesible, IControladorDeSubestados controlador) y SetControladorDeSubestadosAll(bool subestadosDeAccesible, IControladorDeSubestados controlador). Estos métodos permiten asociar 2 contoladores de subestados para cada servicio (uno para el estado de desconectado y uno para el de accesible), y 2 generales que actuarían sobre todos los servicios.

Cuando el sistema comprueba el estado de los servicios para mostrarlo en el panel, tras comprobar si es accesible o está desconectado, llama al controlador específico de este servicio (si existe) y le pide subestados, después, si este le devuelve que no tiene información, le pide los subestados al general (si existe). En caso de que ninguno le dé información, pone como estado el estado que obtuvo en la consulta inicial. Para desasociar el controlador de subestados, se llama al mismo método pero pasando null como controlador.

La clase también ofrece los métodos GetControladorDeSubestados, GetControladorDeSubestadosById y GetControladorDeSubestadosAll para obtener los controladores asociados a cada servicio y el general (para uno de los 2 estados).

La interfaz IControladorDeSubestados extiende de IInitializable, y añade el método DeterminaSubestado (bool esAccesible, int posicion, object servicio, out String stringEstado, out Color color), que devuelve true si el controlador quiere que se reemplacen los datos de estado actuales (el nombre y color) por los datos de estado pasados en los parámetros stringEstado y color, y false si quiere que se mantenga como está.

6.8.3.3. Subestados normales de simuladores: Detenido y ejecutando.

Según la interfaz genérica que se describirá más adelante en este documento, existen 2 estados principales de un simulador que está funcionando, detenido y ejecutando.

Como la idea es que los desarrolladores usen esta interfaz para los simuladores, lo normal es que los simuladores tengan estos estados. Por eso, se proporciona una clase,

CtrSubestadosSimulador, que implementa IControladorDeSubestados e incluye ya ésta comprobación de estados adicionales. El desarrollador simplemente tiene que asociar una instancia de esta clase a su simulador tras añadirlo al panel para tener los estados adicionales.

La clase recibe en el constructor un objeto de tipo PropiedadesDeConexion, que utiliza para creación de un objeto SimulatorServiceClient (explicado más adelante) para la detección de los subestados. Es posible que haya otros subestados de accesible. Por eso, en CtrSubestadosSimulador los campos son protegidos y los métodos son virtuales, lo que hace que sea posible extender los estados con facilidad heredando la clase.

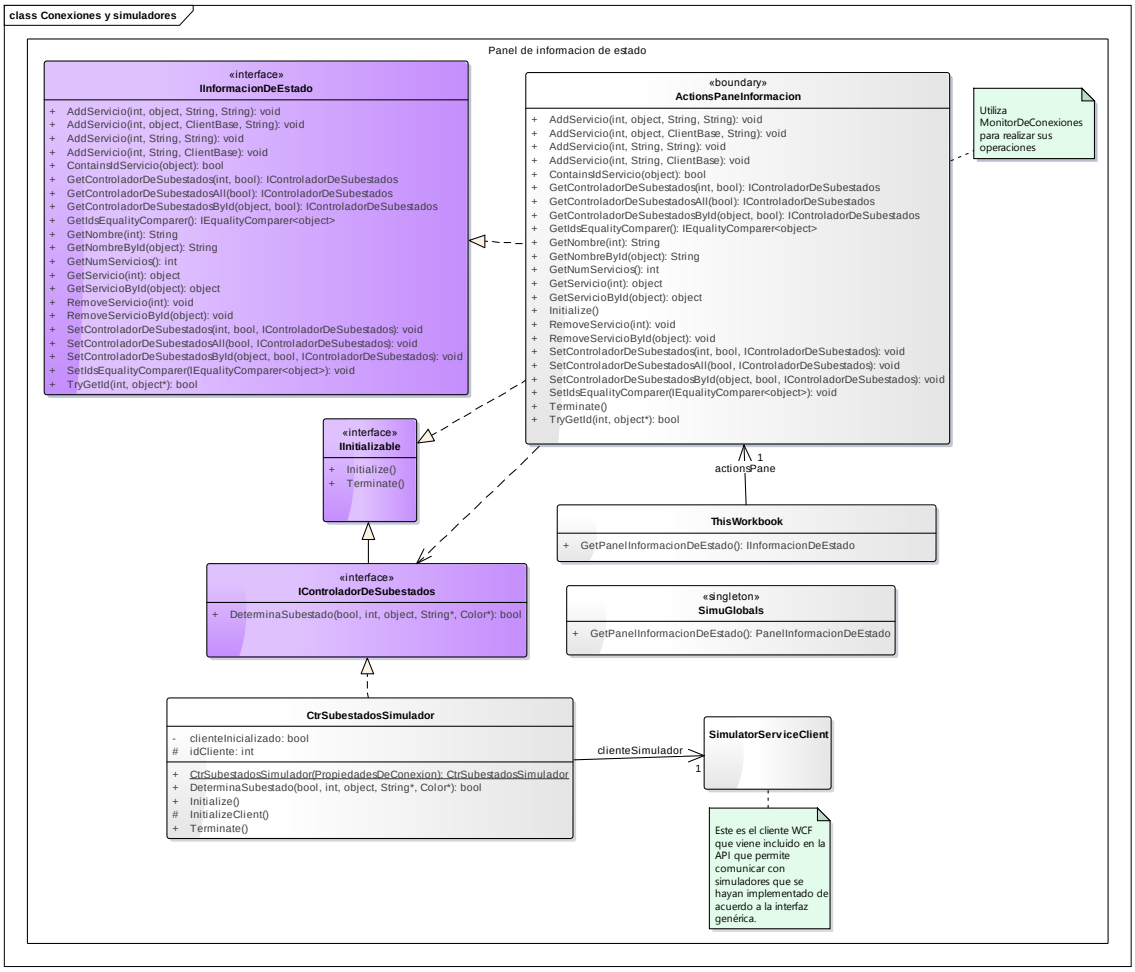

Figura 13. Diagrama de clases de la sección panel de estado de los simuladores

6.8.4. Panel de introducción de datos de simulación

Es común poner en el arranque del documento Excel un panel en el que se recogen los datos de configuración de la conexión al simulador. Esto puede hacerse por muchas razones. Por ejemplo, si el puerto en el que normalmente trabaja el simulador está ocupado, será necesario lanzarlo en otro puerto. En esta situación, si no se piden los datos en el arranque del documento, sería imposible ejecutar la macro.

Se incluye en el proyecto un panel que contiene las opciones típicas de configuración de estos simuladores. Este panel se llama PanelDatosDeConexion. Contiene un campo para elegir el simulador sobre el que se quiere trabajar, uno en el que especificar la dirección ip, y otro en el que especificar el puerto.

Este panel acepta en el constructor una lista de nombres de simuladores (para el campo de elegir simulador), y un valor por defecto para el ip y el puerto. Además, tiene una propiedad DatosDeConexion que permite obtener los datos que ha introducido el usuario. En caso de que el usuario haya pulsado aceptar, se devuelve DialogResult.OK.

6.9. Interfaz genérica para simuladores

Entre los objetivos del proyecto, estaba ofrecer una interfaz genérica para simuladores. La idea era que todos los simuladores usados en los proyectos Excel implementasen dicha interfaz. De este modo, los controles serían completamente reutilizables, y además sería posible reutilizar proyectos completos para simuladores distintos.

La interfaz no puede ser ni demasiado compleja, dado que de ser así, el coste de desarrollar los simuladores sería mayor, ni demasiado simple, dado que sino faltarían opciones y siempre haría falta modificar la interfaz para todos los simuladores.

En caso de tener un simulador que no cumple la interfaz, lo que se haría es crear un servicio intermediario, que transformaría las llamadas y permitiría tratar el simulador como un simulador que cumple la interfaz.

6.9.1. Métodos y tipos de la interfaz

Los simuladores siempre funcionan como servicios que contienen un único simulador corriendo. Todas las clases que acceden al servicio comparten el mismo simulador.

Para poder realizar las diferentes operaciones con el simulador, los usuarios del simulador deben llamar a un método de inicialización, que les da un id de cliente. Después, una vez terminan, para liberar recursos, deben llamar un método del simulador que sirve para eliminar un cliente. Para eso, la interfaz contiene los métodos int Initialize(), y void FinishClient(int idClient).

Los simuladores arrancan en estado detenido, y pueden estar detenidos o corriendo. Además, los simuladores funcionan por pasos, donde en cada paso se avanza en la simulación, y tienen un tiempo de simulación asociado. Estos 3 valores se incluyen en la interfaz en la estructura State, que contiene campos para estas 3 cosas. Además, con el método GetState(int idClient) se obtiene el estado actual.

Los simuladores se suelen poder parar y arrancar. Para esto, se ofrecen los métodos void SendFreeze(int idClient) y void SendRun(int idClient).

Finalmente, los simuladores en su ejecución lo que hacen es modificar en cada paso un conjunto de variables, algunas variables, además de poder consultarse se pueden modificar. Para eso, se ofrecen los métodos void VbleUpdate(int idClient, string nvar, double vble), que permite modificar el valor de una variable, double VbleQuery(int idClient, string vble), que permite obtener el valor de una variable y los métodos List<double> MonitorizeVbles(int idClient, List<string> variables) y List<double> UpdateVbles(int idClient), que permiten establecer un conjunto de variables monitorizadas, de manera que no haga falta hacer un query para cada variable, sino que se hace MonitorizeVbles una vez y a partir de ese momento, cada vez que se haga UpdateVbles se obtendrá el valor de todas las variables marcadas para monitorizar.

El resultado final es la interfaz ISimulatorService:

```
[ServiceContract]
public interface ISimulatorService
{
    [OperationContract]
    /*Initialize all the queues that will be involve in the program*/
    int Initialize();

    [OperationContract]
    void SendRun(int idClient);

    [OperationContract]
    void SendFreeze(int idClient);

    [OperationContract]
    double VbleQuery(int idClient, string vble);

    [OperationContract]
    void VbleUpdate(int idClient, string nvar, double vble);

    [OperationContract]
    List<double> MonitorizeVbles(int idClient, List<string> variables);

    [OperationContract]
    List<double> UpdateVbles(int idClient);

    [OperationContract]
    State GetState(int idClient);

    [OperationContract]
    void FinishClient(int idClient);
}

[DataContract(Namespace = "InteractiveExcelDocsForSims.ISimulatorService")]
public struct State
{
    [DataMember]
    public bool state { get; set; }

    [DataMember]
    public long nStep { get; set; }

    [DataMember]
    public double timet { get; set; }
}
```

Se incluye en el proyecto la clase SimulatorServiceClient, que permite acceder al servicio de ejemplo incluido en el proyecto, y que los desarrolladores Excel deberían modificar para que apuntase a su simulador. Esta clase es usada por elementos del proyecto como el panel de estado.

6.10. ExcelTools

Existen problemas en el manejo de muchos de los elementos de la programación en Excel. Para facilitar la programación, se ofrece la clase ExcelTools, que incluye un conjunto de métodos estáticos que permiten realizar acciones que, si no existiese esta clase, serían muy difíciles de hacer.

6.10.1. Manejo de WCF

Para el manejo de las clases de Windows Communication Foundation, se proporcionan 2 métodos:

- GetUriClienteWCF. Este método recibe un cliente WCF y devuelve la URI asociada a dicho cliente.
- GetDefaultEndpoints. Devuelve el conjunto de datos de los clientes que se encuentran en el archivo app.config. Como clave para los diccionarios se utiliza el nombre del endpoint (que puede ser encontrado en app.config). Este método es muy útil para poder crear configuraciones de conexiones que envíen mensajes a un simulador distinto del apuntado por SimulatorService pero usen el binding de SimulatorService. También sirve para poder realizar pequeños cambios sobre la configuración por defecto a la hora de establecer la configuración para los clientes.

6.10.2. Manejo de objetos Range

Para los objetos Range, hay muchas cosas que con las APIs de Microsoft no se pueden hacer de manera sencilla y que son muy básicas.

La clase ExcelTools ofrece las siguientes operaciones sobre objetos Range:

Cuenta simplificación y características de la celda

- UInt64 TotalCount(Range range). Devuelve el número de celdas del objeto Range. A diferencia de la propiedad Count normal de Range, este método no lanza una excepción cuando se hace sobre selecciones de filas o columnas completas, sino que devuelve el número total de celdas independientemente del tipo de conjunto representado.
- Range SimplificaDireccion(Range range). Simplifica el objeto Range en el sentido de que la propiedad Address del objeto range queda simplificada. Por ejemplo, si uno crea un objeto Range llamando a Evaluate con las siguiente cadena,"A1,A1,A1", realmente la dirección que queda asociada al Range es "A1,A1,A1". Tras llamar al método, la dirección que queda asociada es A1. Se usa sobretodo de manera interna.
- List<Range> SimplificaConjunto(List<Range> ranges). Toma un conjunto de objetos Range, y devuelve un conjunto que tiene un único objeto Range por cada hoja. Por ejemplo, si tenemos 2000 objetos Range que apuntan a celdas distintas de una misma hoja, este método devolvería una lista con un único objeto Range que representaría todas las celdas que estaban contenidas en ese conjunto de 2000. Al igual que el anterior, es sobretodo usado de manera interna, pero dependiendo del tipo de elementos que se desarrolle, puede resultar útil.
- List<Range> ObtenSubconjuntos(Range range). Toma un objeto Range y devuelve una lista que contiene un objeto Range por cada conjunto de celdas que se podrían representar como casilla:casilla, y que por tanto no contienen ninguna "," en su dirección (son por tanto, conjuntos de forma retangular, finitos o infinitos). Es posible que los conjuntos rectangulares se pisen unos a otros.

- void TieneFilasOColumnasCompletasRect(Range range, out bool tieneFilasCompletas, out bool tieneColumnasCompletas). Devuelve si el conjunto de celdas contiene alguna fila o columna completa. El método es solo para objetos Range rectangulares simples (sin ninguna "," en su address).

Limites, ancho y alto

- GetPrimeraColumna, GetPrimeraFila, GetUltimaColumna y GetUltimaFila devuelven la mínima columna, la mínima fila, la máxima columna y la máxima fila en la que hay alguna celda que pertenece al objeto range pasado como parámetro. También hay los mismos métodos pero versión para objetos range rectangulares simplificados.
- GetAncho y GetAlto. Devuelven el ancho y alto, respectivamente, de un objeto Range.

Creación de objetos Range

- Range NewRangeFilasCompletas(Worksheet worksheet, int primeraFila, int ultimaFila). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya fila se encuentra entre primeraFila y ultimaFila (incluidos).
- Range NewRangeColumnasCompletas(Worksheet worksheet, int primeraColumna, int ultimaColumna). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya columna se encuentra entre primeraColumna y ultimaColumna (incluidos).
- public static Range NewRange(Worksheet worksheet, int primeraFila, int primeraColumna, int ultimaFila, int ultimaColumna). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya columna se encuentra entre primeraColumna y ultimaColumna (incluidos) y cuya fila se encuentra entre primeraFila y ultimaFila (incluidos).

Equals y operaciones de conjuntos

- bool AreEqual(Range range1, Range range2), bool AreEqual(List<Range> ranges1, Range range2), bool AreEqual(Range range1, List<Range> ranges2) y bool AreEqual(List<Range> ranges1, List<Range> ranges2). Devuelve true si los 2 conjuntos representan el mismo conjunto de celdas.
- List<Range> Union(List<Range> ranges1, List<Range> ranges2), List<Range> Union(List<Range> ranges1, Range range2), List<Range> Union(Range range1, List<Range> ranges2) y List<Range> Union(Range range1, Range range2). El resultado es una lista de objetos Range donde hay un objeto range por cada hoja, y el conjunto de objetos Range representa la unión de todos los objetos range pasados como parámetro.
- Range UnionMismoWorksheet(Range range1, Range range2). Realiza una union de 2 objetos range de la misma hoja. En caso de que no sean de la misma hoja, saltará una excepción.
- List<Range> Interseccion(List<Range> ranges1, Range range2), List<Range> Interseccion(Range range1, List<Range> ranges2), List<Range>

Interseccion(List<Range> ranges1, List<Range> ranges2) y Range Interseccion(Range range1, Range range2). Devuelve la intersección del primer conjunto con el segundo. La lista devuelta contiene un objeto range por cada hoja (o un único objeto range en el caso de la intersección de objetos Range simples).

- List<Range> Diferencia(List<Range> ranges1, Range range2), List<Range> Diferencia(Range range1, List<Range> ranges2), List<Range> Diferencia(List<Range> ranges1, List<Range> ranges2) y Range Diferencia(Range range1, Range range2). Realiza la operación de conjuntos diferencia, y devuelve el resultado. El resultado es conjunto1-conjunto2. La lista devuelta contiene un objeto range por cada hoja (excepto en el caso en el que los 2 conjuntos son objetos Range individuales, donde se devuelve un objeto Range en vez de una lista).

6.11. Simulador de ejemplo

Se incluye también un simulador de ejemplo en el proyecto. El simulador es un simulador muy simple con 5 variables. Se guarda un hashset con los clientes activos. Si se intenta hacer una monitorización pasando un id de un cliente que no está en la lista, salta una excepción.

En caso de que el estado sea corriendo, cada segundo el simulador da un paso, y por tanto nStep aumenta en 1. Además, en cada paso se modifica el valor de las variables VAR1, VAR2, VAR3 y VAR4 según la siguiente fórmula:

- long x = estadoSimulador.nStep + 1;
- var1 = Math.Pow(x%10, 2);
- var2 = x%100;
- var3 = Math.Log((x%10)+1)*50;
- var4 = ((x - 1 / 5) % 2) == 0 ? 0 : 10;

SimulatorServiceClient por defecto apunta a este simulador, pero no es difícil configurarlo para que apunte a un simulador distinto.

Este simulador puede ser útil, por ejemplo, para poder tener algo sobre lo que probar el programa cuando no se tiene el simulador, o el acceso a éste es muy complicado.

6.12. Detalles de implementación

La mayoría del código de la implementación final del proyecto no difiere mucho de los modelos, y no tiene mucho sentido comentarlo. Sin embargo, hay algunos detalles de la implementación que sí que merece la pena comentar:

- Para los temporizadores de SistemaCompDinamicos, dado que deben trabajar en la hebra principal (para evitar conflictos con Excel), se utiliza el timer System.Windows.Forms.Timer.
- Para el monitor de conexiones, dado que trabaja en paralelo, para evitar bloquear la hebra principal, se ha utilizado el timer System.Threading.Timer. Se ha escogido este tipo de timer porque trabaja en paralelo y es muy simple de manejar.
- Para la implementación del MonitorDeConexiones, para controlar los problemas del paralelismo con las variables compartidas, se utilizado un lock de c#, usando la directiva lock y un objeto de tipo Object, y para permitir que se espere en el método

GetEstadoConexion sin que sea una espera activa, se han utilizado condiciones, para esto se ha usado System.Threading.Monitor, con los método Wait (para bloquearse y esperar), y PulseAll (para despertar a los que estaban esperando en la condición).

- Se ha utilizado reflexión para crear los objetos cliente en la clase PropiedadesDeConexion. En particular, se ha utilizado Activator.CreateInstance, que permite crear un objeto a partir de un tipo y una lista de parámetros.
- Para la implementación de los tipos serializables (ListSerializable, etc…), se han convertido las listas en arrays de elementos (al serializar) y después de arrays de vuelta a listas (al deserializar). Dado que los arrays sí que son serializables, esto ha permitido que se puedan serializar los elementos.
- Para la implementación de RangeSerializable, se ha utilizado el método Address de Range para obtener un string representando el objeto al serializar, y después, para deserializar, se ha utilizado el método Evaluate, que es capaz de obtener un objeto Range en base a un string que represente rangos y uniones. Además, se ha utilizado el nombre de la hoja al serializar para poder obtener la hoja del range al deserializar.
- Para los demás tipos Excel serializables, generalmente se ha utilizado el nombre del objeto y de la hoja al serializar, lo cual ha permitido después cargar la hoja en base al nombre, y después, dentro de la hoja, cargar el objeto de la lista correspondiente según el nombre (los nombres son siempre únicos dentro de estos conjunto del mismo tipo).

7. Pruebas

Para el proyecto se ha desarrollado un conjunto bastante extenso de pruebas. En esta sección no se van a colocar las pruebas en sí, ya que ocupan bastantes páginas. Las pruebas en detalle se encuentran en el anexo. A continuación se comentará por encima los tipos de pruebas que se han hecho y como se han hecho.

7.1. Pruebas automáticas

Microsoft ofrece un conjunto de librerías para realizar pruebas. Con estas librerías, los métodos se marcan como método de prueba, y se ejecutan dentro de visual studio. Por desgracia, con VSTO esto no es posible, dado que es necesario que la aplicación esté corriendo para poder consultar las celdas, las gráficas, etc…

Esto ha hecho imposible utilizar el sistema de pruebas de Microsoft por completo. La estrategia que se ha seguido para solucionar este problema ha sido crear un proyecto igual al que se estaba desarrollando (en la versión final de las pruebas se ha utilizado la plantilla para crear este proyecto), y en la función de arranque de ThisWorkbook, se han lanzado las pruebas manualmente, permitiendo así poder ejecutar las pruebas y comprobar si fallaban o no.

Principalmente se han hecho pruebas unitarias. Para cada clase se ha desarrollado una clase de pruebas (excepto en algunos casos, que se han juntado las pruebas de más de una clase en un mismo archivo). Después, dentro de esta clase se ha creado un método EjecutaPruebas, y dentro se han ejecutado las pruebas de la clase. De este modo, desde ThisWorkbook era posible indicar qué clase de pruebas se quería ejecutar.

El resto es igual que siempre que se desarrollan pruebas. Se ha creado un método por prueba, y dentro de las pruebas se han usado los métodos de Assert de la clase de pruebas de Microsoft.

Para los nombres de los métodos se ha utilizado el siguiente sistema: NombreDelMetodo_ EstadoQueSeEstaProbando_ComportamientoEsperado.

7.2. Pruebas manuales

Algunas de las características del framework son gráficas. Cuando había que probar el comportamiento de un elemento gráfico, se han hecho pruebas manuales en vez de automáticas. Para esto, lo que se ha hecho es definir una secuencia de pasos a seguir (1.el tester pulsa el botón X, 2.el sistema muestra el panel X…), y después he seguido la secuencia de pasos y he comprobado que el sistema se comporta de la manera esperada.

8. Producto resultante del proyecto: entregables

Como resultado del desarrollo de este proyecto se han obtenido los siguientes productos finales:

- 2 templates para la creación de complementos Excel para simuladores. Estas plantillas permiten crear un proyecto que contiene los elementos del framework y las herramientas de éste. Una de las plantillas permite desarrollar para Excel 2010 y la otra para Excel 2013.
- Un template para crear simuladores que cumplen con la interfaz genérica definida en este proyecto. Por defecto, viene con una implementación de un simulador simple con 5 variables.
- La interfaz ISimulatorService, que debe ser usada por los simuladores a la hora de implementarse (o al crear un servicio intermediario) para que los elementos del framework y los controles de proyectos anteriores puedan funcionar correctamente con dicho simulador y para que los controles desarrollados para el simulador puedan ser usados para proyectos futuros. Esta interfaz, además de en la plantilla, viene incluida en el manual de uso.

9. Conclusiones

En este proyecto se ha desarrollado un framework para el desarrollo de complementos de Excel de nivel de documento para simuladores. Este framework se ha implementado usando VSTO para Visual Studio 2012 con Excel 2013 y se ha diseñado usando Entreprise Architect. Para poder usar el framework, se han creado 2 templates para visual studio, uno para crear aplicaciones con el framework con Excel 2010 y otro para Excel 2013.

Para asegurar la calidad y facilitar el desarrollo y el control de cambios, se ha utilizado la herramienta de control de versiones centralizada TFVC, usando como servidor los servidores gratuitos de visual studio online. Además, para facilitar el uso del framework se han añadido etiquetas a todos los métodos que son reconocidas por el intellisense mientras se programa, y se ha creado una documentación html con la herramienta doxygen.

El framework desarrollado permite crear proyectos que incluyen desde el principio los elementos de interfaz comunes a todas las aplicaciones de desarrollo de simuladores, tales como

los botones ribbon para el cambio de modo de uso o el panel para mostrar el estado de los simuladores.

Además, incluye también un conjunto de herramientas y un mecanismo estándar para el desarrollo de controles, permitiendo que estos sean reutilizados total o parcialmente para otros proyectos y/o simuladores y que sean desarrollados con mayor facilidad.

Finalmente, se ha desarrollado una interfaz genérica para simuladores que permite que los controles y aplicaciones que solo utilicen los métodos incluidos en la interfaz puedan interactuar con cualquier simulador existente y por existir que contenga los métodos de la interfaz genérica y con simuladores que no cumplan la interfaz pero para los que sea posible implementar un servicio intermediario que sí que la cumpla.

