INTERFAZ GRÁFICA PARA UNA PLATAFORMA DE SIMULADORES DE PROCESOS DE USO ACADÉMICO

Anexo 4: Documentación Técnica de Programación

INGENIERÍA INFORMÁTICA

Septiembre de 2023

ALUMNO:

ÁLVARO GARCÍA LABRADOR

70913088V

TUTORES

MARIO FRANCISCO SUTIL

PASTORA ISABEL VEGA CRUZ

Índice

Ilustraciones

1. Introducción

Este anexo está centrado en los aspectos técnicos del desarrollo del programa. Aquí se mostrará el código fuente de la aplicación desarrollada, así como la explicación de su estructura y funcionamiento, un manual para el programador y las pruebas unitarias que se realizaron para el continuo progreso del sistema.

2. Código Fuente

En este apartado se mostrará el código fuente desarrollado para el correcto funcionamiento de la aplicación. En primera instancia se comentará la estructura de carpetas que contienen los ficheros necesarios para el funcionamiento.

Al tratarse de una aplicación desarrollada con el App Designer de MATLAB y usando como entorno de ejecución de la aplicación el propio MATLAB, todos los ficheros deben estar en la misma carpeta siendo así la forma de trabajar que tiene MATLAB.

La carpeta principal se llama ArchivosSim esta carpeta contendrá BSM1_Alvaro que será la carpeta dónde se guardan los ficheros, imágenes, documentos, etc.

Dentro de esta última carpeta se encuentran los archivos más importantes en lo que respecta al desarrollo del código fuente, estos son: interfazGráfica.mlapp, Tanques_alvaro.slx y benchmark.mdl

- interfazGráfica.mlapp: aquí es dónde se encuentra la mayor parte del código fuente desarrollado, que está dividido en Funciones y Callbacks.
- Tanques_alvaro.slx y benchmark.mdl: en estos archivos de Simulink se han simplificado los esquemas de ambos modelos creando subsistemas y se han desarrollado interfaces que permitan modificar variables para cada uno de estos subsistemas, esto se ha desarrollado gracias a la herramienta proporcionada por Simulink llamada *Mask*.

Por ende, el apartado de código fuente se ha divido en: Funciones, Callbacks y Masks.

2.1. Funciones

En el caso de las funciones desarrolladas, nos encontramos con estas:

Figura 1 Funciones

2.1.1. visiblesVentanas

El primer tipo de funciones desarrolladas son las de cambio de pantalla, que permiten la transición de una ventana a otra de forma sencilla y eficaz. Lo que hace la función es poner en visible la ventana que te interesa que se muestre y poner en invisible todas las demas.

```
function visiblePrincipal(app)
    set(app.Principal, 'Visible', 'on');
    set(app.Basicos, 'Visible', 'off');
    set(app.Avanzados,'Visible','off');
    set(app.Graficas,'Visible','off');
    set(app.Informes,'Visible','off');
    set(app.Comparar,'Visible','off');
end
```

Figura 2 visiblesVentanas

Ej:

En este caso *visiblePrincipal* lo que se hace es ocultar todas las ventanas excepto la ventana *Principal*, de esta forma se mostrará sólo esta.