En definitiva, con este framework, los desarrolladores de complementos para simuladores podrán desarrollar los complementos en un menor tiempo. Además, serán capaces de desarrollar complementos que podrán reutilizarse con simuladores distintos al simulador para el que fueron diseñados, y podrán desarrollar, en menos tiempo, controles que serán perfectamente reutilizables en otros complementos y con otros simuladores.

Esta reutilización completa llevará a que progresivamente cueste cada vez menos tiempo desarrollar complementos para simuladores, ya que para la mayoría de tareas, con el tiempo se tendrá ya un control de otro proyecto que realice la tarea que se sedea implementar, haciendo que los equipos de desarrollo sean cada vez más eficientes y se gaste menos tiempo y dinero en el desarrollo de los complementos para simuladores.

Referencias bibliográficas

Programación con VSTO:
http://msdn.microsoft.com/en-us/library/d2tx7z6d.aspx

Representación de eventos c# en diagramas UML:
http://pic.dhe.ibm.com/infocenter/rsmhelp/v7r5m0/index.jsp?topic=%2Fcom.ibm.xtools.viz.dotnet.doc%2Ftopics%2Fr_cs_mapping.html

Página de Visual Studio Online
http://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx

Doxygen:
http://www.stack.nl/~dimitri/doxygen/

Enterprise architect:
http://www.sparxsystems.com/products/ea/

Visual studio:
http://www.visualstudio.com/

Anexos técnicos

1. Manual de uso del framework

1.1. Primeros pasos para la utilización del framework: templates y configuración

Para utilizar el framework, se proporcionan 2 templates de visual studio. Uno para Visual Studio 2012 para Excel 2010 y otro para Visual Studio 2012 y Excel 2013. Estos templates generan un proyecto que contiene la barra Ribbon, y actions pane desarrollados en el proyecto y demás elementos visuales comunes en las aplicaciones para simuladores (cambio de modo, etc…), y además contienen todas las herramientas para el desarrollo de aplicaciones Excel para simuladores que se explicarán más adelante en este manual de uso.

Para poder crear proyectos basados en estas plantillas, se deben copiar los archivos de plantilla y se deben pegar en la carpeta "…\Documents\Visual Studio 2012\Templates\ ProjectTemplates\Visual C#", que se encuentra en C:\Users\NombreDelUsuario. Donde NombreDeUsuario depende del nombre de usuario de Windows. Es decir, en la carpeta de templates propios de visual studio.

Una vez se pega en esa carpeta, se pueden crear proyecto en base a estas plantillas. Generalmente, las plantillas de esta carpeta se encontrarán abajo del todo en la lista de templates a la hora de crear un nuevo proyecto.

También se ofrece un template adicional llamado Simulador conforme a ISimulatorService, que permite crear un simulador conforme a la interfaz genérica de simuladores que se explicará más adelante. Por defecto, viene con una implementación sencilla de ejemplo. En caso de estar comenzando o simplemente querer hacer pruebas, puede ser buena idea utilizar directamente la implementación inicial que viene en esta plantilla, y después una vez se tenga acceso al simulador, cambiar el simulador al que apunta SimulatorService por el simulador real.

Por último, para crear proyectos office 2013, es necesario tener instalado el paquete de idioma de español en Visual Studio 2012, y también es necesario instalar el paquete para desarrollo de proyectos Office 2013. Este último puede descargarse en:

http://aka.ms/OfficeDevToolsForVS2012

1.2. Almacenamiento de datos persistentes

Para implementar casi todas las características de Excel, es necesario poder almacenar datos que se mantengan tras cerrar el documento. Por ejemplo, si queremos que al abrirse el documento Excel se sigan monitorizando las mismas variables que se estaban monitorizando al cerrar el documento, estos datos de monitorización deben almacenarse de manera que sean persistentes.

Las diferentes clases de la API consiguen esto de la manera que se explicará a continuación. Así es como se espera que el usuario de la API lo haga también, ya que al mantener una forma estándar de marcado de datos persistentes será más fácil reutilizar elementos de proyectos anteriores, al no tener que investigar su comportamiento (se sabe como funciona porque sigue el esquema estándar).

Para el almacenamiento de datos que sobrevivan al cierre y apertura del documento excel, se utilizará el atributo Cached. Lo que se hace es que se coge la instancia que queremos que sea

persistente, y se incluye una propiedad que envuelve a esta instancia en un libro u hoja (Cached solo funciona en clases hoja o libro, como por ejemplo, ThisWorkbook). Después, esta propiedad se marca con el atributo Cached. El marcarlo con este atributo es realmente lo que le dice a Excel que el contenido de esa propiedad quieres que se guarde al cerrar el documento y se cargue de nuevo al abrirlo.

Ejemplo:

```
public partial class ThisWorkbook{

    [Cached]
    public UnaClase UnaPropiedad { get {...} set {...} }

}
```

En el ejemplo, UnaPropiedad envuelve a una instancia de UnaClase, y está marcada con Cached. El get de UnaPropiedad se ejecutará al guardar los datos para el cierre del documento, y el set se ejecutará al abrir el documento de nuevo, dejando el objeto que envuelve UnaPropiedad igual que como estaba antes del cierre.

1.2.1. Campos almacenados

En el ejemplo, la propiedad UnaPropiedad es del tipo UnaClase. En UnaClase, habrán campos que queramos que persistan, y campos que queramos que se reinicien cada vez que se abra el documento Excel. Por ejemplo, imaginemos que tenemos un campo que representa un conjunto de datos copiados de manera temporal esperando a que un usuario haga click en pegar en alguna parte. Este conjunto de datos no queremos que sigan ahí al cerrar el documento.

Las reglas para definir propiedades o campos y si son persistentes son las siguientes:

- Para hacer que una propiedad o campo publica sea persistente, basta con ponerla publica y no ponerle el atributo XmlIgnore.
- Para hacer que una propiedad o campo publica no sea persistente, y por tanto se reinicie su valor cada vez que se abre el documento excel, se debe poner la propiedad pública y poner el atributo XmlIgnore. XmlIgnore sirve para indicar al sistema de Excel que este campo no debe almacenarlo de manera persistente.
- Para hacer que una propiedad o campo privada sea persistente, el tema no es tan simple. Excel coge solo los campos que son publicos que no tienen el atributo XmlIgnore y los privados y demás los ignora completamente. Por eso, lo que se debe hacer es ponerlo como publico, y para indicar al programado que ese campo no debe ser modificado, lo que se hace es ponerle un nombre que empiece con '_'. Al empezar con _, a pesar que es público, el programador sabrá que no debe tocar ese campo.
- Para hacer que una propiedad o campo privada no sea persistente, basta con poner esa propiedad como privada, ya que Excel ignora todos los campos y propiedades privadas de una clase.

En el apartado siguiente (clases estáticas) se ofrecerá un ejemplo con todos los tipos de propiedades.

1.2.2. Clases estáticas

Solo se pueden marcar instancias como persistentes, por lo que no es posible marcar directamente que una serie de campos estáticos son persistentes.

Para solucionar esto, se debe utilizar el patrón singleton, de manera que los campos en vez de ser estáticos, serían campos normales, y se accederían de manera global a través de la instancia única a la que se puede acceder a través de la clase (es un campo estático). Con la misma idea de antes de seguir una forma estandar en el marcado de datos persistentes, siempre que una clase tenga campos que deban ser accedidos de manera global, se implementará exactamente el patrón Singleton que se pone a continuación. Se debe respetar la estructura y los nombres, para que estos elementos sean más reutilizables.

Estructura para el patrón Singleton:

```
public class MiClase{
        private static MiClase _instance = new MiClase();
          public static MiClase Instance {get{return _instance;} set{_instance = value;}}

        //Constructor vacío requerido. Es privado para evitar otras instancias.
          private MiClase(){}

        //Propiedades persistentes
        public Tipo1 _Var1{ get; set; }
        public Tipo2 Var2{ get; set; }

        //Propiedades no persistentes
        private Tipo3 Var3{ get; set; }
         [XmlIgnore]
        public Tipo4 Var4{ get; set; }
}
```

Después, en la clase ThisWorkbook o en alguna clase que representa a una hoja, se colocaría el siguiente código:

```
[Cached]
public MiClase DatosMiClase {
      get{return MiClase.Instance;}
      set{MiClase.Instance = value;}
}
```

Cosas que se pueden destacar del patrón:

- La instancia única de la clase se llama Instance, este nombre es bastante estándar, y es aplicable a cualquier clase sin crear confusión.
- Los campos persistentes públicos se dejan sin atributo XmlIgnore y los “privados” se ponen con _ en el nombre y se ponen como públicos.
- Los campos no persistentes públicos se dejan con el atributo XmlIgnore, y los privados simplemente se dejan privados.

1.2.3. Clases con campos estáticos y no estáticos

En caso de tener una clase que tiene campos que queremos que sean accesibles de manera global y campos que queremos que pertenezcan a cada instancia, dado que el patrón singleton impide que podamos hacer esto, lo que se debe hacer en estos casos es dividir la clase en 2 clases, una que contiene la parte accesible de manera global (que implementa el patrón Singleton) y una que contiene la parte no global.

En caso de que una de las clases deba acceder a un campo de la otra que no se quiere que se acceda desde fuera, simplemente se marcará como pública y se le pondrá un nombre que empiece por '_', de manera que solo la clase esta accederá al campo de la otra, pero los programadores de otros elementos del sistema nunca accederan a esta propiedad debido a su nombre, consiguiendo que solo sea accedida por estas 2 clases.

1.2.4. Uso de otros sistemas: Custom Xml Parts

Se debe usar Cached como método de persistencia siempre que sea posible, ya que si se usase otro sistema, cuando se quisiese reutilizar los elementos unos meses más tarde, nadie se acordaría exactamente como se hacía para hacerlo persistente, ya que no seguiría la forma estandar.

Sin embargo, algunos componentes puede que usen cantidades enormes de memoria. Esto no suele ocurrir, pero por ejemplo si se tuviese un sistema que permite dibujar, y que tuviese que guardar todos los puntos de la imagen de manera persistente, es posible que superase los 100 MB (aunque incluso en esta situación es improbable). Por lo general no hay que preocuparse de esto, pero para los extraños casos donde se diese esto, habría que utiliar Custom Xml Parts en vez de Cached. El sistema no fuerza a que uses Cached, por lo que en caso de ser necesario, se podría usar Custom Xml Parts.

Cached tiene un limite de 10MB por objeto, y 100MB en total. Es evidente que este límite es enorme para los datos persistentes, y no es normal que se supere, ya que solo estamos almacenando instancias muy simples con un pequeño grupo de campos.

1.2.5. Problemas: clases no serializables

Cuando una clase tiene una propiedad que representa una instancia de un tipo no serializable, la propiedad falla al serializarse, y la clase devuelve null (pero no salta una excepcion), haciendo a la clase que contiene la propiedad no serializable. Por eso, hay que tener mucho cuidado con los tipos que se usan, y comprobar que realmente son aceptados por Cached.

Para algunos de estos tipos, se incluye una clase envoltorio en la API. Al usar estas clases en vez de las orignales, ya no se produce el problema. Las clases para las que se incluye envoltorio en la API son:

- Tuple. Es muy útil para guardar conjuntos de diferentes tipos. Para poder seguir usando Tuple, se define una clase TupleSerializable. Esta clase viene implementada para tuplas de tamaño 2,3 y 4.
- List, y Hashset. Se definen las clases ListSerializable y HashSetSerializable para poder guardar las listas y hashsets.
- Dictionary. Se define la clase DictionarySerializable para poder guardar diccionarios.

En el caso de los HashSet y los Diccionarios, debido a que es imposible serializar una interfaz utilizando la tecnología de Cached (al igual que con Custom Xml Parts), se ofrecen 2 versiones, una que ignora completamente el Comparer, y otra que recibe explícitamente el tipo del Comparer utilizado (haciendo así posible la serialización).

Si se encuentran tipos adicionales que no se puedan marcar como persistentes, habrá que crear un tipo propio que lo envuelva. Se puede mirar la implementación de algunos de los implementados en la API para coger una idea de cómo hacerlo.

Además de estos tipos, tampoco son serializables la mayoría de tipos de VSTO (Range, Worksheet, etc…). La serialización de estos tipos se discute en la sección de controles persistentes.

1.3. Cambio de modo

1.3.1. Manejo del cambio de modo en la API

Para indicar si se requiere contraseña para el cambio de modo a modo edición y cuál es la contraseña existen 2 mecanismos posibles. Primero, se puede modificar el archivo ParametrosGenerales.resx, el cual contiene 2 campos relacionados con la contraseña. La segunda forma es a través de las propiedades RequierePasswordEdicion y PasswordEdicion de SimuGlobals.

Para ver el modo actual, se puede consultar la propiedad ModoActual de SimuGlobals.

SimuGlobals implementa el patrón Singleton, y se puede acceder a estas propiedades a través de SimuGlobals.Instance.

1.3.2. Panel Ribbon de cambio de modo

Se ofrece un botón para el cambio de modo en la barra Ribbon del proyecto base. En el cambio de modo a modo edición, se ofrece una ventana Windows Forms que permite introducir la contraseña para cambio de modo.

1.4. Copiar, pegar, cortar o eliminar controles basados en celdas

Es común en las aplicaciones Excel con simuladores tener controles que trabajan sobre celdas. Por ejemplo, un sistema de monitorización que vaya escribiendo en una celda el valor de una variable de un simulador.

Sobre estas celdas, lo normal es que el usuario del documento Excel pueda copiar las celdas y el comportamiento asociado a estas, así como poder cortar, eliminar y pegar después este comportamiento. Por ejemplo, para la monitorización, lo normal sería que el usuario pudiese copiar (o cortar) la monitorización asociada a una celda y colocar esa monitorización en otra celda, o simplemente eliminar la monitorización de la celda.

Estas tareas de copiado, cortado (que se implementa como una copia seguida inmediatamente por una eliminación), eliminación y pegado vienen implementadas en la API. La API ofrece un panel contextual con las 4 opciones (solo visibles en modo edición), y el usuario de la API es el que debe encargarse de hacer estas operaciones para sus controles particulares.

Para ello, se debe utilizar los métodos estáticos de la clase SistemaOpCeldas. Lo primero que hay que se debe hacer es suscribir un controlador a un serie de celdas, que son las celdas sobre las que participa nuestro control. Para eso, se utilizan los métodos AddControladorOpCeldas,

SetCeldasAsociadas, GetCeldasAsociadas y RemoveControladorOpCeldas, que permiten asociar un controlador a un conjunto de celdas (de manera que si se hace una copia, cortado o eliminación el controlador será avisado) y permiten obtener las celdas asociadas a un controlador y desuscribir un controlador por completo.

En estos métodos, se pasa un objeto que implementa la interfaz IControladorOpCelda. Cuando se produzca alguna de las operaciones sobre una de las celdas a las que está asociado el objeto, se ejecutarán los métodos CopiaComportamiento y DestruyeComportamiento. Este objeto IControladorOpCelda pertenecerá a una clase que deberá haberse creado para realizar el copiado y eliminación de cada control particular. Por ejemplo, para la monitorización se podría tener una clase ControladorOpEspecialesMonitor, que realizaría el copiado y eliminación para las celdas monitorizadas.

Finalmente, CopiaComportamiento devuelve un objeto de tipo IDatosComportamientoCeldas, que es utilizado para pegar el comportamiento del control en una celda determinada. Se debe crear una clase propia que extienda IDatosComportamientoCeldas, y que implemente el método PegaComportamiento (celda), en el cual se deberá reconstruir el comportamiento de la celda copiada en una nueva celda (y devolver un objeto de este tipo nuevo en el método CopiaComprotamiento). Esta clase deberá contener todos los datos necesarios para esta reconstrucción, y debe implementarse de manera que sea posible eliminar el comportamiento de las celdas originales y seguir pudiendo pegar.

En caso de que un control tenga asociadas etiquetas de texto, puede ser buena idea gestionar el copiado de estas celdas también, porque al hacer copiar celdas, el sistema no realiza ningún copiado de manera automática, por lo que las celdas no se copiarían (y debería el usuario copiarlas a mano). Aunque también es posible hacer que no se copien y listo, y que el usuario se encargue de copiarlas después con el copiado tradicional.

En la operación de Copia, en caso de que no se pueda copiar (o no se quiera), el método deberá devolver null. Las celdas asociadas vienen en forma de listas de objetos Range, esto es porque se puede estar suscrito a celdas de distintas hojas.

1.4.1. Caso especial: celdas que no se puede copiar individualmente

En la operación de copia de IControladorOpCelda, se recibe como parámetro la celda a copiar y el conjunto de celdas seleccionadas. La razón por la que se ofrece esto es porque existen situaciones en las que es imposible copiar solo una celda. Por ejemplo, podría haber un control que opera sobre 4 celdas que siempre tienen que estar una al lado de la otra. En esta situación, no se puede simplemente copiar una de las celdas.

Cuando esto ocurra, lo que se puede hacer es realizar la intersección entre el conjunto de celdas seleccionadas y el conjunto de celdas que trabajan juntas. Entonces, el método de copia será llamado para las X celdas seleccionadas que estén entre estas 4, pero se podría solo copiar, cuando la celda a copiar fuese la de la esquina superior izquierda de la intersección (y devolver null para las demás). Esto se podría hacer por ejemplo con la operación de obtener subconjuntos de ExcelTools (viene más adelante en el manual de uso) y obteniendo para cada subconjunto el [1,1] y viendo el que tiene la menor posición fila columna. El objeto IDatosComportamientoCelda devuelto estaría diseñado para poder pegar el comportamiento completo (y tendría como información en que celda relativa a la primera del conjunto se copió). Después, al pegar pegaría el comportamiento de las 4 celdas.

En el caso del eliminado, se haría algo similar, además de solo hacer algo en la de la esquina superior izquierda, se devolvería false en el parámetro de salida desasociaCelda para todas las eliminaciones, y en la que realmente se haría la eliminación, se llamaría a SetCeldasAsociadas

y se desasociarían las 4 celdas (o en caso de ser todas las controladas por el controlador, se llamaría a RemoveControladorOpCeldas). Hay que tener en cuenta que aunque se haga la eliminación en la primera llamada a DestruyeComportamiento y se desasocien las celdas, se seguirán haciendo todas las llamadas para las demás celdas. Esto es porque durante una eliminación, se realiza una copia del estado de los controladores antes de comenzar el proceso, por lo que aunque los cambios son tenidos en cuenta por las operaciones posteriores, no afectarán a la operación de eliminado que se esté ejecutando en el momento. Una buena forma de tratar este problema es comprobar que la celda esté contenida en GetCeldasAsociadas mediante la operación de Interseccion de ExcelTools.

1.5. Nombre de la aplicación

En el actions pane se muestra el nombre de la aplicación arriba del todo. Para cambiar el nombre que sale arriba, se debe cambiar el campo NombreAplicacion del archivo ParametrosGenerales.resx. Es un archivo de recursos normal y corriente, por lo que desde el editor de visual studio se puede cambiar su valor.

1.6. Controles persistentes

En este framework se denominan controles a los sistemas independientes que realizan cambios en el documento Excel. Por ejemplo, un sistema de monitorización sería un control Excel, al igual que un sistema que permitiese añadir gráficas.

Estas funcionalidades siempre se proporcionan de manera bastante independiente, es decir, que podemos eliminar un sistema, y los demás seguirán funcionando sin problemas (en caso de que ambos sistemas compartan una clase, obviamente esta clase no la podemos eliminar del proyecto).

Los elementos que comúnmente se encuentran en estos controles son:

- Un botón Ribbon (o en el actions pane o alguna otra parte del documento), que al ser pulsado acciona un código.
- Un código que se ejecuta como consecuencia de pulsar el botón, que crea ventanas WindowsForms, y suele añadir algún comportamiento al documento (una gráfica, una celda que monitoriza una variable, etc…).
- Un gestor de elementos. Este gestor lo que permite es que un conjunto de clases de un tipo puedan persistir al cierre del documento.
- Objetos individuales que son los que realmente realizan la acción y que son gestionados por el gestor.
- Componentes individuales. Estos componentes generalmente se ejecutan desde la primera vez que se abre el documento y realizan una serie de tareas sobre el documento a lo largo del programa. Por ejemplo, en vez de tener un gestor y un montón de elementos que escriben el valor de una variable a una casilla, podría haber un componente que hiciese todo el trabajo.

Son necesarios los gestores porque un elemento no puede simplemente persistir, sino que para que sea posible que persista, tiene que estar en la clase ThisWorkbook con el atributo Cached o pertenecer a un objeto que está contenido en el ThisWorkbook con la etiqueta Cached. Esto es lo que hace el gestor, es el gestor el que tiene la etiqueta, y todos los elementos simplemente están contenidos en el gestor, y como consecuencia persisten.

Para facilitar la reutilización y reducir el coste de desarrollo, se proporcionan en este framework un conjunto de librerías y una forma estándar de crear controles. A continuación se menciona la forma estándar en la que se deben definir los controles y cómo usar las diferentes herramientas que reducien el coste de desarrollo.

1.6.1. Forma estándar de definir los controles

Para facilitar la reutilización de los controles, a continuación se describen las reglas de nomenclatura y de uso que deberían seguirse.

1.6.1.1. Botón Ribbon

En los controles suele haber un botón en la barra Ribbon (también puede estar en cualquier otra parte, como por ejemplo en el actions pane, pero por simplificar el manual, se asumirá que es un botón en la barra Ribbon). Este botón generalmente simplemente es un botón de tamaño grande que tiene una determinada imagen. Por lo tanto, lo único que diferencia un botón de otro es generalmente la imagen que se muestra para el botón, que generalmente, suele ser una de las imágenes predefinidas de office.

Si se quiere reutilizar el control, realmente se puede hacer poniendo la interfaz que uno quiera. Sin embargo, si se quiere que al reutilizar el control no haga falta gastar un tiempo buscando que imagen ponerle, una buena idea es guardar junto a los objetos almacenados para ser reutilizados, un documento que contenga una lista de las imágenes recomendadas para los botones de cada control.

1.6.1.2. Acción al pulsar el botón

Para hacer que sea más fácil reutilizar la acción que se produce al pulsar el botón, se recomienda encapsular el código en una clase con un único método estático Run en el que se ejecute el código consecuencia de pulsar el botón. Para que sea más fácil de identificar, se recomienda que esta clase tenga como nombre Operacion[Nombre], por ejemplo, OperacionMonitorizar. Entonces, si alguien quiere reutilizar el código del click del botón, lo único que tiene que hacer es en el código del click del botón poner OperacionMonitorizar.Run().

En este código se suelen crear ventanas Windows Forms. Una vez más, para facilitar la reutilización, se recomienda que en caso de haber opciones aceptar y cancelar, se modifique la propiedad DialogResult de estos botones para que el de aceptar tenga como valor OK y el de cancelar tenga como valor Cancel. Después, en el código del botón en sí (es decir, en el método run), se debería lanzar la ventana con ShowDialog, y comprobar si el usuario acepto. Si el resultado es DialogResult.OK, entonces es que acepto. Estas ventanas lo normal es que se usen para recoger datos. En caso de que este sea su propósito, se deberían construir de manera que recoger y devolver los datos sea lo único que hagan. Es decir, no es recomendable que la acción en sí que se realiza en base a esos datos se haga en la ventana, dado que esto haría imposible reutilizar la ventana en sí para otros controles.

Los datos recogidos se pueden ofrecer como propiedades públicas en la ventana que permitan obtener estos valores (en cuyo caso, los nombres de las propiedades deben estar muy bien elegidos). Otra opción es pasar un objeto en el constructor y que dentro de este objeto se rellenen estos datos recogidos.

Siguiendo estas reglas simples, será posible reutilizar ventanas de un control a otro (por ejemplo, cambiando algún campo) y será muy fácil reutilizar un control de un proyecto a otro, simplemente hay que realizar las tareas necesarias para los componentes y gestores (explicadas

más adelante), y en la función de click del botón que se quiera que ejecute la acción principal del control, llamar al método Run de la clase Operacion del control.

1.6.1.3. Gestión de conexiones

Para garantizar la reutilización, es necesario gestionar las conexiones de un modo particular haciendo un uso adecuado de las herramientas de conexiones ofrecidas en la API. Esta parte no se discutirá aquí, sino que se tratará en la sección de manejo de conexiones con simuladores.

1.6.1.4. Componentes y gestores

Como se ha comentado antes, los controles suelen estar compuestos por uno o más componentes o gestores principales, que suele tener subactividades en el caso de los componentes, como un componente de monitorización que tiene un conjunto de monitorizaciones de variables en casillas, y suelen tener elementos que representan subactividades y que son gestionados en el caso de los gestores.

Gestores y componentes deben ser inicializados y marcados como persistentes. Por eso, ambos elementos deben implementar IInitializable, y deben implementar el patrón singleton. De esta manera hay una forma estándar de reutilizar componentes y gestores. Siempre que haga falta reutilizar un componente o gestor, se debe envolver su propiedad Instance en una propiedad de ThisWorkbook con el atributo cached (como se indica en la sección de persistencia) y debe llamarse a la función initialize en la función de arranque del documento y a la función Terminate en la función de cierre del documento.

Además, para que sean fácilmente identificables es recomendable llamar a los componentes Componente[Nombre], y a los gestores Gestor[Nombre]. Por ejemplo, ComponenteMonitorizacion.

Siempre que hayan objetos individuales que realicen las tareas, se debe crear un gestor, y siempre que todas las tareas las haga un único objeto, se debe crear un componente (que no viene a ser más que una clase que implementa IInitializable y el patrón singleton mencionado en esta guía).

1.6.1.5. Resumen de pasos para reutilización

Para reutilizar un control, se debe copiar todas las clases y pegarlas en el proyecto en el que queremos añadir el control. Buscar todas las clases que sean componentes y gestores, y marcarlas como persistentes (envolviendo su propiedad Instance en una propiedad marcada con Cached en la clase ThisWorkbook) y llamar a Initialize en la función de arranque en ThisWorkbook y a Terminate en la función de cierre de ThisWorkbook.

Finalmente, se debe elegir donde se quiere que esté el botón principal del control, y en el código de click del botón, llamar al método Run de la clase de operación del control (que será fácil de saber cuál es porque su nombre empezará por Operacion).

1.6.2. Desarrollo de controles

A continuación se explica cómo utilizar las diferentes herramientas de la API para el desarrollo de controles y como desarrollar controles en general. El uso correcto de estas herramientas reduce el tiempo de desarrollo de los controles y hace que estos sean reutilizables.

1.6.2.1. Inicialización, terminación y persistencia

Al utilizarse Cached (o cualquier otro sistema de persistencia de datos de Excel), a la hora de cargar los datos, el constructor vacío se ejecuta y después se carga el valor de cada una de las propiedades y campos. Eso significa que al ejecutarse el constructor no está cargado el valor de las propiedades. A veces es necesario ejecutar código después de que se carguen las propiedades. También a veces es necesario realizar alguna actividad al cerrarse el documento Excel (como llamar a la operación de cerrar cliente del simulador).

Siempre que se necesite realizar alguna actividad de inicialización y terminación, se debe implementar la interfaz IInitializable, que incluye 2 métodos, Initialize y Terminate(). Cuando se implemente un gestor, componente o elemento gestionado, se deberá implementar esta interfaz.

Para los elementos gestionados la persistencia, inicialización y finalización es manejada por el gestor y para los demás, se debe implementar el patrón Singleton que viene en la sección de Almacenamiento de datos persistentes. Después, para hacer los datos persistentes, se debe envolver el objeto Instance de la clase de la manera que se explica en la sección de Almacenamiento de datos persistentes. Además, se deberá llamar al método Initialize en la función de arranque y a Terminate en la de finalización.

1.6.2.2. Timers

Se proporciona un mecanismo de timers propio que trabaja sobre la misma hebra que el documento Excel. Este sistema es mucho más simple que los otros proporcionados por las APIs de Microsoft, y es recomendable su uso siempre que se quiera realizar una actividad periódica. La modificación del documento Excel siempre se recomienda que se haga en la hebra principal, por eso este sistema de temporizadores trabaja en la hebra principal. Además, no hay que preocuparse de paralelismo, todas las llamadas se producen en la misma hebra y sin interrupciones.

Los temporizadores se manejan con la clase SistemaCompDinamicos. Para añadir una actividad periódica, se llama al método AddTareaPeriodica de esta clase. Como parámetros se pasa el método que se quiere que se ejecute periódicamente, el periodo, el objeto de estado que se debe pasar al método. Como resultado se recibe en el último parámetro el id del temporizador (un long).

Para parar la ejecución del método, se llama al método RemoveTareaPeriodica, y para comprobar si el timer asociado a un determinado id está corriendo, se utiliza el método IsTareaPeriodica.

Como se puede ver, es extremadamente simple de utilizar, solo hay que crear un método que acepta un objeto de tipo Object como parámetro que es el que queremos que se ejecute periódicamente, y llamar a AddTareaPeriodica y a partir de ese momento el método será llamado periódicamente. El primer id que da la clase SistemaCompDinamicos es el 1. El 0 nunca se asigna a ninguna tarea.

1.6.2.3. Manejo de tipos Excel

Los tipos Excel: Worksheet, Range, Chart no son persistentes. Es decir, si queremos apuntar a una hoja y que un objeto que hemos marcado como persistente siga apuntando a la misma hoja tras el cierre y apertura del documento, no bastaría con guardar una propiedad de tipo Worksheet que apuntase a dicha hoja, ya que al no ser serializable, al abrir el documento el campo valdría null.

Para reparar este problema, se proporcionan las clases RangeSerializable, WorksheetSerializable, GraficoFlotanteSerializable y GraficoHojaSerializable, que envuelven a objetos Range, Worksheet y a graficas incrustadas en hojas y hojas de gráficos, respectivamente. Si se intenta marcar un objeto de alguno de estos tipos como persistente, tras el cierre y apertura del documento, seguirán apuntando al mismo objeto. El RangeSerializable contendrá un objeto Range que apuntara al mismo conjunto en la misma hoja, el objeto WorksheetSerializable seguiría conteniendo a la misma hoja, etc…

Con estos objetos, uno puede mantener referencias a objetos de estos tipos sin tener que preocuparse de perder la referencia tras el cierre del documento. Estos objetos, además de envolver a los otros y hacer que persistan, solucionan unos cuantos problemas que suelen presentarse en la programación Excel.

Primero, además de aceptar objetos nativos como parámetro, también aceptan los objetos host ítem que extienden su funcionalidad. De manera que es posible, por ejemplo, pasar un objeto Microsoft.Office.tools.Excel.WorksheetBase e internamente se extrae el objeto nativo asociado.

Para los castings en la otra dirección no se proporciona nada, ya que para obtener el host ítem a partir del nativo basta con hacer Globals.Factory.GetVstoObject(objetoNativo).

Otro problema que se soluciona es el de la comparación de objetos Range. Range.Equals realiza una comparación por referencia, por lo que es imposible comparar que 2 objetos range contienen las mismas celdas. Para permitir, por ejemplo, guardar diccionarios de objetos Range (que utilicen las celdas de los Range como claves), cosa que es muy común, se proporciona la clase MismasCeldasComparer. Esta clase debe ser usada como comparer en los diccionarios y hashsets para que 2 objetos RangeSerializable se consideren iguales solo si representan el mismo conjunto de celdas.

Finalmente, el último problema que se soluciona es el de la eliminación de las hojas. Cuando una hoja se elimina, los objetos Range dejan de ser válidos, e intentar acceder a alguna de sus propiedades lanza una excepción. RangeSerializable establece el valor de campos WorksheetRef en el mismo momento que se asigna el objeto Range, haciendo así posible comparar si un objeto range pertenece a una hoja aunque la hoja haya sido eliminada. En caso de diccionarios o hashsets que utilicen MismasCeldasComparer, la eliminación de los objetos una vez se sabe que son inútiles puede ser un poco complicada (ya que buscar el objeto hará que salte una excepción), por eso a continuación se pone un ejemplo. La clave está en iterar en el diccionario en vez de hacer búsquedas.

A continuación se pone un ejemplo de código de cómo debe hacerse en el evento de cierre para eliminar los objetos range de una lista que pertenecen a las hojas eliminadas (en este caso se eliminan de un diccionario llamado Monitorizaciones). Para ello se utiliza LINQ. Si el lector de este manual no sabe usar LINQ para c#, ya viene siendo hora de que aprenda, porque LINQ simplifica enormemente la programación de cualquier cosa en c#, y viene siempre precargado en las clases de c#:

```
private void Instance_SheetClosed(object sender, SheetClosedEventArgs e)
{
   HashSet<Excel.Worksheet> hojasCerradas = e.HojasCerradas;
   List<RangeSerializable> listaAEliminar
            = new List<RangeSerializable>(Monitorizaciones.Keys.Where(x => hojasCerradas.Contains(x.WorksheetRef)));
   Monitorizaciones
            = Monitorizaciones.Where(x => !listaAEliminar.Contains(x.Key)).ToDictionary(x => x.Key, x => x.Value);
}
```

1.6.2.4. Acceso a ThisWorkbook y Application

Para acceder al objeto único que representa la hoja y al objeto Application se recomienda utilizar las propiedades WorkbookExcel y Application de SimuGlobals. Acceder por estas propiedades reduce las dependencias y repara algunos problemas de acceso a estos objetos.

1.6.2.5. Gestores de elementos

Como se ha explicado antes, siempre que se quiera tener un conjunto de elementos individuales que persistan al cierre de la aplicación y para los que se quiera tener un método que permita crearlos (y a lo mejor también eliminarlos), se debe crear un gestor que gestione estos elementos.

Para esta tarea, se proporciona la clase GestorDeElementos<T>. Se debe crear un gestor propio, y este gestor debe tener una propiedad de tipo GestorDeElementos, que gestione los objetos creados por el gestor. El gestor debería implementar el patrón Singleton, debería implementar IInitializable, y en el initialize llamar al método Initialize del objeto GestorDeElementos, y lo mismo en el Terminate.

El gestor debería tener también un método de creación de elementos gestionados, por ejemplo, CreaGrafica, en este método se debería llamar al método AddElemento del objeto GestorDeElementos. Si se implementa un método de eliminación de elementos, en este debería llamarse a RemoveElemento. Y para iterar por los elementos está la propiedad Elementos de GestorDeElementos. Se recomienda que estos métodos no sean estáticos, sino que deban ser llamados a través de Instance.

1.6.2.6. Eventos especiales y cambio de modo

Es muy común tener que realizar tareas cuando se cambia de modo, o comprobar en qué modo se está. Para comprobar el modo, se utiliza la propiedad ModoActual de SimuGlobals. Para poder realizar una tarea cuando se produce el cambio de modo, se proporciona el evento ModoCambiado en SimuGlobals.

Cuando se cierra una hoja, los objetos asociados a la hoja ya no son válidos, y lanzan excepciones en el acceso. La solución es suscribirse al evento de cierre de hoja y eliminar los objetos cuando se produce dicho evento. Este evento no existe en Excel y por eso lo añade la API. El evento se llama SheetClosed, y se encuentra tanto en SimuGlobals como en ThisWorkbook.

Como el evento de cambio de modo es común usarlo en ThisWorkbook para ocultar y mostrar los botones del Ribbon, se proporcionan los métodos ConfiguraSegunModo y

SimuGlobals_ModoCambiado, llamados cada vez que se produce un cambio de modo (ConfiguraSegunModo también es llamado en el arranque de la aplicación). A continuación se pone un ejemplo de cómo se debe hacer para asegurar que los objetos se eliminan cuando ya no hacen falta:

```
public class UnaClase:IInitializable, IDisposable{
	public void Initialize(){
		SimuGlobals.Instance.ModoCambiado+= this.ModoCambiado;
		ConfiguraSegunModo(SimuGlobals.Instance.ModoActual);//Primera llamada
	}
	public void ConfiguraSegunModo(ModoUso nuevoModo){
		// Aquí se realizan los cambios según el modo en el que se esté
	}
	public void ModoCambiado(Object sender, ModoCambiadoEventArgs e){
		ConfiguraSegunModo(e.ModoDeUso);
	}
	public virtual void Dispose(){
		SimuGlobals.Instance.ModoCambiado-= this.ModoCambiado;
	}
}
```

1.7. Manejo de simuladores

1.7.1. Conexiones en componentes y gestores

A la hora de implementar un componente o gestor, dado que todo el desarrollo de Excel gira entorno a los simuladores, lo normal es que este gestor o componente haga uso de una conexión a un simulador y envíe una serie de mensajes a este.

Para que el componente sea realmente reutilizable, debería ser capaz de trabajar con cualquier configuración de conexión, siempre y cuando esta configuración utilice la misma interfaz para el binding que el cliente usado por el componente o gestor (esto es debido a limitaciones técnicas).

Para conseguir facilitar la especificación de esta conexión, se proporciona una clase llamada PropiedadesDeConexion, que contiene información del binding y del endpoint, que son los 2 elementos principales que definen la configuración de una conexión WCF. Además, la clase ofrece un método CreaClienteWCF<T> que crea un cliente WCF del tipo pasado como parámetro (por ejemplo SimulatorServiceClient) usando los datos de la clase PropiedadesDeConexion. Es recomendable crear los clientes usando este método, ya que así serán más flexibles.

A la hora de crear un gestor o componente, este debería presentar una propiedad llamada ConfiguSimulador, de tipo PropiedadesDeConexion. Después, el valor de esta configuración debería ser utilizado para construir el cliente. Por defecto esta propiedad debe tomar el valor null, lo cual debe interpretarse como que toma se debe usar el valor de SimuGlobals.Instace.ConfigBaseSimuladores. En principio esta configuración debería establecerse antes del método Initialize, pero puede ser buena idea hacer que se pueda cambiar la configuración de la conexión después de que la llamada a Initialize.

Es recomendable que los componentes y gestores usen la clase SimulatorServiceClient, que viene incluida en la API, ya que así los componentes será perfectamente reutilizables en cualquier aplicación. A la hora de reutilizar el componente o gestor, en caso de haber una conexión a simulador adicional y querer usar los datos de esta conexión, se deben extraer los

datos de configuración de este simulador, pero a la hora de construir el objeto PropiedadesDeConexion, en vez de usar la configuración de endpoint de este simulador, se debe pasar como la de SimulatorServiceClient (se puede ver el nombre en app.config). De este modo, si el simulador incluye los métodos de la interfaz genérica, el componente o gestor seguirá funcionando perfectamente con el nuevo simulador independientemente de donde se encuentre o cual sea su configuración para la conexión.

Aunque es poco habitual que haga falta, puede ser beneficioso hacer pública la propiedad de los gestores y componentes que representa el cliente WCF usado por éstos. En caso de querer modificar el cliente, lo más fácil es que solo se permita antes de initialize, y que en caso de hacerse, se ignore la propiedad ConfigSimulador (en el método initialize se comprobaría si el cliente es null antes de crear uno nuevo). Esto puede ser necesario, por ejemplo, para especificar el nombre y contraseña para hacer una conexión segura (aunque hay otras formas). Un nombre posible para esta propiedad es ClientePrincipal. Se debe poner XmlIgnore sobre este campo, ya que los clientes no son serializables y no deben marcarse como persistentes.

1.7.2. Consulta del estado: accesible o no accesible

Para consultar si un simulador es accesible o no accesible (si está conectado o no, por ejemplo) se usa la clase MonitorDeConexiones. Para hacer las consultas posibles de manera eficiente, se realizan consultas periódicas del estado de los simuladores. Así, cuando uno consulta el estado actual, lo que se recibe es realmente un booleano que estaba almacenado en MonitorDeConexiones, y por lo tanto, la consulta es inmediata.

El sistema internamente utiliza un conjunto de hebras en paralelo para realizar las consultas, por lo que la monitorización no produce ningún impacto en el rendimiento de la hebra principal del sistema (ni en las hebras desde las que se usan los métodos del MonitorDeConexiones) y es una manera muy eficiente de consultar el estado de los simuladores. Si el sistema aún está realizando la primera consulta, el que llama debe esperar (se puede usar HayDatosDeEstado para comprobar si se ha producido). Este pequeño tiempo de espera no producirá ningún efecto visible en la aplicación, dado que solo ocurrirá la primera vez.

El tiempo de espera utilizado en las consultas de estado de la conexión se encuentra en la propiedad Timeout. Si se quiere cambiar el timeout, basta con cambiar esta propiedad.

Los métodos aceptan para la conexión tanto strings como objetos ClientBase (clientes WCF). El sistema no almacena los clientes, lo que hace es extraer la URI y almacenarla. El sistema funciona con simuladores, pero también con cualquier otra clase de cliente WCF, ya que simplemente usa la URL para la consulta.

1.7.2.1. Configuración y consulta de que conexiones se monitorizan

Para configurar qué simuladores (u otra clase de servicio) se monitorizan, se deben usar los métodos AddMonitorizacionDeEstado (conexion, int? Timeout) y RemoveMonitorizacionDeEstado (conexión). El segundo parámetro en el método add indica el timeout para las consultas de estado y es opcional. En caso de ser null, se utiliza el valor de la propiedad Timeout de MonitorDeConexiones

AddMonitorizacionDeEstado se puede llamar múltiples veces sobre la misma conexión, y cada vez que es llamado se incrementa un contador de monitorización asociado a la conexion. Si una conexión ya estaba siendo monitorizada, y el timeout pasado es distinto del actual, se cambia el timeout para la conexión. El método Add incrementa el contador para la conexión y el método

Remove lo reduce, cuando es mayor que 0, se monitoriza la conexión, y cuando es igual a 0, no.

Esto se hace así para permitir que sistemas completamente distintos usen esta clase, generalmente el desarrollador no tiene que preocuparse de que existe un contador, solo tiene que asegurarse de hacer un remove por cada add (normalmente se hace un add al arranque de la aplicación y un remove en el cierre).

Se puede especificar el timeout por separado con el método SetTimeoutMonitorizacion (conexión, int? timeout). Y se puede obtener el timeout para una conexión con GetTimeoutMonitorizacion(conexion).

Para consultar si un servicio está siendo monitorizado, se debe usar el método EstaSiendoMonitorizada, que devuelve true si la conexión está siendo monitorizada.

1.7.2.2. Consulta del estado

Para la consulta del estado, la clase contiene los métodos:

- GetEstadoConexion(conexion, out bool esAccesible). Obtiene el último estado que se obtuvo de una conexión que está siendo monitorizada. En caso de no haber llegado a terminar ninguna consulta todavía, el sistema se para en la hebra actual hasta terminar la primera consulta. Devuelve en esAccesible si el simulador (u otro tipo de conexión) se puede acceder. Este método requiere que la conexión haya sido añadida a las conexiones monitorizadas.

- IntentaAccesoAConexion (conexion, out bool esAccesible) e IntentaAccesoAConexion (conexión, int timeout, out bool esAccesible). Este método no utiliza los datos de estado almacenados (y no requiere que la conexión haya sido añadida a las monitorizadas), sino que realiza la consulta de estado desde 0 y devuelve si se pudo o no acceder al simulador. Devuelve en esAccesible si el servicio esta accesible. Timeout indica el timeout para la consulta de estado.
- HayDatosDeEstado (conexion), que devuelve true si se ha realizado ya la primera consulta del estado del simulador, es decir, si hay datos de estado para la conexión.

1.7.2.3. Mensajes a los simuladores

Cuando se usa en una aplicación Excel un cliente de un simulador u otro tipo de servicio WCF, las llamadas al servicio se realizan continuamente. Si el simulador no está accesible, las consultas llevan unos cuantos segundos. Como consecuencia, la aplicación se va parando hasta el punto de no ser utilizable.

Esta clase soluciona este problema. Lo único que hay que hacer es añadir el simulador a los simuladores monitorizados, y llamar al método GetEstadoConexion antes de realizar cualquier llamada con el cliente. Si el simulador no es accesible (no hay conexión a internet, por ejemplo), se obtendrá false. Lo que hay que hacer es coger ese valor devuelto, y solo realizar la llamada al servicio si el valor es igual a true.

Aun así se debe realizar la llamada a los servicios entre try catch, dado que es posible que el simulador se desconecte justo después de realizar la consulta a GetEstadoConexion (o se haya desconectado después de la última consulta que hizo el sistema y aun no se ha dado cuenta el sistema de monitorización del cambio).

Si ocurre esto, y salta una excepción, el tiempo que la aplicación se pare dependerá, sobretodo, de la configuración del cliente WCF que realiza las llamadas al servicio y del timeout establecido en el sistema de consulta de estado. El que tenga un timeout mayor será el que determine cuanto tiempo se para. En un cliente WCF no modificado y sin cambiar el timeout, el tiempo suele ser de unos 5 segundos. Tras esos 5 segundos, la aplicación funcionará correctamente (si no se usase el monitor de estados, simplemente dejaría de ser utilizable a partir de ese momento).

1.7.3. Panel de estado de los simuladores

En el ActionsPane se muestra el estado de los simuladores registrados. Para estos, se muestra si está accesible o no y además estados adicionales.

Evidentemente, para qué simuladores se muestra el estado y el nombre de estos depende de cada aplicación, y por tanto deben especificarse cuales son los que se quiere mostrar, y, en caso de ser necesario, que estados adicionales tienen.

Para configurar el panel de estado de los simuladores, se utiliza el objeto contenido en la propiedad PanelInformacionDeEstado de ThisWorkbook o de SimuGlobals, que es de tipo IInformacionDeEstado.

Para añadir un simulador o servicio al panel, se utiliza el método AddServicio sobre este objeto. Este método permite especificar la posición en la que mostrar el simulador (ordenado de arriba abajo), un id (opcional) que puede ser usado para buscar el objeto con los demás métodos, un string representando el nombre a mostrar en el panel para este simulador o servicio, y la URL o el cliente para la conexión o servicio para el que se quiere mostrar el estado. Generalmente se recomienda pasar la URL. Para extraer la URL a partir de un cliente WCF se puede utilizar el método de ExcelTools que sirve justo para esto.

También se puede eliminar un servicio o simulador, mediante el método RemoveServicio o RemoveServicioById, ver si hay algún elemento con un determinado id, ContainsIdServicio, obtener el nombre asociado a un servicio, con GetNombre y GetNombreByID, obtener el servicio en sí en base a la posición el id, con GetServicio y GetServicioById, obtener el id para el servicio de una posición dada, mediante TryGetId, y finalmente ver el número de servicios en el panel, con NumServicios y ver o modificar el comparador usado para los ids, a través de la propiedad IdsEqualityComparer.

1.7.3.1. Especificación de estados

Con las operaciones antes mencionadas se consigue que el panel muestre para los simuladores y servicios si son accesibles o no. Sin embargo, pueden requerirse estados adicionales.

Para esto se proporciona la interfaz IControladorDeSubestados, que extiende de IInitializable y añade el método DeterminaSubestado (bool esAccesible, int posicion, object servicio, out String stringEstado, out Color color), que devuelve true si el controlador quiere que se reemplacen los datos de estado actuales (el nombre y color) por los datos de estado pasados en los parámetros stringEstado y color, y false si quiere que se mantenga como está. esAccesible indica si se está expandiendo el estado accesible (true) o el de no accesible (false). Los demás parámetros son simplemente datos del servicio que pueden ser útiles para determinar el subestado.

Si se quiere especificar estado adicionales se debe crear una clase que implemente esta interfaz y asociar el objeto a los simuladores o servicios. Existen 0 o 1 objetos controlador asociados a cada estado de cada conexión (uno para accesible y uno para no accesible), y 0 o 1 objetos

controlador generales para cada estado (uno para accesible y uno para no accesible). Para especificar los primeros, se usan los métodos SetControladorDeSubestados y SetControladorDeSubestadosById. Y para el general, se usa SetControladorDeSubestadosAll. En los 3 métodos se debe especificar si se quiere expandir el estado de accesible o el de no accesible.

Finalmente, se puede consultar los controladores específicos y generales con los métodos GetControladorDeSubestados, GetControladorDeSubestadosById y GetControladorDe SubestadosAll.

1.7.3.2. Subestados detenido y ejecutando

Según la interfaz genérica que se describirá más adelante en este documento, existen 2 estados principales de un simulador que está funcionando, detenido y ejecutando. Por tanto, los simuladores normalmente tendrán estos 2 estados.

Por eso, como es normal que tengan estos estados adicionales, se proporciona la clase CtrSubestadosSimulador, que especifica estos 2 estados para la configuración pasada como parámetro. Hay que tener en cuenta que internamente utiliza la clase SimulatorServiceClient, y que para usarlo hay que llamar al método SetControladorDeSubestados con un nuevo objeto de este tipo para cada simulador.

La clase utiliza las propiedades pasadas en el constructor para construir un nuevo cliente. En caso de que se requiera configuración adicional para dicho cliente WCF, se deberá crear una clase nueva que herede de CtrSubestadosSimulador, y estos cambios al cliente deberán hacerse en el constructor de la nueva clase (en el constructor, el cliente estará ya creado pero aún no se habrá enviado el mensaje de inicialización al simulador). Finalmente, se deberá crear una instancia de la clase nueva para pasárselo a SetControladorDeSubestados en vez de una instancia de CtrSubestadosSimulador.

1.7.3.3. Ejecución del panel

El panel solo muestra el estado cuando se está en modo ejecución (dado que es el modo en el que se ejecutan las cosas), en modo edición muestra para cada servicio el último estado que se detectó en gris.