2.1.2. mostrarGrafica

Esta función permitirá al usuario mostrar por pantalla la gráfica que haya seleccionado en la listBox determinada.
La función toma la selecciona a evaluar el valor de la listBox que se ha seleccionado, que en este caso será el nombre de la gráfica, de esta manera evaluará el nombre con el swicth, el caso que concuerde con el nombre ejecutará la figura en cuestión y esta se mostrará en una ventana emergente al usuario.
Existe una llamada a la función c*ompararGráficas(app)* que se utilizará cuando se hacen las comparación de varias simulaciones en la gráfica seleccionada.
En la imagen a continuación se puede apreciar la función:

```
function mostrarGrafica(app,listBox)
    seleccion=listBox.Value;
    switch seleccion
        case 'SI_Concentración de materia orgánica inerte soluble (g DQO/m3)'
            evalin('base', ['run ' 'SIplot.m']);
            compararGraficas(app);
        case 'SS_Concentración de sustrato fácilmente biodegradable (g DQO/m3)'
            evalin('base', ['run ' 'SSplot.m']);
            compararGraficas(app);
        case 'XI_Concentración de materia orgánica inerte insoluble (g DQO/m3)'
            evalin('base', ['run ' 'XIplot.m']);
            compararGraficas(app);
        case 'XS_Concentración de sustrato de degradación lenta (g DQO/m3)'
            evalin('base',['run ' 'XSplot.m']);
            compararGraficas(app);
        case 'XBH_Concentración de biomasa heterótrofa activa (g DQO/m3)'
            evalin('base',['run ' 'XBHplot.m']);
            compararGraficas(app);
        case 'XBA_Concentración de biomasa autótrofa activa (g DQO/m3)'
            evalin('base',['run ' 'XBAplot.m']);
            compararGraficas(app);
```

• • •

Figura 3 mostrarGrafica

2.1.3. compararGraficas

La llamada de esta función ya aparece en la anterior y, como se ha descrito brevemente, se trata de la función encargada de añadir los datos de una simulación en la gráfica que se haya seleccionado.

Está diseñada para la comparación de 3 simulaciones de forma que a medida que se vaya llamando a la función con diferentes datos, tomarán los subplots de cada una y en un bucle irá superponiendo cada dato de cada simulación en cada subplot, de forma que al final la figura en cuestión contenta la comparación de tres simulaciones diferentes representadas en una misma gráfica.

En el caso en el que se llame varias veces a la función con los mismos datos, lo único que ocurrirá será que se dibuje una gráfica encima de la otra, se superpongan los mismos datos.

```
function compararGraficas(app)
            % Obtén los subplots de la primera figura generada por figura1.m
            axes_figura1 = findobj(gcf, 'type', 'axes');
            % Obtén los subplots de la segunda figura generada por figura2.m
            axes_figura2 = findobj(gcf, 'type', 'axes');
            % Obtén los subplots de la tercera figura generada por figura3.m
            axes_figura3 = findobj(gcf, 'type', 'axes');

            % Superponer los gráficos de cada subplot en la misma posición
            for i = 1:numel(axes_figura1)
                % Cambia al subplot correspondiente en la primera figura
                subplot(axes_figura1(i));

                % Activa el modo 'hold' en el subplot de la primera figura
                hold on;

                % Obtén los gráficos de los subplots de la segunda figura
                graficos_figura2 = get(axes_figura2(i), 'children');

                % Copia los gráficos de los subplots de la segunda figura al subplot de la primera figura
                copyobj(graficos_figura2, axes_figura1(i));

                % Obtén los gráficos de los subplots de la tercera figura
                graficos_figura3 = get(axes_figura3(i), 'children');

                % Copia los gráficos de los subplots de la tercera figura al subplot de la primera figura
                copyobj(graficos_figura3, axes_figura1(i));
            end
```

Figura 4 compararGraficas

2.2. Callbacks

Los callbacks a diferencia de las funciones explicadas anteriormente, permiten la ejecución asíncrona de código, es decir, ninguna de estas funciones bloquea el flujo de ejecución mientras espera una operación para completarse. En general estas funciones se ejecutan por eventos, cuando un evento ocurre, se ejecuta cierta función definida.

2.2.1. Movimientos de Ventanas

En este apartado se comentarán todos los callbacks relacionados con los movimientos del usuario por las diferentes ventanas de la aplicación.
Los callbacks utilizados son los siguientes:

Figura 5 Callbacks Movimientos de Ventanas

Lo que hacen estos callbacks es llamar la función *visibleVentana*, siendo en cada función la que corresponda con su ventana.
El callback se activará cuando se pulse el botón correspondiente, entonces se ejecutará la función y se redirigirá al usuario a la ventana en cuestión.