1.7.4. Panel de introducción de datos de simulación

Es común poner en el arranque del documento Excel un panel en el que se recogen los datos de configuración de la conexión al simulador. Se incluye en el proyecto un panel con las opciones más comunes. La clase se llama PanelDatosDeConexion, y se encuentra en la capeta UIElements. Contiene un campo para elegir el simulador sobre el que se quiere trabajar, uno en el que especificar la dirección ip, y otro en el que especificar el puerto.

Este panel acepta en el constructor una lista de nombres de simuladores (para el campo de elegir simulador), y un valor por defecto para el ip y el puerto. Además, tiene una propiedad DatosDeConexion que permite obtener los datos que ha introducido el usuario. En caso de que el usuario haya pulsado aceptar, se devuelve DialogResult.OK.

En caso de que se quiera usar como es, basta con instanciarlo y hacer ShowDialog. En caso de que se quiera cambiar los campos que contiene, se recomienda realizar una copia del panel (con otro nombre) y modificar este nuevo panel, para no cargarse el original. El panel está programado de una manera muy simple, por lo que realizar cambios en éste y adaptarlo a las necesidades específicas de la aplicación es sencillo.

1.8. Interfaz genérica para simuladores

Los simuladores siempre funcionan como servicios que contienen un único simulador corriendo. Todas las clases que acceden al servicio comparten el mismo simulador.

Para poder realizar las diferentes operaciones con el simulador, los usuarios del simulador deben llamar a un método de inicialización, que les da un id de cliente. Después, una vez terminan, para liberar recursos, deben llamar un método del simulador que sirve para eliminar un cliente. Para eso, la interfaz contiene los métodos int Initialize(), y void FinishClient(int idClient).

Los simuladores arrancan en estado detenido, y pueden estar detenidos o corriendo. Además, los simuladores funcionan por pasos, donde en cada paso se avanza en la simulación, y tienen un tiempo de simulación asociado. Estos 3 valores se incluyen en la interfaz en la estructura State, que contiene campos para estas 3 cosas. Además, con el método GetState(int idClient) se obtiene el estado actual.

Los simuladores se suelen poder parar y arrancar. Para esto, se ofrecen los métodos void SendFreeze(int idClient) y void SendRun(int idClient).

Finalmente, los simuladores en su ejecución lo que hacen es modificar en cada paso un conjunto de variables, algunas variables, además de poder consultarse se pueden modificar. Para eso, se ofrecen los métodos void VbleUpdate(int idClient, string nvar, double vble), que permite modificar el valor de una variable, double VbleQuery(int idClient, string vble), que permite obtener el valor de una variable y los métodos List<double> MonitorizeVbles(int idClient, List<string> variables) y List<double> UpdateVbles(int idClient), que permiten establecer un conjunto de variables monitorizadas, de manera que no haga falta hacer un query para cada variable, sino que se hace MonitorizeVbles una vez y a partir de ese momento, cada vez que se haga UpdateVbles se obtendrá el valor de todas las variables marcadas para monitorizar.

La interfaz ISimulatorService que debe implementarse en los simuladores para que funcionen bien en la API es:

```
[ServiceContract]
public interface ISimulatorService
{
    [OperationContract]
    /*Initialize all the queues that will be involve in the program*/
    int Initialize();

    [OperationContract]
    void SendRun(int idClient);

    [OperationContract]
    void SendFreeze(int idClient);

    [OperationContract]
    double VbleQuery(int idClient, string vble);

    [OperationContract]
    void VbleUpdate(int idClient, string nvar, double vble);

    [OperationContract]
    List<double> MonitorizeVbles(int idClient, List<string> variables);

    [OperationContract]
```

```
        List<double> UpdateVbles(int idClient);

        [OperationContract]
        State GetState(int idClient);

        [OperationContract]
        void FinishClient(int idClient);
    }

    [DataContract(Namespace = "InteractiveExcelDocsForSims.ISimulatorService")]
    public struct State
    {
        [DataMember]
        public bool state { get; set; }

        [DataMember]
        public long nStep { get; set; }

        [DataMember]
        public double timet { get; set; }
    }
```

Se incluye en el proyecto la clase SimulatorServiceClient, que permite acceder al servicio de ejemplo incluido en el proyecto. Este cliente no debe eliminarse bajo ninguna circunstancia, ya que las clases lo utilizan. Lo correcto es utilizar este mismo cliente para todos los simuladores añadidos al proyecto.

En caso de haber solo un simulador, se puede cambiar la configuración del service reference SimulatorService para que apunte al simulador de interés. Esto se haría cambiando app.config. Darle a actualizar servicio o a cambiar la configuración con el editor de Visual Studio puede provocar en ocasiones que la clase SimulatorServiceClient cambie de nombre, lo cual haría que la API no funcionase (y que no se pudiesen usar los componentes y gestores de otras aplicaciones). Por esta razón, si hace falta utilizar algún método adicional de un simulador o hay algún valor de la configuración que no se puede cambiar simplemente con app.config, se debe realizar alguno de los siguientes procedimientos:

A) Entrar en la carpeta del proyecto donde están los service references y entrar en la carpeta de SimulatorService (usando el explorador de Windows) y cambiar la clase cliente contenida en references.cs para que se llame SimulatorServiceClient (para eso basta con coger el nombre de cliente que se quiere cambiar, por ejemplo, SimulatorNuclearClient, y reemplazar todas las ocurrencias de esta palabra en el archivo por SimulatorServiceClient). Desde el propio visual studio esto no se puede hacer, porque no deja cambiar los archivos autogenerados. Esta opción no me ha dado problemas, pero puede asustar por modificarse a mano un archivo autogenerado.

B) Crear una nueva referencia de servicio para el simulador. Después, siempre que se pueda se debería utilizar SimulatorServiceClient para realizar las operaciones con el simulador (más adelante se explica cómo hacer esto).

En caso de haber más de un simulador, o uno de ellos o los 2 deberán crearse con una referencia de servicio nueva (esto no cambia que se deberían manejar con SimulatorServiceClient siempre que se pueda para maximizar la reutilización)

A la hora de manejar un simulador que no está directamente conectado a SimulatorService Client (por ejemplo, cuando se ha creado una nueva referencia de servicio para el simulador), sigue siendo básico y esencial poder utilizar SimulatorServiceClient para enviar mensajes a este

simulador. Si no se pudiese, no sería posible utilizar clases como CtrSubestadosSimulador o componentes y gestores de otros proyectos.

Para hace esto, se debe configurar los componentes y gestores (cambiar su propiedad ConfigSimulador o la propiedad general de configuración de SimuGlobals), así como CtrSubestadosSimulador utilizando una configuración que use el nombre del endpoint de SimulatorServiceClient, pero cogiendo los otros datos de conexión del simulador en cuestión, generalmente la Uri, (se puede usar para coger los datos el método ExcelTools. GetDefaultEndpoints).

En caso de que algún simulador no cumpla la interfaz, se debería implementar un servicio intermediario que convirtiese las operaciones de una interfaz a otra.

1.9. ExcelTools

Existen problemas en el manejo de muchos de los elementos de la programación en Excel. Para facilitar la programación, se ofrece la clase ExcelTools, que incluye un conjunto de métodos estáticos que permiten realizar acciones que, si no existiese esta clase, serían muy difíciles de hacer.

1.9.1. Manejo de WCF

Para el manejo de las clases de Windows Communication Foundation, se proporcionan 2 métodos:

- GetUriClienteWCF. Este método recibe un cliente WCF y devuelve la URI asociada a dicho cliente.
- GetDefaultEndpoints. Devuelve el conjunto de datos de los clientes que se encuentran en el archivo app.config. Como clave para los diccionarios se utiliza el nombre del endpoint (que puede ser encontrado en app.config). Este método es muy útil para poder crear configuraciones de conexiones que envíen mensajes a un simulador distinto del apuntado por SimulatorService pero usando SimulatorServiceClient para los mensajes. También sirve para poder realizar pequeños cambios sobre la configuración por defecto a la hora de establecer la configuración para los clientes.

1.9.2. Manejo de objetos Range

Para los objetos Range, hay muchas cosas que con las APIs de Microsoft no se pueden hacer de manera sencilla y que son muy básicas.

La clase ExcelTools ofrece las siguientes operaciones sobre objetos Range:

Cuenta simplificación y características de la celda

- UInt64 TotalCount(Range range). Devuelve el número de celdas del objeto Range. A diferencia de la propiedad Count normal de Range, este método no lanza una excepción cuando se hace sobre selecciones de filas o columnas completas, sino que devuelve el número total de celdas independientemente del tipo de conjunto representado.
- Range SimplificaDireccion(Range range). Simplifica el objeto Range en el sentido de que la propiedad Address del objeto range queda simplificada. Por ejemplo, si uno crea un objeto Range llamando a Evaluate con las siguiente cadena,”A1,A1,A1”, realmente

la dirección que queda asociada al Range es “A1,A1,A1”. Tras llamar al método, la dirección que queda asociada es A1. Se usa sobretodo de manera interna.

- List<Range> SimplificaConjunto(List<Range> ranges). Toma un conjunto de objetos Range, y devuelve un conjunto que tiene un único objeto Range por cada hoja. Por ejemplo, si tenemos 2000 objetos Range que apuntan a celdas distintas de una misma hoja, este método devolvería una lista con un único objeto Range que representaría todas las celdas que estaban contenidas en ese conjunto de 2000. Al igual que el anterior, es sobretodo usado de manera interna, pero dependiendo del tipo de elementos que se desarrolle, puede resultar útil.
- List<Range> ObtenSubconjuntos(Range range). Toma un objeto Range y devuelve una lista que contiene un objeto Range por cada conjunto de celdas que se podrían representar como casilla:casilla, y que por tanto no contienen ninguna "," en su dirección (son por tanto, conjuntos de forma retangular, finitos o infinitos). Es posible que los conjuntos rectangulares se pisen unos a otros.
- void TieneFilasOColumnasCompletasRect(Range range, out bool tieneFilasCompletas, out bool tieneColumnasCompletas). Devuelve si el conjunto de celdas contiene alguna fila o columna completa. El método es solo para objetos Range rectangulares simples (sin ninguna “,” en su address).

Limites, ancho y alto

- GetPrimeraColumna, GetPrimeraFila, GetUltimaColumna y GetUltimaFila devuelven la mínima columna, la mínima fila, la máxima columna y la máxima fila en la que hay alguna celda que pertenece al objeto range pasado como parámetro. También hay los mismos métodos pero versión para objetos range rectangulares simplificados.
- GetAncho y GetAlto. Devuelven el ancho y alto, respectivamente, de un objeto Range.

Creación de objetos Range

- Range NewRangeFilasCompletas(Worksheet worksheet, int primeraFila, int ultimaFila). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya fila se encuentra entre primeraFila y ultimaFila (incluidos).
- Range NewRangeColumnasCompletas(Worksheet worksheet, int primeraColumna, int ultimaColumna). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya columna se encuentra entre primeraColumna y ultimaColumna (incluidos).
- public static Range NewRange(Worksheet worksheet, int primeraFila, int primeraColumna, int ultimaFila, int ultimaColumna). Crea un nuevo objeto Range que representa todas las celdas de la hoja worksheet cuya columna se encuentra entre primeraColumna y ultimaColumna (incluidos) y cuya fila se encuentra entre primeraFila y ultimaFila (incluidos).

Equals y operaciones de conjuntos

- bool AreEqual(Range range1, Range range2), bool AreEqual(List<Range> ranges1, Range range2), bool AreEqual(Range range1, List<Range> ranges2) y bool AreEqual(List<Range> ranges1, List<Range> ranges2). Devuelve true si los 2 conjuntos representan el mismo conjunto de celdas.
- List<Range> Union(List<Range> ranges1, List<Range> ranges2), List<Range> Union(List<Range> ranges1, Range range2), List<Range> Union(Range range1, List<Range> ranges2) y List<Range> Union(Range range1, Range range2). El resultado es una lista de objetos Range donde hay un objeto range por cada hoja, y el conjunto de objetos Range representa la unión de todos los objetos range pasados como parámetro.
- Range UnionMismoWorksheet(Range range1, Range range2). Realiza una union de 2 objetos range de la misma hoja. En caso de que no sean de la misma hoja, saltará una excepción.
- List<Range> Interseccion(List<Range> ranges1, Range range2), List<Range> Interseccion(Range range1, List<Range> ranges2), List<Range> Interseccion(List<Range> ranges1, List<Range> ranges2) y Range Interseccion(Range range1, Range range2). Devuelve la intersección del primer conjunto con el segundo. La lista devuelta contiene un objeto range por cada hoja (o un único objeto range en el caso de la intersección de objetos Range simples).
- List<Range> Diferencia(List<Range> ranges1, Range range2), List<Range> Diferencia(Range range1, List<Range> ranges2), List<Range> Diferencia (List <Range> ranges1, List<Range> ranges2) y Range Diferencia(Range range1, Range range2). Realiza la operación de conjuntos diferencia, y devuelve el resultado. El resultado es conjunto1-conjunto2. La lista devuelta contiene un objeto range por cada hoja (excepto si los 2 conjuntos son objetos Range, donde se devuelve un objeto Range).

1.10. Simulador de ejemplo

Hay implementado un simulador de ejemplo. El simulador es un simulador muy simple con 5 variables. Se guarda un hashset con los clientes activos. Si se intenta hacer una monitorización pasando un id de un cliente que no está en la lista, salta una excepción.

En caso de que el estado sea corriendo, cada segundo el simulador da un paso, y por tanto nStep aumenta en 1. Además, en cada paso se modifica el valor de las variables VAR1, VAR2, VAR3 y VAR4 según la siguiente fórmula:

- long x = estadoSimulador.nStep + 1;
- var1 = Math.Pow(x%10, 2);
- var2 = x%100;
- var3 = Math.Log((x%10)+1)*50;
- var4 = ((x - 1 / 5) % 2) == 0 ? 0 : 10;

SimulatorServiceClient por defecto apunta a este simulador, pero no es difícil configurarlo para que apunte a un simulador distinto.

Este simulador puede ser útil, por ejemplo, para poder tener algo sobre lo que probar el programa cuando no se tiene el simulador, o el acceso a éste es muy complicado.

2. Ejemplo de aplicación

A continuación se explican los detalles de una aplicación de ejemplo que se ha desarrollado usando el framework. Con este pequeño proyecto, se ofrece un ejemplo que muestra cómo crear proyectos usando el framework, cómo crear componentes y gestores, y cómo utilizar las herramientas del framework.

2.1. Introducción

La aplicación Excel consiste en una macro para Excel de nivel de documento que interactúa con 2 simuladores y contiene 3 controles:

- Un control que permite crear botones para arrancar y parar el simulador. Al crear el botón, no se ofrecen opciones.
- Un control que permite monitorizar una variable y escribir el valor de la variable del simulador en una casilla. Al crear la monitorización se permite especificar el nombre de la variable y el periodo.
- Un control que permite crear gráficas que muestran la evolución de un conjunto de variables a lo largo de los pasos. Al crear la gráfica se permite especificar el conjunto de variables que mostrar, y el periodo.

En cuanto a los simuladores, son muy similares. Los 2 tienen 5 variables. La única diferencia es cómo evolucionan estas variables.

2.2. Creación de los simuladores

Lo primero que se hace en este ejemplo es desarrollar los simuladores. El primero de los simuladores simplemente es el simulador de ejemplo que viene ya incluido en el proyecto base.

El segundo sí que es un simulador nuevo. Para crearlo, se ha creado una nueva biblioteca de servicios WCF. Como la idea es que el simulador siga la interfaz genérica definida en este proyecto, se ha cogido la interfaz ISimulatorService (que se puede encontrar en el manual de uso y en el proyecto WcfServicioDeEjemplo), y se ha reemplazado la interfaz con la que comenzaba el proyecto por ésta (cambiando el archivo de configuración para que apunte a la interfaz ISimulatorService que se acaba de copiar).

Después, simplemente se ha implementado las operaciones del simulador que aparecen en esta interfaz en la clase que implementa el servicio (en este caso, por simplificar, se ha decidido llamar a la clase SimulatorService). La implementación es muy similar que la WcfServicioDeEjemplo, por lo que no merece la pena perder mucho el tiempo explicando cómo ha sido implementado. La única diferencia es que para var1 se calcula el seno, para var2 el coseno, y para var3 seno más coseno. Var4 se calcula igual que en el simulador de ejemplo.

Finalmente, se hace que SimulatorService apunte al primer simulador (en este caso no hay que hacer nada, ya que por defecto ya apunta al primer simulador de ejemplo), y, como tenemos 2 simuladores, se debe añadir una nueva referencia de servicio para el segundo simulador. Esto crea un conjunto de clases para este simulador y modifica automáticamente app.config añadiendo los datos de la nueva conexión.

2.3. Elementos generales de la aplicación

Antes de entrar en los controles, existen un conjunto de elementos de la aplicación que son independientes de los controles.

2.3.1. Panel de introducción de datos de conexión

Primero, por el tema de que hay varios simuladores entre los que elegir en la aplicación, y para permitir que se pueda cambiar el puerto o dirección ip en caso de que sea necesario, se ofrece en el arranque de la aplicación un panel en el que se puede elegir el simulador y se puede seleccionar el puerto y dirección ip.

Para el panel se utiliza el panel PanelDatosDeConexion que viene incluido en el proyecto base, pasando la lista de strings "SIMULADOR1","SIMULADOR2". Si este panel no viniese ya incluido, haría falta desarrollarlo desde 0.

Una vez se obtienen los datos de qué simulador se quiere, se utiliza ExcelTools.GetDefaultEndpoints() para obtener un diccionario con todas las configuraciones de endpoints incluidas en el proyecto (en este caso hay 2, una por simulador). Del conjunto cogemos la dirección (URI) encontrada en la configuración del simulador elegido. Esta dirección se modifica poniendo el nuevo puerto y dirección IP usando UriBuilder.

Se crea un nuevo objeto PropiedadesDeConexion, con el nombre de la configuración de endpoint de SimulatorServiceClient, y se pasa la dirección que se acaba de extraer.

Para que se puedan hacer consultas del estado del simulador, se llama a MonitorDeConexiones.AddMonitorizacionDeEstado con la URI del cliente. Finalmente, se utiliza MonitorDeConexiones.GetEstadoConexion para comprobar si la configuración corresponde a un simulador válido (o conectado). En caso de que sí, se continua, sino se vuelve a mostrar la ventana (y se quita la conexión de las monitorizadas).

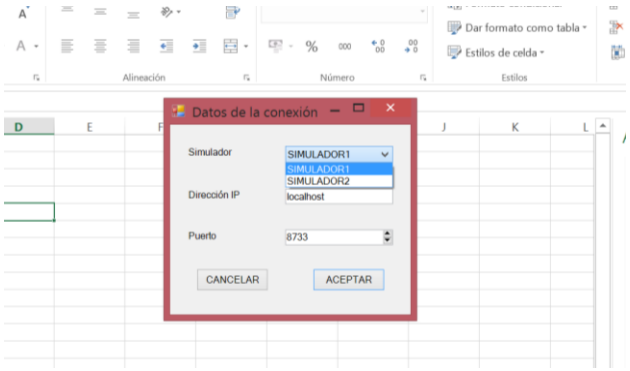

Figura 14. En la imagen se puede ver el panel de introducción de datos se simulación

2.3.2. Panel de información de estado

Lo siguiente que hay que montar es el panel de información de estado. El proyecto base ya proporciona un actions pane que muestra el estado de los simuladores.

El panel comienza vacío, por lo que hay que incluir el simulador. Para esto se usa this.PanelInformacionDeEstado.AddServicio para añadir la URI que se acaba de extraer con la ventana. Finalmente, se llama a SetControladorDeSubestados, pasando un nuevo objeto CtrSubestadosSimulador con la configuración antes montada. Ya está, con esto está configurado el panel de estado del simulador.

2.3.3. Configuración de los componentes y gestores

Aún no se han desarrollado los componentes y gestores, sin embargo, esto no nos impide configurarlos. Para eso, cambiamos la propiedad SimuGlobals.Instance.ConfigBaseSimuladores utilizando la configuración que montó antes, y a partir de ese momento todos los gestores y componentes que desarrollemos usarán dicha configuración.

2.3.4. Nombre y otros elementos de la interfaz

Se modifica el archivo ParametrosGenerales para cambiar el nombre de la aplicación que se muestra en el panel. En este ejemplo, se pone Iceberg como nombre de la aplicación. La contraseña para modo edición se deja como está.

Con esto, se tiene una aplicación básica sin controles, pero que tiene un panel en el que se ve el estado del simulador cuando se está en modo ejecución (y también se ve el nombre de la aplicación), en la que arriba se ve el modo de uso actual, y con un botón de cambio de modo y uno de mostrar y ocultar el panel en la barra Ribbon, y que permite cambiar de modo con la contraseña especificada. Además, al hacer el cambio de modo este modo se conserva y es accesible desde todos los elementos del proyecto.

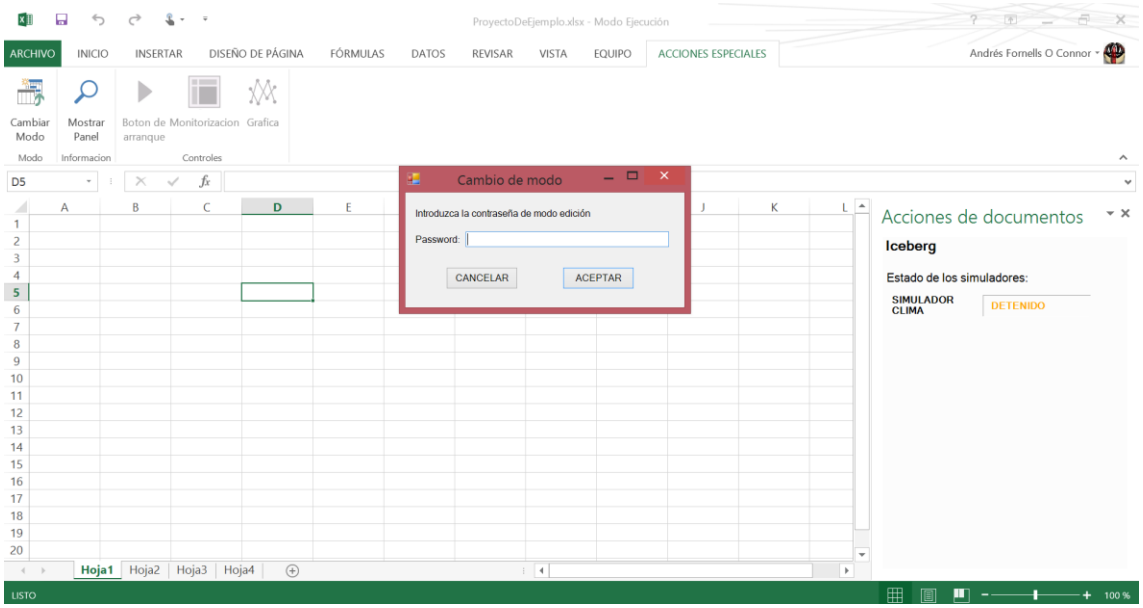

Figura 15. En la imagen es pueden ver los elementos comentados, incluido el panel de cambio de modo

2.4. Controles

A continuación se describe como se implementan los diferentes controles. Solo se describirán el gestor de gráficas y el componente de monitorización (de casillas), porque el control del botón es igual que el de gráficas, es simplemente otro gestor. Tiene algún detalle diferente, pero en lo que se refiere al uso del framework es prácticamente igual.

2.4.1. Control de gráficas de monitorización de variables

Este control se ha implementado con 2 clases: GestorDeGraficas y OperacionGrafica, un form: FormNuevaGrafica y un botón en la barra Ribbon. El botón de la barra Ribbon lo único que hace es OperacionGrafica.Run(). Los nombres están escogidos de acuerdo al manual de uso para facilitar la reutilización de los diferentes elementos.

2.4.1.1. FormNuevaGrafica

El form es utilizado para permitir al usuario especificar las variables que mostrar en la gráfica y el periodo que se debe utilizar para la lectura de los valores de las variables. El form, tal y como se recomienda en el manual de uso, devuelve DialogResult.OK en caso de que el usuario acepte y Cancel en caso de que no. Además, se limita a recoger los datos, por lo que contiene 2 propiedades para consultar los valores introducidos por el usuario una vez éste pulse aceptar, que son: VarsSeleccionadas y Periodo.

2.4.1.2. OperacionGrafico

Esta clase tiene un único método estático Run, que lo que hace es mostrar un nuevo form del tipo FormNuevaGrafica. Si éste devuelve DialogResult.OK, entonces llama al método correspondiente de GestorDeGraficas para montar una nueva gráfica en el conjunto de casillas seleccionadas.

2.4.1.3. GestorDeGraficas

Esta clase es un gestor de elementos. Los objetos que gestiona son un tipo interno llamado GraficaLineal, que representa una gráfica individual.

Como debe hacerse siempre con los gestores, se implementa el patrón singleton incluido en el manual de uso. A continuación, como el gestor va a manejar conexiones con un simulador, se incluyen las propiedades ConfigSimulador y ClientePrincipal, que hacen que el gestor sea perfectamente reutilizable en otro proyecto diferente independientemente del simulador utilizado.

Además, se añade una propiedad int? idCliente, que contiene el id de cliente en el simulador y un método InicializaCliente privado que comprueba si hay un id de cliente y en caso de no haber uno, inicializa un cliente en el simulador. También hay un método FinalizaCliente, que en caso de haber un cliente, llama al método FinishClient del simulador.

Para que los objetos gestionados persistan y reciban una llamada al método initialize y Terminate, se guarda una propiedad _GestorInterno de tipo GestorDeElementos <GraficaLineal>.

La clase implementa la interfaz IInitializable para realizar operaciones de inicialización y de cierre. En la inicialización llama a _GestorInterno.Initialize, y en el método Terminate llama a _GestorInterno.Terminate.

En el método initialize, se llama al método AddMonitorizacionDeEstado, consiguiendo que el componente sea independiente y que se puedan realizar consultas de estado de la conexión sin tener que esperar a que se realice la consulta. Después se llama al método InicializaCliente. Finalmente, se suscribe al evento de hoja cerrada para poder eliminar las gráficas asociadas a la hoja en el cierre y evitar que salten excepciones por todas partes.

Realizar las consultas de conexión por MonitorDeConexiones es más eficiente, y además en caso de pérdida de conexión no produce un bloqueo permanente de la aplicación.

En la función Terminate se llama a FinalizaCliente (para que no se acumulen los datos en el simulador).

Adicionalmente, se tiene el método Instance_SheetClosed, que responde al evento de hoja cerrada. En este método, simplemente se eliminan los graficos asociados a cada hoja. Los gráficos contienen objetos GraficoFlotanteSerializable, que contienen una propiedad WorksheetRef que puede ser accedida aunque se haya cerrado ya la hoja Excel. Se usa esta propiedad para ver si pertenecen al conjunto de hojas eliminadas.

Por último, hay un método para crear un gráfico. Este método es usado por OperacionGrafico para solicitar que se cree un nuevo gráfico que coincida con los criterios especificados por el usuario. Este método simplemente crea un objeto del tipo GraficoLineal, lo inicializa y los añade a _GestorInterno con AddElemento, el cual se encarga de que sea persistente.

2.4.1.4. GraficaLineal

Esta clase está definida dentro de GestorDeGraficas, y es la que realmente controla el gráfico. La clase implementa IInitializable (ya que tiene funciones de inicialización y finalización, y porque si no no se podría añadir al gestor).

Además de las propiedades normales, como el periodo y la lista de variables monitorizadas, contiene un objeto de tipo GraficoFlotanteSerializable (una referencia al gráfico). Este objeto permite apuntar al gráfico de la hoja de manera que tras el cierre del documento, se siga apuntando al mismo gráfico.

La gráfica solo debería actualizar los datos en modo ejecución. Para no tener temporizadores de sobra corriendo en el sistema, se lanza el temporizador cuando se entra en modo ejecución, y se para cuando se entra en modo edición. Para eso, en el método Initialize se suscribe el método Instance_ModoCambiado al evento de ModoCambiado de SimuGlobals. En este método se llama a ConfiguraSegunModo. Además, en initialize, tras suscribirse al evento, se llama a ConfiguraSegunModo.

En ConfiguraSegunModo, se lanza o detiene la tarea según el modo pasado como parámetro. Para crear la tarea periódica, se utiliza el método AddTareaPeriodica de SistemaCompDinamicos. Para parar la tarea, se utiliza RemoveTareaPeriodica. Como periodo se pone el periodo pasado en el constructor, y como método se pasa un método interno llamado ActualizaGrafica.

En el método ActualizaGrafica, se comprueba que el simulador sea accesible con el método GetEstadoConexion de MonitorDeConexiones. Si es accesible, se obtiene el estado. Del estado se extrae el paso actual, para saber si modificar la gráfica o no, y para saber dónde colocar el valor en la gráfica. Finalmente, se coge el valor actual de las variables de la gráfica y se actualiza el conjunto mostrado. Se usa el cliente de GestorDeGraficas para realizar todas las operaciones.

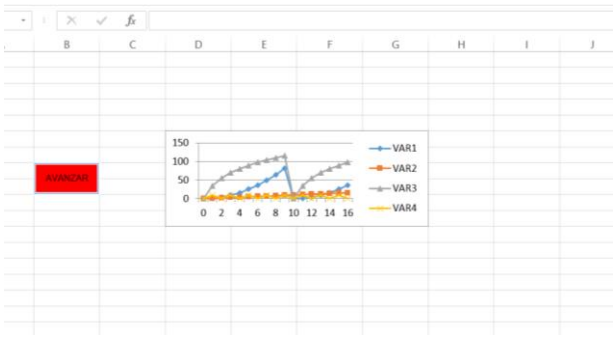

Figura 16. En la imagen se puede ver la gráfica tras unos cuantos pasos.

2.4.2. Control de monitorización de variables en celdas

Este control se ha implementado con 2 clases: ComponenteMonitorDeVariables y OperacionMonitorizar, un form: FormMonitorizacion y un botón en la barra Ribbon. El botón de la barra Ribbon lo único que hace es OperacionMonitorizar.Run(). Los nombres están escogidos de acuerdo al manual de uso para facilitar la reutilización de los diferentes elementos.

La implementación de los diferentes elementos no es muy diferente de la del control de gráficas, y no tiene sentido repetir otra vez lo mismo. Por eso, solo se comentarán los elementos que son distintos.

La clase contiene una clase interna llamada DatosCopia, que implementa IDatosComportamientoCelda. Esta clase contiene el nombre de la variable y el periodo, y en la implementación del método PegaComportamiento, llama a CreaMonitorizacion de ComponenteMonitorDeVariables para realizar el pegado.

Internamente se almacena un diccionario de RangeSerializable a TupleSerializable<string, int>, que utiliza como comparador MismasCeldasComparer. Este diccionario asocia a las celdas pares variable periodo. Al usar mismas celdas comparer, en la comparación realmente se compara que la celda es la misma, en vez de hacer una comparación por referencia. Se utiliza un DictionarySerializable que recibe MismasCeldasComparer como tercer parámetro tipo.

En el método de creación de monitorizaciones, se utiliza el método AddControladorOpCeldas de SistemaOpCeldas para suscribir un controlador al conjunto de celdas que monitoriza. Para que se monitoricen las de antes y las de ahora, se utiliza ExcelTools.Union sobre el conjunto obtenido de hacer GetCeldasAsociadas y el nuevo objeto Range.

Como controlador se utiliza la propia clase ComponenteMonitorDeVariables, que implementa IControladorOpCeldas. Para esto, se incluye el método CopiaComportamiento, en el cual se devuelve un objeto DatosCopia con los datos de variable y periodo, y el método DestruyeComportamiento, donde simplemente elimina la entrada de los diccionarios.

En el método initialize también se ejecuta el método AddControladorOpCeldas, pasando el conjunto de celdas que se encuentran en el diccionario de monitorizaciones (el cual persiste tras el cierre y apertura de la aplicación).

2.4.3. Persistencia, inicialización y terminación de los controles

Para marcar los controles como persistentes, tal y como se indica en el manual, se envuelve el atributo Instance en una propiedad en ThisWorkbook que está marcada con el atributo Cached. En la función de arranque del documento se llama a Initialize, y en la de cierre se llama a Terminate.

3. Pruebas en detalle

A continuación se incluyen todas las pruebas que se han realizado para el proyecto.

3.1. Pruebas automáticas: pruebas unitarias

Para este conjunto de pruebas, todas las clases que ejecutan pruebas tienen acceso a las siguientes variables:

```
        Excel.Worksheet hoja1;
        Excel.Worksheet hoja2;
        Excel.Chart hojaDeGrafico;
        Excel.ChartObject graficoHoja1;

        hoja1 = SimuGlobals.Instance.WorkbookExcel.Worksheets[Globals.Hoja1.Name];
        hoja2 = SimuGlobals.Instance.WorkbookExcel.Worksheets[Globals.Hoja2.Name];
        hojaDeGrafico =
SimuGlobals.Instance.WorkbookExcel.Charts[Globals.Gráfico1.Name];
        graficoHoja1 = (Globals.Hoja1.Chart_1.Parent as Excel.ChartObject);
```

Estas variables representan 2 hojas válidas, una hoja de gráfico válida y un gráfico que se encuentra en la hoja 1.

Además, las clases hacen uso de una clase desarrollada para el manejo de la serialización con XmlSerializer y también de una clase que compara strings ignorando ignorando las mayúsculas o minúsculas:

```
    public class HerramientasParaPruebas
    {
        public static string ToXmlString<T>(T input) where T : class
        {
            using (var writer = new StringWriter())
            {
                new XmlSerializer(typeof(T)).Serialize(writer, input);
                return writer.ToString();
            }
        }
        public static T FromXmlString<T>(string input) where T : class
        {
            using (var reader = new StringReader(input))
            {
                return new XmlSerializer(typeof(T)).Deserialize(reader) as T;
            }
        }
    }
    public class IgnoreCaseComparer : IEqualityComparer<string>
    {
        private static int nextInt = new Random().Next(200);

        public int intAleatorio = 0;
        public IgnoreCaseComparer() { intAleatorio = nextInt; nextInt++; }
        public bool Equals(string x, string y)
        {
            if (x == null || y == null) return x == y;
            return string.Equals(x, y, StringComparison.OrdinalIgnoreCase);
        }

        public int GetHashCode(string obj)
        {
            return obj.ToUpperInvariant().GetHashCode();
        }
    }
```

Pruebas para ExcelTools

```
        #region Datos WCF (clientes y datos de configuracion)
        public void GetUriClienteWCF_ClienteSimple_UriCorrecta()
        {
            SimulatorServiceClient clienteSimulador = new SimulatorServiceClient();
            string uriRecibida = ExcelTools.GetUriClienteWCF(clienteSimulador);

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo/Simulato
rService/", uriRecibida);
        }

        public void GetUriClienteWCF_ClienteSimple2_UriCorrecta()
        {
            SimulatorService2Client clienteSimulador = new SimulatorService2Client();
            string uriRecibida = ExcelTools.GetUriClienteWCF(clienteSimulador);

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/Simulat
orService/", uriRecibida);
        }

        public void GetUriClienteWCF_ClienteComplejo_UriCorrecta()
        {
            string uriUsada =
"http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/SimulatorService/";
            SimulatorServiceClient clienteSimulador = new
SimulatorServiceClient("BasicHttpBinding_ISimulatorService1", uriUsada);
            string uriRecibida = ExcelTools.GetUriClienteWCF(clienteSimulador);
            Assert.AreEqual(uriUsada, uriRecibida);
        }

        public void GetUriClienteWCF_ClienteNull_ExcepcionLanzada()
        {
            bool excepcionLanzada = false;
            SimulatorService2Client clienteSimulador = null;
            try
            {
                ExcelTools.GetUriClienteWCF(clienteSimulador);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }

        public void GetDefaultEndpoints_2Configs_ConfigsEncontradas()
        {
            Dictionary<string, ChannelEndpointElement> datosConfigs =
ExcelTools.GetDefaultEndpoints();
            Assert.AreEqual(2, datosConfigs.Count);

Assert.IsTrue(datosConfigs.ContainsKey("BasicHttpBinding_ISimulatorService1"));

Assert.IsTrue(datosConfigs.ContainsKey("BasicHttpBinding_ISimulatorService2"));

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo/Simulato
rService/", datosConfigs["BasicHttpBinding_ISimulatorService1"].Address.AbsoluteUri);

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/Simulat
orService/", datosConfigs["BasicHttpBinding_ISimulatorService2"].Address.AbsoluteUri);
        }

        #endregion
```

```
        #region Cuenta, simplificacion y descomposicion en subconjuntos cuadrados

        public void TotalCount_RangeCuadrado_CuentaCorrecta()
        {
            Range unRange = hoja1.Range["C3:F5"];
            Assert.AreEqual((UInt64)12, ExcelTools.TotalCount(unRange));
        }
        public void TotalCount_RangeComplejo_CuentaCorrecta()
        {
            Range unRange = (Range)hoja1.Evaluate("C3:F5, C3:D6, H10");
            Assert.AreEqual((UInt64)15, ExcelTools.TotalCount(unRange));
        }
        public void TotalCount_RangeNull_CuentaCorrecta()
        {
            Range unRange = null;
            Assert.AreEqual((UInt64)0, ExcelTools.TotalCount(unRange));
        }

        public void SimplificaDireccion_RangeSimplificado_QuedaIgual()
        {
            Range unRange = hoja1.Range["C3:F5"];
            Assert.IsTrue(ExcelTools.AreEqual(unRange,
ExcelTools.SimplificaDireccion(unRange)) );
        }
        public void SimplificaDireccion_CeldasRepetidas_SeSimplifica()
        {
            Range unRange = (Range)hoja1.Evaluate("G17, G17, G17");
            Assert.IsTrue(ExcelTools.AreEqual(hoja1.Range["G17"],
ExcelTools.SimplificaDireccion(unRange)));
        }
        public void SimplificaDireccion_RangeNull_QuedaIgual()
        {
            Range unRange = null;
            Assert.IsTrue(ExcelTools.AreEqual((Range)null,
ExcelTools.SimplificaDireccion(unRange)));
        }

        public void SimplificaConjunto_2RangesMismaHoja_SeSimplifica()
        {
            Range unRange = hoja1.Range["C3:F5"];
            Range unRange2 = hoja1.Range["D4:G6"];
            List<Range> ranges = new List<Range>(new Range[] { unRange, unRange2 });
            List<Range> rangesSimplificados = ExcelTools.SimplificaConjunto(ranges);
            Assert.AreEqual(1, rangesSimplificados.Count);
            Range unionCeldas = ExcelTools.UnionMismoWorksheet(ranges[0], ranges[1]);
            Assert.IsTrue(ExcelTools.AreEqual(unionCeldas, rangesSimplificados[0]));
        }
        public void SimplificaConjunto_RangesHojasDistintas_SeSimplifica()
        {
            Range unRange = hoja1.Range["D4:G6"];
            Range unRange2 = hoja2.Range["C3:F5"];
            Range unRange3 = hoja2.Range["D4:G6"];

            List<Range> ranges = new List<Range>(new Range[] { unRange, unRange2,
unRange3 });
            List<Range> rangesSimplificados = ExcelTools.SimplificaConjunto(ranges);
            Assert.AreEqual(2, rangesSimplificados.Count);
            Range unionCeldas = ExcelTools.UnionMismoWorksheet(ranges[1], ranges[2]);
            //Se crea un diccionario y se compara
            Dictionary<Worksheet, Range> celdasPorHojaSimplificado = new
Dictionary<Worksheet, Range>();
```

```
            celdasPorHojaSimplificado[rangesSimplificados[0].Worksheet] =
rangesSimplificados[0];
            celdasPorHojaSimplificado[rangesSimplificados[1].Worksheet] =
rangesSimplificados[1];

            Assert.IsTrue(ExcelTools.AreEqual(unRange,
celdasPorHojaSimplificado[unRange.Worksheet]));
            Assert.IsTrue(ExcelTools.AreEqual(unionCeldas,
celdasPorHojaSimplificado[unionCeldas.Worksheet]));
        }
        public void SimplificaConjunto_1Range_QuedaIgual()
        {
            Range unRange = hoja1.Range["C3:F5"];
            List<Range> ranges = new List<Range>(new Range[] { unRange });
            List<Range> rangesSimplificados = ExcelTools.SimplificaConjunto(ranges);
            Assert.AreEqual(1, rangesSimplificados.Count);
            Assert.IsTrue(ExcelTools.AreEqual(unRange, rangesSimplificados[0]));
        }
        public void SimplificaConjunto_ListaVacia_QuedaIgual()
        {
            List<Range> ranges = new List<Range>();
            List<Range> rangesSimplificados = ExcelTools.SimplificaConjunto(ranges);
            Assert.AreEqual(0, rangesSimplificados.Count);
        }
        public void SimplificaConjunto_Null_DevuelveListaVacia()
        {
            List<Range> ranges = null;
            List<Range> rangesSimplificados = ExcelTools.SimplificaConjunto(ranges);
            Assert.AreEqual(0, rangesSimplificados.Count);
        }

        public void ObtenSubconjuntos_RangeSimple_QuedaIgual()
        {
            Range unRange = (Range)hoja1.Evaluate("B3:C5");
            List<Range> subconjuntos
                = ExcelTools.ObtenSubconjuntos(unRange);
            Assert.AreEqual(1, subconjuntos.Count);
            ExcelTools.AreEqual(hoja1.Range["B3:C5"], subconjuntos[0]);
        }
        public void ObtenSubconjuntos_RangeComplejo_SeDivideEnSubconjuntos()
        {
            Range unRange = (Range)hoja1.Evaluate("B3:C5, H7:J9");
            List<Range> subconjuntos
                = ExcelTools.ObtenSubconjuntos(unRange);
            Assert.AreEqual(2, subconjuntos.Count);
            ExcelTools.AreEqual(hoja1.Range["B3:C5"], subconjuntos[0]);
            ExcelTools.AreEqual(hoja1.Range["H7:J9"], subconjuntos[1]);
        }
        public void ObtenSubconjuntos_RangeNull_DevuelveListaVacia()
        {
            Range unRange = null;
            List<Range> subconjuntos
                = ExcelTools.ObtenSubconjuntos(unRange);
            Assert.AreEqual(0, subconjuntos.Count);
        }

        #endregion

        #region Deteccion de tipo de Range (Con filas completas, con columnas
completas, etc...)
```

```
        public void
TieneFilasOColumnasCompletasRect_ConjuntoNormal_DevuelveNoFilasNiColumnasCompletas()
        {
            bool filasCompletas;
            bool columnasCompletas;

            Range unRange = (Range)hoja1.Evaluate("A1:D3");
            ExcelTools.TieneFilasOColumnasCompletasRect(unRange, out filasCompletas,
out columnasCompletas);
            Assert.IsFalse(filasCompletas);
            Assert.IsFalse(columnasCompletas);
        }
        public void
TieneFilasOColumnasCompletasRect_ConjuntoFilasCompletas_DevuelveFilasCompletas()
        {
            bool filasCompletas;
            bool columnasCompletas;

            Range unRange = (Range)hoja1.Evaluate("2:4");
            ExcelTools.TieneFilasOColumnasCompletasRect(unRange, out filasCompletas,
out columnasCompletas);
            Assert.IsTrue(filasCompletas);
            Assert.IsFalse(columnasCompletas);
        }
        public void
TieneFilasOColumnasCompletasRect_ConjuntoColumnasCompletas_DevuelveColumnasCompletas()
        {
            bool filasCompletas;
            bool columnasCompletas;

            Range unRange = (Range)hoja1.Evaluate("E:J");
            ExcelTools.TieneFilasOColumnasCompletasRect(unRange, out filasCompletas,
out columnasCompletas);
            Assert.IsFalse(filasCompletas);
            Assert.IsTrue(columnasCompletas);
        }
        public void
TieneFilasOColumnasCompletasRect_Hoja_DevuelveColumnasYFilasCompletas()
        {
            bool filasCompletas;
            bool columnasCompletas;

            Range unRange = hoja1.Cells;
            ExcelTools.TieneFilasOColumnasCompletasRect(unRange, out filasCompletas,
out columnasCompletas);
            Assert.IsTrue(filasCompletas);
            Assert.IsTrue(columnasCompletas);
        }
        public void
TieneFilasOColumnasCompletasRect_ConjuntoNull_DevuelveNoFilasNiColumnasCompletas()
        {
            bool filasCompletas;
            bool columnasCompletas;

            Range unRange = null;
            ExcelTools.TieneFilasOColumnasCompletasRect(unRange, out filasCompletas,
out columnasCompletas);
            Assert.IsFalse(filasCompletas);
            Assert.IsFalse(columnasCompletas);
        }

        #endregion
```

```
        #region Obtencion de limites

        public void GetLimites_ConjuntoNormal_DevuelveLimitesCorrectos()
        {
            Range conjunto = hoja1.Evaluate("D4:F4,C2:E5,D6,H6");
            Assert.AreEqual(2, ExcelTools.GetPrimeraFila(conjunto));
            Assert.AreEqual(3, ExcelTools.GetPrimeraColumna(conjunto));
            Assert.AreEqual(6, ExcelTools.GetUltimaFila(conjunto));
            Assert.AreEqual(8, ExcelTools.GetUltimaColumna(conjunto));
            Assert.AreEqual(6, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(5, ExcelTools.GetAlto(conjunto));
        }
        public void GetLimites_ConjuntoFilasCompletas_DevuelveLimitesCorrectos()
        {
            Range conjunto = hoja1.Evaluate("D4:F4,C2:E5,H6,3:4");
            Assert.AreEqual(2, ExcelTools.GetPrimeraFila(conjunto));
            Assert.AreEqual(1, ExcelTools.GetPrimeraColumna(conjunto));
            Assert.AreEqual(6, ExcelTools.GetUltimaFila(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count,
ExcelTools.GetUltimaColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(5, ExcelTools.GetAlto(conjunto));
        }
        public void GetLimites_ConjuntoColumnasCompletas_DevuelveLimitesCorrectos()
        {
            Range conjunto = hoja1.Evaluate("D4:F4,C2:E5,H6,E:G");
            Assert.AreEqual(1, ExcelTools.GetPrimeraFila(conjunto));
            Assert.AreEqual(3, ExcelTools.GetPrimeraColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count,
ExcelTools.GetUltimaFila(conjunto));
            Assert.AreEqual(8, ExcelTools.GetUltimaColumna(conjunto));
            Assert.AreEqual(6, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count, ExcelTools.GetAlto(conjunto));
        }
        public void
GetLimites_ConjuntoFilasYColumnasCompletas_DevuelveLimitesCorrectos()
        {
            Range conjunto = hoja1.Evaluate("D4:F4,C2:E5,H6,E:G,3:4");
            Assert.AreEqual(1, ExcelTools.GetPrimeraFila(conjunto));
            Assert.AreEqual(1, ExcelTools.GetPrimeraColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count,
ExcelTools.GetUltimaFila(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count,
ExcelTools.GetUltimaColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count, ExcelTools.GetAlto(conjunto));
        }
        public void GetLimites_ConjuntoHoja_DevuelveLimitesCorrectos()
        {
            Range conjunto = hoja1.Cells;
            Assert.AreEqual(1, ExcelTools.GetPrimeraFila(conjunto));
            Assert.AreEqual(1, ExcelTools.GetPrimeraColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count,
ExcelTools.GetUltimaFila(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count,
ExcelTools.GetUltimaColumna(conjunto));
            Assert.AreEqual(hoja1.Cells.Columns.Count, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(hoja1.Cells.Rows.Count, ExcelTools.GetAlto(conjunto));
        }
        public void GetLimites_ConjuntoNull_ExcepcionesLanzadasYAnchoYAlto0()
        {
```

```
            Range conjunto = null;
            bool excepcionPrimeraFila = false;
            bool excepcionPrimeraColumna = false;
            bool excepcionUltimaFila = false;
            bool excepcionUltimaColumna = false;
            //Excepciones
            try { ExcelTools.GetPrimeraFila(conjunto); }
            catch (ArgumentNullException) { excepcionPrimeraFila = true; }
            try { ExcelTools.GetPrimeraColumna(conjunto); }
            catch (ArgumentNullException) { excepcionPrimeraColumna = true; }
            try { ExcelTools.GetUltimaFila(conjunto); }
            catch (ArgumentNullException) { excepcionUltimaFila = true; }
            try { ExcelTools.GetUltimaColumna(conjunto); }
            catch (ArgumentNullException) { excepcionUltimaColumna = true; }
            //Asertos
            Assert.IsTrue(excepcionPrimeraFila);
            Assert.IsTrue(excepcionPrimeraColumna);
            Assert.IsTrue(excepcionUltimaFila);
            Assert.IsTrue(excepcionUltimaColumna);

            //Ahora finalmente se comprueba el ancho y alto
            Assert.AreEqual(0, ExcelTools.GetAncho(conjunto));
            Assert.AreEqual(0, ExcelTools.GetAlto(conjunto));
        }

        #endregion

        #region Creacion de objetos Range

        public void NewRangeFilasCompletas_1Fila_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeFilasCompletas(hoja1, 1, 1);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, hoja1.Range["1:1"]));
        }
        public void NewRangeFilasCompletas_3Filas_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeFilasCompletas(hoja1, 3, 6);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, hoja1.Range["3:6"]));
        }
        public void NewRangeFilasCompletas_ParametrosInconsistentes_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeFilasCompletas(hoja1, 6, 3);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }
        public void NewRangeFilasCompletas_HojaNull_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeFilasCompletas(null, 3, 6);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }

        public void NewRangeColumnasCompletas_1Columna_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeColumnasCompletas(hoja1, 1, 1);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, hoja1.Range["A:A"]));
        }
        public void NewRangeColumnasCompletas_3Columnas_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeColumnasCompletas(hoja1, 3, 6);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, hoja1.Range["C:F"]));
        }
        public void
NewRangeColumnasCompletas_ParametrosInconsistentes_CreaElNuevoRange()
```

```
        {
            Excel.Range celdas = ExcelTools.NewRangeColumnasCompletas(hoja1, 6, 3);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }
        public void NewRangeColumnasCompletas_HojaNull_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRangeColumnasCompletas(null, 3, 6);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }

        public void NewRange_ConjuntoNormal_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRange(hoja1, 3, 2, 7, 8);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, hoja1.Range["B3:H7"]));
        }
        public void NewRange_ParametrosInconsistentes_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRange(hoja1, 7, 8, 3, 2);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }
        public void NewRange_HojaNull_CreaElNuevoRange()
        {
            Excel.Range celdas = ExcelTools.NewRange(null, 3, 2, 7, 8);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, (Range)null));
        }

        #endregion

        #region Equals y operaciones de conjuntos (Union, Interseccion y Diferencia

        public void AreEqual_ComparaConjuntoNormalConHoja_DevuelveFalse()
        {
            Assert.IsFalse(ExcelTools.AreEqual(hoja1.Range["1:1"], hoja1.Cells));
        }
        public void AreEqual_ConjuntosSimplesDistintos_DevuelveFalse()
        {
            Assert.IsFalse(ExcelTools.AreEqual((Range)hoja1.Evaluate("D4:F4"),
(Range)hoja1.Evaluate("C4:E4")));
        }
        public void AreEqual_ConjuntosSimplesMismasCeldasDistintaHoja_DevuelveFalse()
        {
            Assert.IsFalse(ExcelTools.AreEqual((Range)hoja1.Evaluate("D4:F5"),
(Range)hoja2.Evaluate("D4:F5")));
        }
        public void AreEqual_ConjuntosSimplesIguales_DevuelveTrue()
        {
            Assert.IsTrue(ExcelTools.AreEqual((Range)hoja1.Evaluate("D4:F4"),
(Range)hoja1.Evaluate("D4:F4")));
        }
        public void AreEqual_ComparaConjuntosIgualesDistintoAddress_DevuelveTrue()
        {

Assert.IsTrue(ExcelTools.AreEqual((Range)hoja1.Evaluate("D4:F4,D5:E5,D6,F6"),
(Range)hoja1.Evaluate("D4:F4,D4:E5,D4:D6,F6")));
        }
        public void AreEqual_ComparaConjuntoNullConConjunto_DevuelveFalse()
        {
            Assert.IsFalse(ExcelTools.AreEqual((Range)null,
(Range)hoja1.Evaluate("D4:F5")));
        }
        public void AreEqual_ComparaConjuntosNull_DevuelveTrue()
        {
```

```
            Assert.IsTrue(ExcelTools.AreEqual((Range)null, (Range)null));
        }

        public void AreEqual_ListasSimplesDistintas_DevuelveFalse()
        {
            List<Range> lista1 = new List<Range>(new Range[] {
(Range)hoja1.Evaluate("D4:F4"), (Range)hoja1.Evaluate("C4:E4") });
            List<Range> lista2 = new List<Range>(new Range[] {
(Range)hoja1.Evaluate("D4:F4") });
            Assert.IsFalse(ExcelTools.AreEqual(lista1, lista2));
        }
        public void AreEqual_ListasSimplesIguales_DevuelveTrue()
        {
            List<Range> lista1 = new List<Range>(new Range[] {
(Range)hoja1.Evaluate("D4:E4"), (Range)hoja1.Evaluate("E4:F4") });
            List<Range> lista2 = new List<Range>(new Range[] {
(Range)hoja1.Evaluate("D4:F4") });
            Assert.IsTrue(ExcelTools.AreEqual(lista1,lista2));
        }

        public void Union_ConjuntosMismaHoja_DevuelveLaUnion()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja1.Evaluate("F5,E6");
            List<Range> union = ExcelTools.Union(range1, range2);
            Assert.AreEqual(1, union.Count);

Assert.IsTrue(ExcelTools.AreEqual(hoja1.Evaluate("D4:F6,F5,F6"),union[0]));
        }
        public void Union_ConjuntosDistintaHoja_DevuelveLaUnion()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja2.Evaluate("F5");
            Range range3 = (Range)hoja2.Evaluate("E6");
            List<Range> union = ExcelTools.Union(new List<Range>(new Range[]{range1,
range2}), range3);
            Assert.AreEqual(2, union.Count);
            Assert.IsTrue(ExcelTools.AreEqual(hoja2.Evaluate("F5,E6"),
union[0].Worksheet.Equals(hoja1) ? union[1] : union[0]));
            Assert.IsTrue(ExcelTools.AreEqual(hoja1.Evaluate("D4:F6"),
union[0].Worksheet.Equals(hoja1) ? union[0] : union[1]));
        }
        public void Union_ConjuntoConNull_DevuelveLaUnion()
        {
            Range range1 = hoja1.Evaluate("D4:F6,F5,F6");
            List<Range> union = ExcelTools.Union(new List<Range>(new Range[]{range1}),
(Range)null);
            Assert.AreEqual(1, union.Count);
            Assert.IsTrue(ExcelTools.AreEqual(range1, union[0]));
        }
        public void Union_NullConNull_DevuelveLaUnion()
        {
            List<Range> union = ExcelTools.Union((Range)null, (Range)null);
            Assert.AreEqual(0, union.Count);
        }

        public void UnionMismoWorksheet_ConjuntosMismaHoja_DevuelveLaUnion()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja1.Evaluate("F5,E6");
            Range union = ExcelTools.UnionMismoWorksheet(range1, range2);
            Assert.IsTrue(ExcelTools.AreEqual(hoja1.Evaluate("D4:F6,F5,F6"), union));
```

```
        }
        public void UnionMismoWorksheet_ConjuntosDistintaHoja_ExcepcionLanzada()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja2.Evaluate("F5");

            bool excepcionLanzada = false;
            try { ExcelTools.UnionMismoWorksheet(range1, range2); }
            catch(ArgumentOutOfRangeException) { excepcionLanzada = true; }

            Assert.IsTrue(excepcionLanzada);
        }
        public void UnionMismoWorksheet_ConjuntoConNull_DevuelveLaUnion()
        {
            Range range1 = hoja1.Evaluate("D4:F6,F5,F6");
            Range union = ExcelTools.UnionMismoWorksheet(range1, (Range)null);
            Assert.IsTrue(ExcelTools.AreEqual(range1, union));
        }
        public void UnionMismoWorksheet_NullConNull_DevuelveLaUnion()
        {
            Range union = ExcelTools.UnionMismoWorksheet((Range)null, (Range)null);
            Assert.IsNull(union);
        }

        public void Interseccion_ConjuntosMismaHoja_DevuelveLaInterseccion()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja1.Evaluate("F5,E6");
            Range interseccion = ExcelTools.Interseccion(range1, range2);
            Assert.IsTrue(ExcelTools.AreEqual(hoja1.Evaluate("F5,E6"), interseccion));
        }
        public void Interseccion_ConjuntosDistintaHoja_DevuelveLaInterseccion()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja2.Evaluate("E5:J12");
            Range range3 = (Range)hoja2.Evaluate("E6");
            List<Range> interseccion = ExcelTools.Interseccion(new List<Range>(new
Range[] { range1, range2 }), range3);
            Assert.AreEqual(1, interseccion.Count);
            Assert.IsTrue(ExcelTools.AreEqual(hoja2.Evaluate("E6"), interseccion[0]));
        }
        public void Interseccion_ConjuntoConNull_DevuelveLaInterseccion()
        {
            Range range1 = hoja1.Evaluate("D4:F6,F5,F6");
            List<Range> interseccion = ExcelTools.Interseccion(new List<Range>(new
Range[] { range1 }), (Range)null);
            Assert.AreEqual(0, interseccion.Count);
        }
        public void Interseccion_NullConNull_DevuelveLaInterseccion()
        {
            Range interseccion = ExcelTools.Interseccion((Range)null, (Range)null);
            Assert.IsNull(interseccion);
        }

        public void Diferencia_ConjuntosMismaHoja_DevuelveLaDiferencia()
        {
            Range range1 = (Range)hoja1.Evaluate("A1:D3,D3:H7");
            Range range2 = (Range)hoja1.Evaluate("C2:E4,G6");
            Range diferencia = ExcelTools.Diferencia(range1, range2);

Assert.IsTrue(ExcelTools.AreEqual((Range)hoja1.Evaluate("A1:D1,A2:B2,A3:B3,F3:H4,D5:H5
,D6:F6,H6,D7:H7"), diferencia));
```

```
        }
        public void Diferencia_ConjuntosDistintaHoja_DevuelveLaDiferencia()
        {
            Range range1 = hoja1.Range["D4:F6"];
            Range range2 = (Range)hoja2.Evaluate("E5:J12");
            Range range3 = (Range)hoja2.Evaluate("E6");
            List<Range> diferencia = ExcelTools.Diferencia(new List<Range>(new Range[]
{ range1, range2 }), range3);
            Assert.AreEqual(2, diferencia.Count);
            Assert.IsTrue(ExcelTools.AreEqual(hoja2.Evaluate("E5:J5,F6:J6,E7:J12"),
diferencia[0].Worksheet.Equals(hoja1) ? diferencia[1] : diferencia[0]));
            Assert.IsTrue(ExcelTools.AreEqual(hoja1.Evaluate("D4:F6"),
diferencia[0].Worksheet.Equals(hoja1) ? diferencia[0] : diferencia[1]));
        }
        public void Diferencia_ConjuntoConNull_DevuelveLaDiferencia()
        {
            Range range1 = hoja1.Evaluate("D4:F6,F5,F6");
            List<Range> diferencia = ExcelTools.Diferencia(new List<Range>(new Range[]
{ range1 }), (Range)null);
            Assert.AreEqual(1, diferencia.Count);
            Assert.IsTrue(ExcelTools.AreEqual(range1, diferencia[0]));
        }
        public void Diferencia_NullConConjunto_DevuelveLaDiferencia()
        {
            Range range1 = hoja1.Evaluate("D4:F6,F5,F6");
            List<Range> diferencia = ExcelTools.Diferencia((Range)null, new
List<Range>(new Range[] { range1 }));
            Assert.AreEqual(0, diferencia.Count);
        }
        public void Diferencia_NullConNull_DevuelveLaDiferencia()
        {
            Range diferencia = ExcelTools.Diferencia((Range)null, (Range)null);
            Assert.IsNull(diferencia);
        }

        #endregion
```