Ej:

```
% Button pushed function: ModelosBasicosButton
function modelosBasicos(app, event)
    visibleModelosBasicos(app);
end

% Button pushed function: ModelosAvanzadosButton
function modelosAvanzados(app, event)
    visibleModelosAvanzados(app);
end

% Callback function: Cerrar, cerrarAppButton
function cerrarApp(app, event)
    delete(app);
end
```

Figura 6 Callback Ej

En este caso se pueden ver las funciones de *modelosBasicos*, *modelosAvanzados* y *cerrarApp*.
Las dos primeras permitirán al usuario que al pulsar el botón correspondiente le mostrará la ventana de modelos básicos o la de modelos avanzados, así es como funcionan todos los demás callbacks de este estilo.
La función de *cerrarApp* lo que hace es borrar la app, en consecuencia, cerrar la aplicación.

2.2.2. Gestionar Circuito

A continuación, se explicarán todos los callbacks relacionados con la pestaña del Circuito RC. Las funciones son las siguientes:

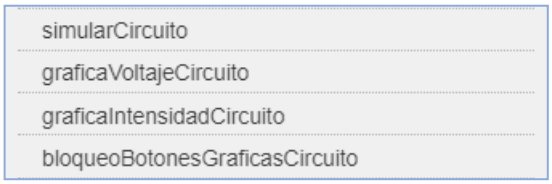

Figura 7 Callbacks Gestionar Circuito

- *simularCircuito*: esta es la función encargada de tomar los tiempos de simulación y el voltaje inicial del condensador escritos por el usuario y añadirlos a la base de Matlab, pues serán esos valores con los que se ejecutará la simulación.

Después ejecuta el archivo *circuitoRCprincipal.m* y activa los botones de voltaje e intensidad para que el usuario pueda visualizar los resultados.

```
% Button pushed function: SimularCircuitoButton
function simularCircuito(app, event)
    app.IntensidadButton.Value=false;
    app.VoltajeButton.Value=false;
    tsim=[app.tinicialCircuito.Value, app.tfinalCircuito.Value];
    assignin('base','tsim', tsim);
    assignin('base','v0',app.v0Circuito.Value);
    evalin('base', ['run ' 'circuitoRCprincipal']);
    cla(app.UIAxes);
    app.UIAxes.Title.String='';
    app.VoltajeButton.Enable='on';
    app.IntensidadButton.Enable='on';
end
```

Figura 8 simularCircuito

- *graficaVoltajeCircuito* y *graficaIntensidadCircuito*: estas dos funciones son las encargadas de dibujar los resultados de la simulación en la gráfica de la ventana cuando se pulse su botón correspondiente.
  Se toman los valores que se van a representar en la gráfica, se escribe el nombre de la gráfica, de los valores y se hace el plot con los valores correspondientes en la gráfica en cuestión: *app.UIAxes*.

```
% Value changed function: VoltajeButton
function graficaVoltajeCircuito(app, event)
    % Obtener los valores ingresados por el usuario de los edit text
    t = evalin('base', 't');
    v = evalin('base', 'v');
    app.UIAxes.Title.String='Gráfica Voltaje';
    app.UIAxes.YLabel.String='voltaje';
    plot(app.UIAxes, t,v,'R');
    grid(app.UIAxes, 'on');
    app.IntensidadButton.Value=false;
end

% Value changed function: IntensidadButton
function graficaIntensidadCircuito(app, event)
    t = evalin('base', 't');
    I = evalin('base', 'I');
    app.UIAxes.Title.String='Gráfica Intensidad';
    app.UIAxes.YLabel.String='intensidad';
    plot(app.UIAxes, t,I);
    grid(app.UIAxes, 'on');
    app.VoltajeButton.Value=false;
end
```