Pruebas para los tipos envoltorio para las clases básicas de CSharp

```
        #region DictionarySerializable

        public void DictionarySerializable_VacioSinComparer_SerializacionCorrecta()
        {
            DictionarySerializable<string, string> dictionary =
(DictionarySerializable<string, string>)new Dictionary<string, string>();
            string claseSerializada = HerramientasParaPruebas.ToXmlString(dictionary);
            DictionarySerializable<string, string> diccionarioRecuperado =
HerramientasParaPruebas.FromXmlString<DictionarySerializable<string,
string>>(claseSerializada);
            //Se comparan los 2 objetos diccionario internos
            Assert.IsNotNull(diccionarioRecuperado);
            Dictionary<string, string> dictionaryOriginal = dictionary;
            Dictionary<string, string> dictionary2 = diccionarioRecuperado;
            Assert.AreEqual(0, dictionary2.Count);
            Assert.AreEqual(new Dictionary<string, string>().Comparer,
dictionary2.Comparer);
        }
        public void DictionarySerializable_VacioStringComparer_SerializacionCorrecta()
        {
            DictionarySerializable<string, string, IgnoreCaseComparer> dictionary =
(DictionarySerializable<string, string, IgnoreCaseComparer>)new Dictionary<string,
string>(new IgnoreCaseComparer());
```

```
            string claseSerializada = HerramientasParaPruebas.ToXmlString(dictionary);
            DictionarySerializable<string, string, IgnoreCaseComparer>
diccionarioRecuperado =
HerramientasParaPruebas.FromXmlString<DictionarySerializable<string, string,
IgnoreCaseComparer>>(claseSerializada);
            //Se comparan los 2 objetos diccionario internos
            Assert.IsNotNull(diccionarioRecuperado);
            Dictionary<string, string> dictionaryOriginal = dictionary;
            Dictionary<string, string> dictionary2 = diccionarioRecuperado;
            Assert.AreEqual(0, dictionary2.Count);
            Assert.IsInstanceOfType(dictionary2.Comparer, typeof(IgnoreCaseComparer));
            //Se comprueba que efectivamente el comparer se ha conservado

Assert.AreEqual(((IgnoreCaseComparer)dictionaryOriginal.Comparer).intAleatorio,
((IgnoreCaseComparer)dictionary2.Comparer).intAleatorio);
        }
        public void
DictionarySerializable_ConElementosSinComparer_SerializacionCorrecta()
        {
            DictionarySerializable<string, string> dictionary =
(DictionarySerializable<string, string>)new Dictionary<string, string>();
            ((Dictionary<string, string>)dictionary).Add("UnaCadena", "UnValor");
            ((Dictionary<string, string>)dictionary).Add("OtraCadena", "OtroValor");
            string claseSerializada = HerramientasParaPruebas.ToXmlString(dictionary);
            DictionarySerializable<string, string> diccionarioRecuperado =
HerramientasParaPruebas.FromXmlString<DictionarySerializable<string,
string>>(claseSerializada);
            //Se comparan los 2 objetos diccionario internos
            Assert.IsNotNull(diccionarioRecuperado);
            Dictionary<string, string> dictionaryOriginal = dictionary;
            Dictionary<string, string> dictionary2 = diccionarioRecuperado;
            Assert.AreEqual(new Dictionary<string, string>().Comparer,
dictionary2.Comparer);
            Assert.AreEqual(dictionaryOriginal.Count, dictionary2.Count);
            foreach (KeyValuePair<string, string> parOriginal in dictionaryOriginal)
            {
                Assert.IsTrue(dictionary2.ContainsKey(parOriginal.Key));
                Assert.AreEqual(parOriginal.Value, dictionary2[parOriginal.Key]);
            }
        }
        public void
DictionarySerializable_ConElementosStringComparer_SerializacionCorrecta()
        {
            DictionarySerializable<string, string, IgnoreCaseComparer> dictionary =
(DictionarySerializable<string, string, IgnoreCaseComparer>)new Dictionary<string,
string>(new IgnoreCaseComparer());
            ((Dictionary<string, string>)dictionary).Add("UnaCadena", "UnValor");
            ((Dictionary<string, string>)dictionary).Add("OtraCadena", "OtroValor");
            string claseSerializada = HerramientasParaPruebas.ToXmlString(dictionary);
            DictionarySerializable<string, string, IgnoreCaseComparer>
diccionarioRecuperado =
HerramientasParaPruebas.FromXmlString<DictionarySerializable<string, string,
IgnoreCaseComparer>>(claseSerializada);
            //Se comparan los 2 objetos diccionario internos
            Assert.IsNotNull(diccionarioRecuperado);
            Dictionary<string, string> dictionaryOriginal = dictionary;
            Dictionary<string, string> dictionary2 = diccionarioRecuperado;
            Assert.IsInstanceOfType(dictionary2.Comparer, typeof(IgnoreCaseComparer));
            //Se comprueba que efectivamente el comparer se ha conservado

Assert.AreEqual(((IgnoreCaseComparer)dictionaryOriginal.Comparer).intAleatorio,
((IgnoreCaseComparer)dictionary2.Comparer).intAleatorio);
```

```
            Assert.AreEqual(dictionaryOriginal.Count, dictionary2.Count);
            foreach (KeyValuePair<string, string> parOriginal in dictionaryOriginal)
            {
                Assert.IsTrue(dictionary2.ContainsKey(parOriginal.Key));
                Assert.AreEqual(parOriginal.Value, dictionary2[parOriginal.Key]);
            }
        }

        #endregion

        #region HashSetSerializable

        public void HashSetSerializable_VacioSinComparer_SerializacionCorrecta()
        {
            HashSetSerializable<string> hashSet = (HashSetSerializable<string>)new
HashSet<string>();
            string claseSerializada = HerramientasParaPruebas.ToXmlString(hashSet);
            HashSetSerializable<string> hashSetRecuperado =
HerramientasParaPruebas.FromXmlString<HashSetSerializable<string>>(claseSerializada);
            //Se comparan los 2 objetos hashSet internos
            Assert.IsNotNull(hashSetRecuperado);
            HashSet<string> hashSetOriginal = hashSet;
            HashSet<string> hashSet2 = hashSetRecuperado;
            Assert.AreEqual(0, hashSet2.Count);
            Assert.AreEqual(new HashSet<string>().Comparer, hashSet2.Comparer);
        }
        public void HashSetSerializable_VacioStringComparer_SerializacionCorrecta()
        {
            HashSetSerializable<string, IgnoreCaseComparer> hashSet =
(HashSetSerializable<string, IgnoreCaseComparer>)new HashSet<string>(new
IgnoreCaseComparer());
            string claseSerializada = HerramientasParaPruebas.ToXmlString(hashSet);
            HashSetSerializable<string, IgnoreCaseComparer> hashSetRecuperado =
HerramientasParaPruebas.FromXmlString<HashSetSerializable<string,
IgnoreCaseComparer>>(claseSerializada);
            //Se comparan los 2 objetos hashSet internos
            Assert.IsNotNull(hashSetRecuperado);
            HashSet<string> hashSetOriginal = hashSet;
            HashSet<string> hashSet2 = hashSetRecuperado;
            Assert.AreEqual(0, hashSet2.Count);
            Assert.IsInstanceOfType(hashSet2.Comparer, typeof(IgnoreCaseComparer));
            //Se comprueba que efectivamente el comparer se ha conservado

Assert.AreEqual(((IgnoreCaseComparer)hashSetOriginal.Comparer).intAleatorio,
((IgnoreCaseComparer)hashSet2.Comparer).intAleatorio);
        }
        public void
HashSetSerializable_ConElementosSinComparer_SerializacionCorrecta()
        {
            HashSetSerializable<string> hashSet = (HashSetSerializable<string>)new
HashSet<string>();
            ((HashSet<string>)hashSet).Add("UnaCadena");
            ((HashSet<string>)hashSet).Add("OtraCadena");
            string claseSerializada = HerramientasParaPruebas.ToXmlString(hashSet);
            HashSetSerializable<string> hashSetRecuperado =
HerramientasParaPruebas.FromXmlString<HashSetSerializable<string>>(claseSerializada);
            //Se comparan los 2 objetos hashSet internos
            Assert.IsNotNull(hashSetRecuperado);
            HashSet<string> hashSetOriginal = hashSet;
            HashSet<string> hashSet2 = hashSetRecuperado;
            Assert.AreEqual(new HashSet<string>().Comparer, hashSet2.Comparer);
            Assert.AreEqual(hashSetOriginal.Count, hashSet2.Count);
```

```
            foreach (string valorOriginal in hashSetOriginal)
            {
                Assert.IsTrue(hashSet2.Contains(valorOriginal));
            }
        }
        public void
HashSetSerializable_ConElementosStringComparer_SerializacionCorrecta()
        {
            HashSetSerializable<string, IgnoreCaseComparer> hashSet =
(HashSetSerializable<string, IgnoreCaseComparer>)new HashSet<string>(new
IgnoreCaseComparer());
            ((HashSet<string>)hashSet).Add("UnaCadena");
            ((HashSet<string>)hashSet).Add("OtraCadena");
            string claseSerializada = HerramientasParaPruebas.ToXmlString(hashSet);
            HashSetSerializable<string, IgnoreCaseComparer> hashSetRecuperado =
HerramientasParaPruebas.FromXmlString<HashSetSerializable<string,
IgnoreCaseComparer>>(claseSerializada);
            //Se comparan los 2 objetos hashSet internos
            Assert.IsNotNull(hashSetRecuperado);
            HashSet<string> hashSetOriginal = hashSet;
            HashSet<string> hashSet2 = hashSetRecuperado;
            Assert.IsInstanceOfType(hashSet2.Comparer, typeof(IgnoreCaseComparer));
            //Se comprueba que efectivamente el comparer se ha conservado

Assert.AreEqual(((IgnoreCaseComparer)hashSetOriginal.Comparer).intAleatorio,
((IgnoreCaseComparer)hashSet2.Comparer).intAleatorio);
            Assert.AreEqual(hashSetOriginal.Count, hashSet2.Count);
            foreach (string valorOriginal in hashSetOriginal)
            {
                Assert.IsTrue(hashSet2.Contains(valorOriginal));
            }
        }

        #endregion

        #region ListSerializable

        public void ListSerializable_Vacio_SerializacionCorrecta()
        {
            ListSerializable<string> list = (ListSerializable<string>)new
List<string>();
            string claseSerializada = HerramientasParaPruebas.ToXmlString(list);
            ListSerializable<string> listRecuperado =
HerramientasParaPruebas.FromXmlString<ListSerializable<string>>(claseSerializada);
            //Se comparan los 2 objetos list internos
            Assert.IsNotNull(listRecuperado);
            List<string> listOriginal = list;
            List<string> list2 = listRecuperado;
            Assert.AreEqual(0, list2.Count);
        }
        public void ListSerializable_ConElementos_SerializacionCorrecta()
        {
            ListSerializable<string> list = (ListSerializable<string>)new
List<string>();
            ((List<string>)list).Add("UnaCadena");
            ((List<string>)list).Add("OtraCadena");
            string claseSerializada = HerramientasParaPruebas.ToXmlString(list);
            ListSerializable<string> listRecuperado =
HerramientasParaPruebas.FromXmlString<ListSerializable<string>>(claseSerializada);
            //Se comparan los 2 objetos list internos
            Assert.IsNotNull(listRecuperado);
            List<string> listOriginal = list;
```

```
            List<string> list2 = listRecuperado;
            Assert.AreEqual(listOriginal.Count, list2.Count);
            Assert.IsTrue(listOriginal.SequenceEqual(list2));
        }

        #endregion

        #region TupleSerializable

        public void TupleSerializable_2Parametros_SerializacionCorrecta()
        {
            TupleSerializable<string, int> tuple = (TupleSerializable<string,int>)new
Tuple<string,int>("Un string",17);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(tuple);
            TupleSerializable<string, int> tupleRecuperado =
HerramientasParaPruebas.FromXmlString<TupleSerializable<string,
int>>(claseSerializada);
            //Se comparan los 2 objetos tuple internos
            Assert.IsNotNull(tupleRecuperado);
            Tuple<string,int> tupleOriginal = tuple;
            Tuple<string,int> tuple2 = tupleRecuperado;
            Assert.AreEqual(tupleOriginal, tuple2);
        }
        public void TupleSerializable_3Parametros_SerializacionCorrecta()
        {
            TupleSerializable<string, int, char> tuple = (TupleSerializable<string,
int, char>)new Tuple<string, int, char>("Un string", 17, 'z');
            string claseSerializada = HerramientasParaPruebas.ToXmlString(tuple);
            TupleSerializable<string, int, char> tupleRecuperado =
HerramientasParaPruebas.FromXmlString<TupleSerializable<string, int,
char>>(claseSerializada);
            //Se comparan los 2 objetos tuple internos
            Assert.IsNotNull(tupleRecuperado);
            Tuple<string, int, char> tupleOriginal = tuple;
            Tuple<string, int, char> tuple2 = tupleRecuperado;
            Assert.AreEqual(tupleOriginal, tuple2);
        }
        public void TupleSerializable_4Parametros_SerializacionCorrecta()
        {
            TupleSerializable<string, int, char, float> tuple =
(TupleSerializable<string, int, char, float>)new Tuple<string, int, char, float>("Un
string", 17, 'z', 100f);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(tuple);
            TupleSerializable<string, int, char, float> tupleRecuperado =
HerramientasParaPruebas.FromXmlString<TupleSerializable<string, int, char,
float>>(claseSerializada);
            //Se comparan los 2 objetos tuple internos
            Assert.IsNotNull(tupleRecuperado);
            Tuple<string, int, char, float> tupleOriginal = tuple;
            Tuple<string, int, char, float> tuple2 = tupleRecuperado;
            Assert.AreEqual(tupleOriginal, tuple2);
        }

        #endregion
```

Pruebas de SimuGlobals

```
        public void PasswordEdicion_ArchivoConfigurado_MismoValorArchivoQueClase()
        {
            SimuGlobals instanciaOriginal = SimuGlobals.Instance;
            string passwordArchivo = ParametrosGenerales.PasswordEdicion;
            string passwordSimuGlobals = instanciaOriginal.PasswordEdicion;

            Assert.AreEqual(passwordArchivo, passwordSimuGlobals);
        }
        public void
RequierePasswordEdicion_ArchivoConfigurado_MismoValorArchivoQueClase()
        {
            SimuGlobals instanciaOriginal = SimuGlobals.Instance;
            bool requierePasswordArchivo;
            if
(!bool.TryParse(ParametrosGenerales.RequierePasswordEdicion.ToLowerInvariant(), out
requierePasswordArchivo)) requierePasswordArchivo = false;
            bool requierePasswordSimuGlobals =
instanciaOriginal.RequierePasswordEdicion;

            Assert.AreEqual(requierePasswordArchivo, requierePasswordSimuGlobals);
        }

        public void ModoActual_SeCambiaElModo_Persiste()
        {
            SimuGlobals instanciaOriginal = SimuGlobals.Instance;
            ModoUso modoAntiguo = instanciaOriginal.ModoActual;
            ModoUso nuevoModo = instanciaOriginal.ModoActual == ModoUso.EDICION ?
ModoUso.EJECUCION : ModoUso.EDICION;
            instanciaOriginal.ModoActual = nuevoModo;
            String simuGlobalsSerializado =
HerramientasParaPruebas.ToXmlString(instanciaOriginal);
            SimuGlobals instanciaDeserializada =
HerramientasParaPruebas.FromXmlString<SimuGlobals>(simuGlobalsSerializado);
            Assert.AreEqual(nuevoModo, instanciaDeserializada.ModoActual);

            instanciaOriginal.ModoActual = modoAntiguo;
        }
        public void PasswordEdicion_SeCambiaPassword_Persiste()
        {
            SimuGlobals instanciaOriginal = SimuGlobals.Instance;
            string passwordAntigua = instanciaOriginal.PasswordEdicion;
            string nuevoPassword = passwordAntigua + "CAMBIO";
            instanciaOriginal.PasswordEdicion = nuevoPassword;
            String simuGlobalsSerializado =
HerramientasParaPruebas.ToXmlString(instanciaOriginal);
            SimuGlobals instanciaDeserializada =
HerramientasParaPruebas.FromXmlString<SimuGlobals>(simuGlobalsSerializado);
            Assert.AreEqual(nuevoPassword, instanciaDeserializada.PasswordEdicion);

            instanciaOriginal.PasswordEdicion = passwordAntigua;
        }
        public void RequierePasswordEdicion_SeCambiaRequierePassword_Persiste()
        {
            SimuGlobals instanciaOriginal = SimuGlobals.Instance;
            bool requierePasswordAntiguo = instanciaOriginal.RequierePasswordEdicion;
            bool nuevoRequierePassword = !requierePasswordAntiguo;
            instanciaOriginal.RequierePasswordEdicion = nuevoRequierePassword;
            String simuGlobalsSerializado =
HerramientasParaPruebas.ToXmlString(instanciaOriginal);
            SimuGlobals instanciaDeserializada =
HerramientasParaPruebas.FromXmlString<SimuGlobals>(simuGlobalsSerializado);
```

```
            Assert.AreEqual(nuevoRequierePassword,
instanciaDeserializada.RequierePasswordEdicion);

            instanciaOriginal.RequierePasswordEdicion = requierePasswordAntiguo;
        }

        public void WorkbookExcel_Normal_LosWorkbookCoinciden()
        {
            Assert.AreEqual(Globals.ThisWorkbook, SimuGlobals.Instance.WorkbookExcel);
        }
        public void Application_Normal_LosApplicationCoinciden()
        {
            Assert.AreEqual(Globals.ThisWorkbook.Application,
SimuGlobals.Instance.Application);
        }

        private static bool eventoCambioDeModoSalto = false;
        private static bool eventoHojaCerradaSalto = false;
        private static ModoUso modoNuevo;
        private static Excel.Worksheet hojaCerrada;
        public void ModoCambiado_SeCambiaDeModo_ElEventoSalta()
        {
            SimuGlobals.Instance.ModoCambiado += Instance_ModoCambiado;

            modoNuevo = SimuGlobals.Instance.ModoActual == ModoUso.EDICION ?
ModoUso.EJECUCION : ModoUso.EDICION;
            SimuGlobals.Instance.ModoActual = modoNuevo;
            Assert.IsTrue(eventoCambioDeModoSalto);

            eventoCambioDeModoSalto = false;

            modoNuevo = SimuGlobals.Instance.ModoActual == ModoUso.EDICION ?
ModoUso.EJECUCION : ModoUso.EDICION;
            SimuGlobals.Instance.ModoActual = modoNuevo;
            Assert.IsTrue(eventoCambioDeModoSalto);

            SimuGlobals.Instance.ModoCambiado -= Instance_ModoCambiado;
        }

        private void Instance_ModoCambiado(object sender, ModoCambiadoEventArgs e)
        {
            eventoCambioDeModoSalto = true;
            Assert.AreEqual(modoNuevo, e.ModoDeUso);
        }

        public void SheetClosed_SeCierraUnaHoja_ElEventoSalta()
        {
            SimuGlobals.Instance.SheetClosed += Instance_SheetClosed;

            Excel.Worksheet nuevaHoja =
SimuGlobals.Instance.WorkbookExcel.Worksheets.Add();
            hojaCerrada = nuevaHoja;

            nuevaHoja.Application.DisplayAlerts = false;
            nuevaHoja.Delete();

            Assert.IsTrue(eventoHojaCerradaSalto);
            eventoHojaCerradaSalto = false;

            SimuGlobals.Instance.SheetClosed -= Instance_SheetClosed;
        }
```

```
        public void Instance_SheetClosed(object sender, SheetClosedEventArgs e)
        {
            eventoHojaCerradaSalto = true;
            Assert.AreEqual(1, e.HojasCerradas.Count);
            Assert.IsTrue(e.HojasCerradas.First().Equals(hojaCerrada));
        }

        public void ConfigBaseSimuladores_Normal_ValorInicialCorrecto()
        {
            Assert.AreEqual(PropiedadesDeConexion.ClientDefault,
SimuGlobals.Instance.ConfigBaseSimuladores);
        }
```

Pruebas SistemaOpCeldas

```
        #region Clases auxiliares para las pruebas

        public class ControladorCopiaTexto:IControladorOpCeldas
        {
            public IDatosComportamientoCelda CopiaComportamiento(Excel.Range
celdaCopiada, Excel.Range seleccionTotal)
            {
                return new DatosTexto(this, celdaCopiada.Value2 + "");
            }

            public void DestruyeComportamiento(Excel.Range celda, out bool
desasociaCelda)
            {
                celda.Value2 = null;
                desasociaCelda = true;
            }
        }

        public class DatosTexto : IDatosComportamientoCelda
        {
            string ContenidoTexto;
            ControladorCopiaTexto Controlador;
            public DatosTexto(ControladorCopiaTexto controlador, string
contenidoTexto)
            {
                this.ContenidoTexto = contenidoTexto;
                this.Controlador = controlador;
            }

            public void PegaComportamiento(Excel.Range celda)
            {
                celda.Value2 = ContenidoTexto;
                SistemaOpCeldas.SetCeldasAsociadas(Controlador,
ExcelTools.Union(SistemaOpCeldas.GetCeldasAsociadas(Controlador), celda));
            }
        }


        #endregion

        public void AddControladorOpCeldas_UnControladorUnRange_QuedaSuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { hoja1.Range["B2:D4"] });
```

```
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }
        public void AddControladorOpCeldas_UnControladorMuchosRange_QuedaSuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { hoja1.Range["B2:D4"], hoja2.Range["G5:I7"] });
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }
        public void AddControladorOpCeldas_UnControladorNull_QuedaSuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = null;
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }

        public void SetCeldasAsociadas_UnControladorUnRange_QuedaSuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { hoja1.Range["B2:D4"] });
            SistemaOpCeldas.SetCeldasAsociadas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }
        public void SetCeldasAsociadas_UnControladorMuchosRange_QuedaSuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { hoja1.Range["B2:D4"], hoja2.Range["G5:I7"] });
            SistemaOpCeldas.SetCeldasAsociadas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }
        public void SetCeldasAsociadas_UnControladorNull_QuedaSuscrito()
        {
```

```
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = null;
            SistemaOpCeldas.SetCeldasAsociadas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }

        public void RemoveControladorOpCeldas_UnControlador_QuedaDesuscrito()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { hoja1.Range["B2:D4"], hoja2.Range["G5:I7"] });
            SistemaOpCeldas.SetCeldasAsociadas(controladorCopiaTexto,
conjuntoSuscrito);
            List<Excel.Range> celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            //Se comprueba que las celdas son las mismas
            Assert.IsTrue(ExcelTools.AreEqual(conjuntoSuscrito, celdasAsociadas));

            //Se desuscribe y se comprueba que ahora es un conjunto vacio
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
            celdasAsociadas =
SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaTexto);
            Assert.IsTrue(ExcelTools.AreEqual(new List<Excel.Range>(),
celdasAsociadas));
        }

        public void CopiaPega_UnControladorCopiaMuchasCeldas_SePegaElTexto()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[]{(Excel.Range)hoja1.Evaluate("A1:C3,D4:F6")});
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            //Se escribe en las celdas
            hoja1.Range["C3"].Value2 = 1;
            hoja1.Range["C4"].Value2 = 2;
            hoja1.Range["D3"].Value2 = 3;
            hoja1.Range["D4"].Value2 = 4;

            //Se realiza la copia y el pegado
            hoja1.Activate();
            hoja1.Range["C3:D4"].Select();//Se selecciona el conjunto
            bool cancelDefault = false;
            SistemaOpCeldas.CopiaCeldasEsp(null, ref cancelDefault);
            hoja1.Range["E5"].Select();//Se selecciona el conjunto
            SistemaOpCeldas.PegaCeldasEsp(null, ref cancelDefault);

            //Se comprueba que el valor de las celdas E5 y F6 son 1 y 4, pero que los
demás siguen estando vacios
            Assert.AreEqual(1, hoja1.Range["E5"].Value2);
            Assert.AreEqual(4, hoja1.Range["F6"].Value2);
            Assert.AreEqual(null, hoja1.Range["E6"].Value2);
            Assert.AreEqual(null, hoja1.Range["F5"].Value2);

            //Se vacian las celdas
            hoja1.Range["C3:F6"].Value2 = "";
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
```

```
        }

        public void CortaPega_UnControladorCopiaMuchasCeldas_SePegaElTexto()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { (Excel.Range)hoja1.Evaluate("A1:C3,D4:F6") });
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            //Se escribe en las celdas
            hoja1.Range["C3"].Value2 = 1;
            hoja1.Range["C4"].Value2 = 2;
            hoja1.Range["D3"].Value2 = 3;
            hoja1.Range["D4"].Value2 = 4;

            //Se realiza la copia y el pegado
            hoja1.Activate();
            hoja1.Range["C3:D4"].Select();//Se selecciona el conjunto
            bool cancelDefault = false;
            SistemaOpCeldas.CortaCeldasEsp(null, ref cancelDefault);

            //Se comprueba que las celdas cortadas están en blanco
            Assert.AreEqual(null, hoja1.Range["C3"].Value2);
            Assert.AreEqual(null, hoja1.Range["D4"].Value2);

            hoja1.Range["E5"].Select();//Se selecciona el conjunto
            SistemaOpCeldas.PegaCeldasEsp(null, ref cancelDefault);

            //Se comprueba que el valor de las celdas E5 y F6 son 1 y 4, pero que los
demás siguen estando vacios
            Assert.AreEqual(1, hoja1.Range["E5"].Value2);
            Assert.AreEqual(4, hoja1.Range["F6"].Value2);
            Assert.AreEqual(null, hoja1.Range["E6"].Value2);
            Assert.AreEqual(null, hoja1.Range["F5"].Value2);

            //Se vacian las celdas
            hoja1.Range["C3:F6"].Value2 = null;
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }

        public void EliminaCeldasEsp_UnControladorEliminaMuchasCeldas_SeEliminan()
        {
            ControladorCopiaTexto controladorCopiaTexto = new ControladorCopiaTexto();
            List<Excel.Range> conjuntoSuscrito = new List<Excel.Range>(new
Excel.Range[] { (Excel.Range)hoja1.Evaluate("A1:C3,D4:F6") });
            SistemaOpCeldas.AddControladorOpCeldas(controladorCopiaTexto,
conjuntoSuscrito);
            //Se escribe en las celdas
            hoja1.Range["C3"].Value2 = 1;
            hoja1.Range["C4"].Value2 = 2;
            hoja1.Range["D3"].Value2 = 3;
            hoja1.Range["D4"].Value2 = 4;

            //Se realiza la copia y el pegado
            hoja1.Activate();
            hoja1.Range["C3:D4"].Select();//Se selecciona el conjunto
            bool cancelDefault = false;
            SistemaOpCeldas.EliminaCeldasEsp(null, ref cancelDefault);

            //Se comprueba que se ha quitado el controlador de estas celdas
```

```
Assert.IsTrue(ExcelTools.AreEqual(SistemaOpCeldas.GetCeldasAsociadas(controladorCopiaT
exto), ExcelTools.Diferencia(conjuntoSuscrito, hoja1.Range["C3:D4"])));

            //Se comprueba que las celdas eliminadas están en blanco
            Assert.AreEqual(null, hoja1.Range["C3"].Value2);
            Assert.AreEqual(null, hoja1.Range["D4"].Value2);
            Assert.AreEqual(2, hoja1.Range["C4"].Value2);
            Assert.AreEqual(3, hoja1.Range["D3"].Value2);

            //Se vacian las celdas
            hoja1.Range["C3:F6"].Value2 = null;
            SistemaOpCeldas.RemoveControladorOpCeldas(controladorCopiaTexto);
        }
```

Pruebas SistemaCompDinamicos

```
        #region clases auxiliares
        public class Wrapper<T> where T : struct
        {
            public T Value { get; set; }
        }
        #endregion

        private static int nEjecuciones = 0;

        private void MetodoInutil(object obj) { }
        private void MetodoPeriodico(object obj)
        {

            nEjecuciones++;
            //Se para el temporizador
            Wrapper<long> idTarea = (Wrapper<long>)obj;
            SistemaCompDinamicos.RemoveTareaPeriodica(idTarea.Value);
        }
        private void DetectaPruebaCorrecta(Object obj)
        {
            Wrapper<long> idTarea = (Wrapper<long>)obj;
            Thread.Sleep(5000);
            Assert.AreEqual(1, nEjecuciones);
            Assert.IsFalse(SistemaCompDinamicos.IsTareaPeriodica(idTarea.Value));
        }
        public void AddYRemoveTareaPeriodica_CreaYElimina_SoloSeEjecuta1Vez()
        {
            Wrapper<long> idTarea = new Wrapper<long>();
            long idRecibido;
            SistemaCompDinamicos.AddTareaPeriodica(MetodoPeriodico, 100, idTarea, out
idRecibido);
            idTarea.Value = idRecibido;

            //Se lanza una hebra en paralelo que comprobará que tras 500 ms solo se ha
ejecutado una vez (y que no está asociada la tarea ya)
            Thread hebra = new Thread(new
ParameterizedThreadStart(DetectaPruebaCorrecta));
            hebra.Start(idTarea);
        }
        public void AddTareaPeriodica_MetodoNull_ExcepcionLanzada()
        {
            long idRecibido;
```

```
            bool excepcionLanzada = false;
            try
            {
                SistemaCompDinamicos.AddTareaPeriodica(null, 100, null, out
idRecibido);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void AddTareaPeriodica_Periodo0_ExcepcionLanzada()
        {
            long idRecibido;
            bool excepcionLanzada = false;
            try
            {
                SistemaCompDinamicos.AddTareaPeriodica(MetodoInutil, 0, null, out
idRecibido);
            }
            catch (ArgumentOutOfRangeException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void AddTareaPeriodica_PeriodoNegativo_ExcepcionLanzada()
        {
            long idRecibido;
            bool excepcionLanzada = false;
            try
            {
                SistemaCompDinamicos.AddTareaPeriodica(MetodoInutil, -1, null, out
idRecibido);
            }
            catch (ArgumentOutOfRangeException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void IsTareaPeriodica_TareaCreaYElimina_DevuelveElValorCorrecto()
        {
            long idRecibido;
            SistemaCompDinamicos.AddTareaPeriodica(MetodoInutil, 200, null, out
idRecibido);
            Assert.IsTrue(SistemaCompDinamicos.IsTareaPeriodica(idRecibido));
            SistemaCompDinamicos.RemoveTareaPeriodica(idRecibido);
            Assert.IsFalse(SistemaCompDinamicos.IsTareaPeriodica(idRecibido));
        }
        public void RemoveTareaPeriodica_TareaEliminada2Veces_NoHayExcepciones()
        {
            long idRecibido;
            SistemaCompDinamicos.AddTareaPeriodica(MetodoInutil, 200, null, out
idRecibido);
            Assert.IsTrue(SistemaCompDinamicos.IsTareaPeriodica(idRecibido));
            SistemaCompDinamicos.RemoveTareaPeriodica(idRecibido);
            Assert.IsFalse(SistemaCompDinamicos.IsTareaPeriodica(idRecibido));
            try
            {
                SistemaCompDinamicos.RemoveTareaPeriodica(idRecibido);
            }
            catch { Assert.Fail(); }
        }
```

```
        #region WorksheetSerializable

        public void WorksheetSerializable_HojaNativa_SerializacionCorrecta()
        {
            WorksheetSerializable worksheet = new WorksheetSerializable(hoja1);
            Assert.AreEqual(hoja1, worksheet.WorksheetRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(worksheet);
            WorksheetSerializable worksheetRecuperado =
HerramientasParaPruebas.FromXmlString<WorksheetSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(worksheetRecuperado);
            Excel.Worksheet worksheetOriginal = worksheet.WorksheetRef;
            Excel.Worksheet worksheet2 = worksheetRecuperado.WorksheetRef;
            Assert.AreEqual(worksheet2, worksheetOriginal);
        }
        public void WorksheetSerializable_HojaHostItem_SerializacionCorrecta()
        {
            WorksheetSerializable worksheet = new
WorksheetSerializable(Globals.Hoja2);
            Assert.AreEqual(hoja2, worksheet.WorksheetRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(worksheet);
            WorksheetSerializable worksheetRecuperado =
HerramientasParaPruebas.FromXmlString<WorksheetSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(worksheetRecuperado);
            Excel.Worksheet worksheetOriginal = worksheet.WorksheetRef;
            Excel.Worksheet worksheet2 = worksheetRecuperado.WorksheetRef;
            Assert.AreEqual(worksheet2, worksheetOriginal);
        }
        public void WorksheetSerializable_ConstructorVacio_SerializacionCorrecta()
        {
            WorksheetSerializable worksheet = new WorksheetSerializable();
            Assert.AreEqual(null, worksheet.WorksheetRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(worksheet);
            WorksheetSerializable worksheetRecuperado =
HerramientasParaPruebas.FromXmlString<WorksheetSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(worksheetRecuperado);
            Excel.Worksheet worksheetOriginal = worksheet.WorksheetRef;
            Excel.Worksheet worksheet2 = worksheetRecuperado.WorksheetRef;
            Assert.AreEqual(worksheet2, worksheetOriginal);
        }
        public void WorksheetSerializable_HojaNativaNull_SerializacionCorrecta()
        {
            WorksheetSerializable worksheet = new
WorksheetSerializable((Excel.Worksheet)null);
            Assert.AreEqual(null, worksheet.WorksheetRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(worksheet);
            WorksheetSerializable worksheetRecuperado =
HerramientasParaPruebas.FromXmlString<WorksheetSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(worksheetRecuperado);
            Excel.Worksheet worksheetOriginal = worksheet.WorksheetRef;
            Excel.Worksheet worksheet2 = worksheetRecuperado.WorksheetRef;
            Assert.AreEqual(worksheet2, worksheetOriginal);
        }
        public void WorksheetSerializable_HojaHostItemNull_SerializacionCorrecta()
        {
            WorksheetSerializable worksheet = new
WorksheetSerializable((WorksheetBase)null);
```

```
            Assert.AreEqual(null, worksheet.WorksheetRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(worksheet);
            WorksheetSerializable worksheetRecuperado =
HerramientasParaPruebas.FromXmlString<WorksheetSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(worksheetRecuperado);
            Excel.Worksheet worksheetOriginal = worksheet.WorksheetRef;
            Excel.Worksheet worksheet2 = worksheetRecuperado.WorksheetRef;
            Assert.AreEqual(worksheet2, worksheetOriginal);
        }

        #endregion

        #region RangeSerializable

        public void RangeSerializable_RangeNormal_SerializacionCorrecta()
        {
            Excel.Range celdas = (Excel.Range)hoja1.Evaluate("B1:D4,J17");
            RangeSerializable range = new RangeSerializable(celdas);
            Assert.IsTrue(ExcelTools.AreEqual(celdas, range.RangeRef));
            string claseSerializada = HerramientasParaPruebas.ToXmlString(range);
            RangeSerializable rangeRecuperado =
HerramientasParaPruebas.FromXmlString<RangeSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(rangeRecuperado);
            Excel.Range rangeOriginal = range.RangeRef;
            Excel.Range range2 = rangeRecuperado.RangeRef;
            Assert.IsTrue(ExcelTools.AreEqual(range2, rangeOriginal));
        }
        public void RangeSerializable_ConstructorVacio_SerializacionCorrecta()
        {
            RangeSerializable range = new RangeSerializable();
            Assert.IsTrue(ExcelTools.AreEqual((Excel.Range)null, range.RangeRef));
            string claseSerializada = HerramientasParaPruebas.ToXmlString(range);
            RangeSerializable rangeRecuperado =
HerramientasParaPruebas.FromXmlString<RangeSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(rangeRecuperado);
            Excel.Range rangeOriginal = range.RangeRef;
            Excel.Range range2 = rangeRecuperado.RangeRef;
            Assert.IsTrue(ExcelTools.AreEqual(range2, rangeOriginal));
        }
        public void RangeSerializable_RangeNull_SerializacionCorrecta()
        {
            RangeSerializable range = new RangeSerializable((Excel.Range)null);
            Assert.IsTrue(ExcelTools.AreEqual((Excel.Range)null, range.RangeRef));
            string claseSerializada = HerramientasParaPruebas.ToXmlString(range);
            RangeSerializable rangeRecuperado =
HerramientasParaPruebas.FromXmlString<RangeSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(rangeRecuperado);
            Excel.Range rangeOriginal = range.RangeRef;
            Excel.Range range2 = rangeRecuperado.RangeRef;
            Assert.IsTrue(ExcelTools.AreEqual(range2, rangeOriginal));
        }
        public void RangeSerializable_CierreDeHoja_WorksheetValidoTrasCierre()
        {
            Excel.Worksheet nuevaHoja =
SimuGlobals.Instance.WorkbookExcel.Worksheets.Add();
            RangeSerializable range = new RangeSerializable(nuevaHoja.Range["D7"]);
            //Se elimina la hoja y se comprueba que el worksheet sigue siendo
accesible aunque la hoja haya sido eliminada
```

```
            nuevaHoja.Application.DisplayAlerts = false;
            nuevaHoja.Delete();
            try
            {
                Assert.AreEqual(nuevaHoja, range.WorksheetRef);
            }
            catch { Assert.Fail(); }
            bool excepcionLanzada = false;
            try
            {
                string unNombre = range.RangeRef.Name;
            }
            catch { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }

        #endregion

        #region MismasCeldasComparer

        public void MismasCeldasComparer_HashSet2Range_ContainsFuncionaCorrectamente()
        {
            HashSet<RangeSerializable> hashSet = new HashSet<RangeSerializable>(new
MismasCeldasComparer());
            hashSet.Add(new RangeSerializable(hoja1.Range["A1:D1"]));
            hashSet.Add(new RangeSerializable(hoja1.Range["B2:F20"]));
            Assert.IsTrue(hashSet.Contains(new
RangeSerializable((Excel.Range)hoja1.Evaluate("A1,B1,C1,D1"))));
            Assert.IsFalse(hashSet.Contains(new
RangeSerializable(hoja1.Range["B2:F4"])));
            Assert.IsFalse(hashSet.Contains(new
RangeSerializable((Excel.Range)hoja2.Evaluate("A1,B1,C1,D1"))));
        }

        #endregion

        #region GraficoHojaSerializable

        public void
GraficoHojaSerializable_HojaDeGraficoNativa_SerializacionCorrecta()
        {
            GraficoHojaSerializable chart = new
GraficoHojaSerializable(hojaDeGrafico);
            Assert.AreEqual(hojaDeGrafico, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoHojaSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoHojaSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.Chart chartOriginal = chart.ChartRef;
            Excel.Chart chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoHojaSerializable_HojaDeGraficoHostItem_SerializacionCorrecta()
        {
            GraficoHojaSerializable chart = new
GraficoHojaSerializable(Globals.Gráfico1);
            Assert.AreEqual(hojaDeGrafico, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoHojaSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoHojaSerializable>(claseSerializada);
```

```
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.Chart chartOriginal = chart.ChartRef;
            Excel.Chart chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void GraficoHojaSerializable_ConstructorVacio_SerializacionCorrecta()
        {
            GraficoHojaSerializable chart = new GraficoHojaSerializable();
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoHojaSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoHojaSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.Chart chartOriginal = chart.ChartRef;
            Excel.Chart chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoHojaSerializable_HojaDeGraficoNativaNull_SerializacionCorrecta()
        {
            GraficoHojaSerializable chart = new
GraficoHojaSerializable((Excel.Chart)null);
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoHojaSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoHojaSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.Chart chartOriginal = chart.ChartRef;
            Excel.Chart chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoHojaSerializable_HojaDeGraficoHostItemNull_SerializacionCorrecta()
        {
            GraficoHojaSerializable chart = new
GraficoHojaSerializable((ChartSheetBase)null);
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoHojaSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoHojaSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.Chart chartOriginal = chart.ChartRef;
            Excel.Chart chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }

        #endregion

        #region GraficoFlotanteSerializable

        public void
GraficoFlotanteSerializable_GraficoFlotanteNativo_SerializacionCorrecta()
        {
            GraficoFlotanteSerializable chart = new
GraficoFlotanteSerializable(graficoHoja1);
            Assert.AreEqual(graficoHoja1.Name, chart.ChartRef.Name);
            Assert.AreEqual(graficoHoja1.TopLeftCell.Worksheet,
chart.ChartRef.TopLeftCell.Worksheet);
```

```
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoFlotanteSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoFlotanteSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.ChartObject chartOriginal = chart.ChartRef;
            Excel.ChartObject chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2.Name, chartOriginal.Name);
            Assert.AreEqual(chart2.TopLeftCell.Worksheet,
chartOriginal.TopLeftCell.Worksheet);
        }
        public void
GraficoFlotanteSerializable_GraficoFlotanteHostItem_SerializacionCorrecta()
        {
            GraficoFlotanteSerializable chart = new
GraficoFlotanteSerializable(Globals.Hoja1.Chart_1);
            Assert.AreEqual(graficoHoja1.Name, chart.ChartRef.Name);
            Assert.AreEqual(graficoHoja1.TopLeftCell.Worksheet,
chart.ChartRef.TopLeftCell.Worksheet);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoFlotanteSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoFlotanteSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.ChartObject chartOriginal = chart.ChartRef;
            Excel.ChartObject chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2.Name, chartOriginal.Name);
            Assert.AreEqual(chart2.TopLeftCell.Worksheet,
chartOriginal.TopLeftCell.Worksheet);
        }
        public void
GraficoFlotanteSerializable_ConstructorVacio_SerializacionCorrecta()
        {
            GraficoFlotanteSerializable chart = new GraficoFlotanteSerializable();
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoFlotanteSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoFlotanteSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.ChartObject chartOriginal = chart.ChartRef;
            Excel.ChartObject chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoFlotanteSerializable_GraficoFlotanteNativoNull_SerializacionCorrecta()
        {
            GraficoFlotanteSerializable chart = new
GraficoFlotanteSerializable((Excel.ChartObject)null);
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoFlotanteSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoFlotanteSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.ChartObject chartOriginal = chart.ChartRef;
            Excel.ChartObject chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoFlotanteSerializable_GraficoFlotanteHostItemNull_SerializacionCorrecta()
        {
```

```
            GraficoFlotanteSerializable chart = new
GraficoFlotanteSerializable((Chart)null);
            Assert.AreEqual(null, chart.ChartRef);
            string claseSerializada = HerramientasParaPruebas.ToXmlString(chart);
            GraficoFlotanteSerializable chartRecuperado =
HerramientasParaPruebas.FromXmlString<GraficoFlotanteSerializable>(claseSerializada);
            //Se comparan los 2 objetos
            Assert.IsNotNull(chartRecuperado);
            Excel.ChartObject chartOriginal = chart.ChartRef;
            Excel.ChartObject chart2 = chartRecuperado.ChartRef;
            Assert.AreEqual(chart2, chartOriginal);
        }
        public void
GraficoFlotanteSerializable_CierreDeHoja_WorksheetValidoTrasCierre()
        {
            Excel.Worksheet nuevaHoja =
SimuGlobals.Instance.WorkbookExcel.Worksheets.Add();
            Excel.ChartObjects graficas = nuevaHoja.ChartObjects();
            Excel.ChartObject nuevaGrafica = graficas.Add(0, 0, 200, 200);
            GraficoFlotanteSerializable graficoFlotante = new
GraficoFlotanteSerializable(nuevaGrafica);
            //Se elimina la hoja y se comprueba que el worksheet sigue siendo
accesible aunque la hoja haya sido eliminada
            nuevaHoja.Application.DisplayAlerts = false;
            nuevaHoja.Delete();
            try
            {
                Assert.AreEqual(nuevaHoja, graficoFlotante.WorksheetRef);
            }
            catch { Assert.Fail(); }
            bool excepcionLanzada = false;
            try
            {
                string unNombre = graficoFlotante.ChartRef.Name;
            }
            catch { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }

        #endregion
```

Pruebas gestor de elementos

```
        public class ClaseInicializable : IInitializable
        {
            public int contador = new Random().Next(200);
            public bool initializeLlamado = false;
            public bool terminateLlamado = false;

            public ClaseInicializable() { }
            public ClaseInicializable(int unContador) { contador = unContador; }
            public void Initialize()
            {
                initializeLlamado = true;
            }

            public void Terminate()
            {
                terminateLlamado = true;
            }
        }
```

```
        public class ContadorComparer : IEqualityComparer<ClaseInicializable>
        {
            public int entero = 0;
            public ContadorComparer() { }
            public ContadorComparer(int unEntero) { entero = unEntero; }

            public bool Equals(ClaseInicializable x, ClaseInicializable y)
            {
                if (x == null || y == null) return x == y;
                return x.contador == y.contador;
            }

            public int GetHashCode(ClaseInicializable obj)
            {
                if (obj == null) return 0;
                return obj.contador;
            }
        }

        public void
GestorDeElementos_GestorConElementosConComparer_SeSerializaCorrectamente()
        {
            GestorDeElementos<ClaseInicializable, ContadorComparer> gestorDeElementos
= new GestorDeElementos<ClaseInicializable, ContadorComparer>(new
ContadorComparer(200));
            gestorDeElementos.AddElemento(new ClaseInicializable(132));
            string claseSerializada =
HerramientasParaPruebas.ToXmlString(gestorDeElementos);
            GestorDeElementos<ClaseInicializable, ContadorComparer> gestorRecuperado =
HerramientasParaPruebas.FromXmlString<GestorDeElementos<ClaseInicializable,
ContadorComparer>>(claseSerializada);

Assert.IsTrue(gestorDeElementos.Elementos.SequenceEqual(gestorRecuperado.Elementos,
new ContadorComparer()));
            Assert.IsNotNull(gestorRecuperado.Elementos.Comparer);

Assert.AreEqual(((ContadorComparer)gestorDeElementos.Elementos.Comparer).entero,
((ContadorComparer)gestorRecuperado.Elementos.Comparer).entero);
        }
        public void
GestorDeElementos_GestorConElementosSinComparer_SeSerializaCorrectamente()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            gestorDeElementos.AddElemento(new ClaseInicializable(215));
            string claseSerializada =
HerramientasParaPruebas.ToXmlString(gestorDeElementos);
            GestorDeElementos<ClaseInicializable> gestorRecuperado =
HerramientasParaPruebas.FromXmlString<GestorDeElementos<ClaseInicializable>>(claseSeri
alizada);

Assert.IsTrue(gestorDeElementos.Elementos.SequenceEqual(gestorRecuperado.Elementos,
new ContadorComparer()));
        }

        public void Initialize_GestorConElementos_SeInicializan()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            gestorDeElementos.AddElemento(new ClaseInicializable());
            gestorDeElementos.AddElemento(new ClaseInicializable());
            gestorDeElementos.Initialize();
```

```
            foreach (ClaseInicializable elemento in gestorDeElementos.Elementos)
            {
                Assert.IsTrue(elemento.initializeLlamado);
            }
        }

        public void Terminate_GestorConElementos_SeLlamaATerminate()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            gestorDeElementos.AddElemento(new ClaseInicializable());
            gestorDeElementos.AddElemento(new ClaseInicializable());
            gestorDeElementos.Terminate();
            foreach (ClaseInicializable elemento in gestorDeElementos.Elementos)
            {
                Assert.IsTrue(elemento.terminateLlamado);
            }
        }

        public void AddElemento_ElementoNormal_QuedaIncluido()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            ClaseInicializable objetoInicializable = new ClaseInicializable();
            gestorDeElementos.AddElemento(objetoInicializable);
            Assert.IsTrue(gestorDeElementos.Elementos.Contains(objetoInicializable));
        }
        public void AddElemento_ElementoNull_ExcepcionLanzada()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            bool excepcionLanzada = false;
            try
            {
                gestorDeElementos.AddElemento(null);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void RemoveElemento_ElementoNormal_SeElimina()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            ClaseInicializable objetoInicializable = new ClaseInicializable();
            gestorDeElementos.AddElemento(objetoInicializable);
            Assert.IsTrue(gestorDeElementos.Elementos.Contains(objetoInicializable));
            gestorDeElementos.RemoveElemento(objetoInicializable);
            Assert.IsFalse(gestorDeElementos.Elementos.Contains(objetoInicializable));
        }
        public void RemoveElemento_ElementoNull_ExcepcionLanzada()
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            bool excepcionLanzada = false;
            try
            {
                gestorDeElementos.RemoveElemento(null);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void RemoveElemento_GestorConElementos_SeVacia()
```

```
        {
            GestorDeElementos<ClaseInicializable> gestorDeElementos = new
GestorDeElementos<ClaseInicializable>();
            ClaseInicializable objetoInicializable = new ClaseInicializable();
            gestorDeElementos.AddElemento(objetoInicializable);
            gestorDeElementos.AddElemento(new ClaseInicializable());
            gestorDeElementos.ClearElementos();
            Assert.AreEqual(0, gestorDeElementos.Elementos.Count);
        }
```

Pruebas ThisWorkbook

```
        private static bool eventoHojaCerradaSalto = false;
        private static Excel.Worksheet hojaCerrada;
        public void SheetClosed_SeCierraUnaHoja_ElEventoSalta()
        {
            SimuGlobals.Instance.WorkbookExcel.SheetClosed += Instance_SheetClosed;

            Excel.Worksheet nuevaHoja =
SimuGlobals.Instance.WorkbookExcel.Worksheets.Add();
            hojaCerrada = nuevaHoja;

            nuevaHoja.Application.DisplayAlerts = false;
            nuevaHoja.Delete();

            Assert.IsTrue(eventoHojaCerradaSalto);
            eventoHojaCerradaSalto = false;

            SimuGlobals.Instance.WorkbookExcel.SheetClosed -= Instance_SheetClosed;
        }

        public void Instance_SheetClosed(object sender, SheetClosedEventArgs e)
        {
            eventoHojaCerradaSalto = true;
            Assert.AreEqual(1, e.HojasCerradas.Count);
            Assert.IsTrue(e.HojasCerradas.First().Equals(hojaCerrada));
        }
```

Pruebas de PropiedadesDeConexion

```
        public void DefaultClient_Normal_ValorCorrecto()
        {
            PropiedadesDeConexion propiedadesDeConexion =
PropiedadesDeConexion.ClientDefault;
            Assert.IsTrue(propiedadesDeConexion.IgnoraPropiedadesDeConexion);
        }

        public void PropiedadesDeConexion_SinArgumentos_CreacionCorrecta()
        {
            PropiedadesDeConexion propiedadesDeConexion = new PropiedadesDeConexion();
            Assert.IsTrue(propiedadesDeConexion.IgnoraPropiedadesDeConexion);
            SimulatorServiceClient cliente =
propiedadesDeConexion.CreaClienteWCF<SimulatorServiceClient>();
            SimulatorServiceClient cliente2 = new SimulatorServiceClient();
            Assert.AreEqual(cliente2.Endpoint.Name, cliente.Endpoint.Name);
            Assert.AreEqual(cliente2.Endpoint.Address.Uri,
cliente.Endpoint.Address.Uri);
        }
```

```
        public void PropiedadesDeConexion_Copia_CreacionCorrecta()
        {
            PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion("BasicHttpBinding_ISimulatorService1");
            Assert.AreEqual(propiedadesDeConexion, new
PropiedadesDeConexion(propiedadesDeConexion));
        }
        public void PropiedadesDeConexion_CopiaNull_ExcepcionLanzada()
        {
            bool excepcionLanzada = true;
            try{
                PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion((PropiedadesDeConexion)null);
            }catch(ArgumentNullException){excepcionLanzada = true;}
            Assert.IsTrue(excepcionLanzada);
        }

        public void
PropiedadesDeConexion_EndpointConfigName_CreacionYClienteCorrectos()
        {
            PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion("BasicHttpBinding_ISimulatorService1");
            Assert.AreEqual("BasicHttpBinding_ISimulatorService1",
propiedadesDeConexion.EndpointConfigurationName);
            SimulatorServiceClient cliente =
propiedadesDeConexion.CreaClienteWCF<SimulatorServiceClient>();
            Assert.AreEqual("BasicHttpBinding_ISimulatorService1",
cliente.Endpoint.Name);
        }
        public void
PropiedadesDeConexion_EndpointConfigName_CreacionCorrectosClienteExcepcion()
        {
            PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion("BasicHttpBinding_ISimulatorService2");
            Assert.AreEqual("BasicHttpBinding_ISimulatorService2",
propiedadesDeConexion.EndpointConfigurationName);

            bool excepcionLanzada = true;
            try{
                SimulatorServiceClient cliente =
propiedadesDeConexion.CreaClienteWCF<SimulatorServiceClient>();
            }
            catch (TargetInvocationException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
        public void PropiedadesDeConexion_EndpointConfigNameNull_ExcepcionLanzada()
        {
            bool excepcionLanzada = true;
            try
            {
                PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion((String)null);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
```

```
        public void
PropiedadesDeConexion_EndpointConfigNameAddress_CreacionYClienteCorrectos()
        {
            PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion("BasicHttpBinding_ISimulatorService1",
"http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/SimulatorService/");
            Assert.AreEqual("BasicHttpBinding_ISimulatorService1",
propiedadesDeConexion.EndpointConfigurationName);

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/Simulat
orService/", propiedadesDeConexion.RemoteAddress.Uri.AbsoluteUri);
            SimulatorServiceClient cliente =
propiedadesDeConexion.CreaClienteWCF<SimulatorServiceClient>();
            Assert.AreEqual("BasicHttpBinding_ISimulatorService1",
cliente.Endpoint.Name);

Assert.AreEqual("http://localhost:8733/Design_Time_Addresses/SimuladorEjemplo2/Simulat
orService/", cliente.Endpoint.Address.Uri.AbsoluteUri);
        }
        public void
PropiedadesDeConexion_EndpointConfigNameAddressNull_ExcepcionLanzada()
        {
            bool excepcionLanzada = true;
            try
            {
                PropiedadesDeConexion propiedadesDeConexion = new
PropiedadesDeConexion("BasicHttpBinding_ISimulatorService1", (string)null);
            }
            catch (ArgumentNullException) { excepcionLanzada = true; }
            Assert.IsTrue(excepcionLanzada);
        }
```

Pruebas de MonitorDeConexiones

```
        public void AddMonitorizacionDeEstado_ClienteValido_DevuelveValorCorrecto()
        {
            SimulatorServiceClient cliente = new SimulatorServiceClient();
            MonitorDeConexiones.AddMonitorizacionDeEstado(cliente);
            Assert.IsTrue(MonitorDeConexiones.EstaSiendoMonitorizada(cliente));
            MonitorDeConexiones.RemoveMonitorizacionDeEstado(cliente);
        }
        public void HayDatosDeEstado_ClienteValido_DevuelveValorCorrecto()
        {
            SimulatorServiceClient cliente = new SimulatorServiceClient();
            MonitorDeConexiones.AddMonitorizacionDeEstado(cliente);
            Assert.IsFalse(MonitorDeConexiones.HayDatosDeEstado(cliente));
            Thread.Sleep(300);
            Assert.IsTrue(MonitorDeConexiones.HayDatosDeEstado(cliente));
            bool esAccesible;
            MonitorDeConexiones.GetEstadoConexion(cliente, out esAccesible);
            Assert.IsTrue(esAccesible);
            MonitorDeConexiones.RemoveMonitorizacionDeEstado(cliente);
        }
        public void
RemoveMonitorizacionDeEstado_ClienteValido2Veces_DevuelveValorCorrecto()
        {
            SimulatorServiceClient cliente = new SimulatorServiceClient();
            MonitorDeConexiones.AddMonitorizacionDeEstado(cliente);
            MonitorDeConexiones.AddMonitorizacionDeEstado(cliente);
            Assert.IsTrue(MonitorDeConexiones.EstaSiendoMonitorizada(cliente));
            MonitorDeConexiones.RemoveMonitorizacionDeEstado(cliente);
            Assert.IsTrue(MonitorDeConexiones.EstaSiendoMonitorizada(cliente));
```

```
    MonitorDeConexiones.RemoveMonitorizacionDeEstado(cliente);
    Assert.IsFalse(MonitorDeConexiones.EstaSiendoMonitorizada(cliente));
}
public void GetEstadoConexion_ClienteValido_DevuelveValorCorrecto()
{
    SimulatorServiceClient cliente = new SimulatorServiceClient();
    MonitorDeConexiones.AddMonitorizacionDeEstado(cliente);
    bool esAccesible;
    MonitorDeConexiones.GetEstadoConexion(cliente, out esAccesible);
    Assert.IsTrue(esAccesible);
    MonitorDeConexiones.RemoveMonitorizacionDeEstado(cliente);
}
public void IntentaAccesoAConexion_ClienteValido_DevuelveValorCorrecto()
{
    SimulatorServiceClient cliente = new SimulatorServiceClient();
    bool esAccesible;
    MonitorDeConexiones.IntentaAccesoAConexion(cliente, out esAccesible);
    Assert.IsTrue(esAccesible);
}
```

3.2. Pruebas manuales

Algunas de las características del proyecto era imposible probarlas por código. Para estos casos, se han realizado pruebas manuales.

Cambio de modo

Las siguientes pruebas comprueban que al cambiar los valores del archivo parámetros generales relativos al modo se producen los cambios correspondientes en las interfaces y en el funcionamiento del sistema.

Prueba 1

1. El tester cambia el campo PasswordEdicion del archivo ParametrosGenerales al valor "PasswordPruebas" y el campo RequierePasswordEdicion a true.
2. El tester arranca la aplicación.
3. El tester se coloca en modo ejecución. Si el sistema comienza en modo edición, debe pulsar el botón de cambio de modo una vez, sino, se salta directamente al paso 4.
4. El tester pulsa en el botón Ribbon de cambio de modo.
5. El sistema muestra un panel pidiendo la contraseña.
6. El tester introduce la contraseña correcta.
7. El sistema cambia de modo a modo edición.

Prueba 2

Los pasos 1 al 5 son iguales que los de Prueba1.

6. El tester introduce una contraseña incorrecta.
7. El sistema muestra un panel informando de que la contraseña es incorrecta.
8. El tester pulsa aceptar.
9. El sistema cierra la ventana.

Prueba 3

1. El tester cambia el campo PasswordEdicion el campo RequierePasswordEdicion a false.
2. El tester arranca la aplicación.
3. El tester se coloca en modo ejecución. Si el sistema comienza en modo edición, debe pulsar el botón de cambio de modo una vez, sino, se salta directamente al paso 4.
4. El tester pulsa en el botón Ribbon de cambio de modo.
5. El sistema cambia de modo a modo edición.

Nombre aplicación en ActionsPane

La siguiente prueba comprueba que al cambiar los valores del archivo parámetros generales relativos al nombre de la aplicación se producen los cambios correspondientes en las interfaces y en el funcionamiento del sistema.

Prueba 1

1. El tester cambia el campo NombreAplicacion del archivo ParametrosGenerales al valor "Aplicación de prueba".
2. El tester arranca la aplicación.
3. El sistema muestra en el actions panel el nombre especificado en el archivo de ParametrosGenerales.

Operaciones especiales sobre celdas

Con las siguientes pruebas se comprueba que el sistema de operaciones especiales sobre celdas funciona correctamente. Para poder probarlo, se incluye en el proyecto de pruebas un control que permite realizar una monitorización de un variable en una celda.

Prueba 1

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona el conjunto B2:D5 y pulsa en el botón de copiar celdas en el panel contextual que aparece al pulsar click derecho.
5. El tester selecciona la casilla F10 y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
6. El sistema pone un 0 en las celdas F10 y G11.

Prueba 2

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona el conjunto B2:D5 y pulsa en el botón de cortar celdas en el panel contextual que aparece al pulsar click derecho.
5. El sistema borra el contenido de las celdas B2 y C3.
6. El tester selecciona la casilla F10 y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
7. El sistema pone un 0 en las celdas F10 y G11.

Prueba 3

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona el conjunto B2:D5 y pulsa en el botón de eliminar celdas en el panel contextual que aparece al pulsar click derecho.
5. El sistema borra el contenido de las celdas B2 y C3.

Prueba 4

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona la hoja y pulsa en el botón de copiar celdas en el panel contextual que aparece al pulsar click derecho.
5. El tester selecciona otra hoja distinta y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
6. El sistema pone un 0 en las celdas B2 y C3 de la hoja en la que se pulso pegar.

Prueba 5

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.

4. El tester selecciona la columna B y pulsa en el botón de copiar celdas en el panel contextual que aparece al pulsar click derecho.
5. El tester selecciona la casilla C4 y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
6. El sistema muestra un mensaje de error informando que la forma de los conjuntos de origen y de destino no coinciden, por lo que es imposible realizar el pegado.

Prueba 6

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona la fila 2 y pulsa en el botón de copiar celdas en el panel contextual que aparece al pulsar click derecho.
5. El tester selecciona la casilla C4 y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
6. El sistema muestra un mensaje de error informando que la forma de los conjuntos de origen y de destino no coinciden, por lo que es imposible realizar el pegado.

Prueba 7

1. El tester arranca la aplicación y se coloca en modo edición.
2. El tester añade monitorizaciones a las celdas B2 y C3.
3. El sistema añade un 0 en ambas celdas.
4. El tester selecciona la hoja y pulsa en el botón de copiar celdas en el panel contextual que aparece al pulsar click derecho.
5. El tester selecciona la casilla C4 y pulsa en el botón de pegar celdas en el panel contextual que aparece al pulsar click derecho.
6. El sistema muestra un mensaje de error informando que la forma de los conjuntos de origen y de destino no coinciden, por lo que es imposible realizar el pegado.

Mostrar y ocultar panel

Con la siguiente prueba se comprueba que la funcionalidad de ocultar y mostrar el actions pane funciona correctamente.

1. El tester arranca la aplicación.
2. El tester pulsa el botón de mostrar/ocultar panel.
3. En caso de que el panel estuviese oculto, se muestra. En caso de que estuviese visible, se oculta.
4. El tester pulsa el botón de mostrar/ocultar panel.
5. En caso de que el panel estuviese oculto, se muestra. En caso de que estuviese visible, se oculta.

Panel de estado de los simuladores

Para estas pruebas se incluyen 1 simulador de ejemplo y se añade al panel. Además, se le asocia un controlador para que se muestren los estados de corriendo o parado. Finalmente, se añade un botón de arranque para poder lanzarlo a correr.

Con estas pruebas se comprueba que realmente el panel muestra el estado correcto de los simuladores.

Prueba 1

1. El tester arranca la aplicación.
2. El tester se coloca en modo edición y añade el botón al documento (para poder pulsarlo).
3. El tester se coloca en modo ejecución.
4. El sistema muestra el estado como parado.
5. El tester pulsa el botón.
6. El sistema muestra el estado como corriendo.
7. El tester se coloca en modo edición.
8. El sistema muestra el estado de corriendo en gris.

Prueba 2

1. El tester arranca la aplicación.
2. El tester se coloca en modo ejecución.
3. El tester apaga el servicio.
4. El sistema muestra el estado como no accesible.
5. El tester intenta pulsar casillas y lo consigue sin ningún problema (el programa no está atascado).