Figura 9 graficaVoltaje y graficaIntensidad

- *bloqueoBotonesGraficasCircuito*: esta función está escrita para los cambios de valores, es decir, cuando se cambia el tiempo de simulación o el voltaje inicial del condensador de los campos editables, entonces, se ejecutará esta función para que el usuario deba simular de nuevo si quiere ver los nuevos resultados.
  Se puede observar cómo se desactivan los botones de voltaje e intensidad para que el usuario simule de nuevo.

```
% Value changed function: tfinalCircuito, tinicialCircuito,
% ...and 1 other component
function bloqueoBotonesGraficasCircuito(app, event)
    app.VoltajeButton.Value=false;
    app.IntensidadButton.Value=false;
    app.IntensidadButton.Enable='off';
    app.VoltajeButton.Enable='off';
end
```

Figura 10 bloqueoBotonesGraficasCircuito

2.2.3. Gestionar Péndulo

En el caso de la ventana del modelo del péndulo sólo tenemos una función que es la de simular.

- *simularPendulo*: esta función ejecuta el archivo *penduloprincipal.m* cuando se pulsa el botón de simular y posteriormente toma los valores necesarios de la base de MATLAB para hacer los diferentes plots, es decir, representaciones, en las dos gráficas que hay: *app.UIAxes2* y *app.UIAxes2_2*.

```
% Button pushed function: SimularPenduloButton
function simularPendulo(app, event)
    evalin('base', ['run ' 'penduloprincipal']);
    t = evalin('base', 't');
    tlin = evalin('base', 'tlin');
    y = evalin('base', 'y');
    ylin = evalin('base', 'ylin');
    plot(app.UIAxes2,t,y(:,1),tlin,ylin(:,1),'r--');
    plot(app.UIAxes2_2,t,y(:,2),tlin,ylin(:,2),'r--');
    grid(app.UIAxes2, 'on');
    grid(app.UIAxes2_2, 'on');
end
```

Figura 11 simularPendulo

2.2.4. Gestionar Tanques Acoplados

Estos son los callbacks utilizados para la ventana de los Tanques Acoplados:

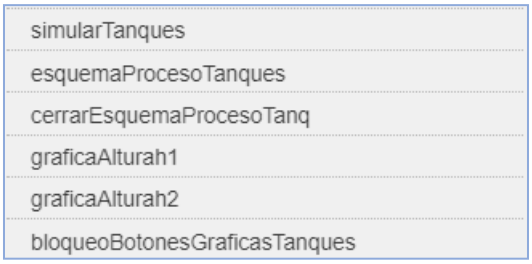

Figura 12 Callbacks Gestionar Tanques Acoplados

- *simularTanques*: esta es la función encargada de la simulación del modelo de tanques acoplados cuando se pulsa el botón de simular.
  En una primera instancia se bloquea el botón de simular hasta que la simulación haya terminado, después se toman los valores del tiempo de simulación y se ejecuta el archivo de Simulink *Tanques_alvaro.slx* con el starttime y stoptime escritos por el usuario en los campos editables de la ventana.
  Una vez ejecutado el fichero añadirán las variables resultantes en la base de MATLAB para que posteriormente se puedan utilizar.

```
% Button pushed function: SimularTanques
function simularTanques(app, event)
    app.SimularTanques.Enable='off';
    pause(1);
    app.Altura1erTanqueButton.Value=false;
    app.Altura2ndoTanqueButton.Value=false;
    ti=app.tinicialTanques.Value;
    tf=app.tfinalTanques.Value;
    simulacion=sim('Tanques_alvaro', 'StartTime', num2str(ti), 'StopTime', num2str(tf));
    assignin('base','h1',simulacion.h1);
    assignin('base','h2',simulacion.h2);
    assignin('base','t',simulacion.t);
    assignin('base','tout',simulacion.tout);
    cla(app.UIAxes3);
    app.UIAxes3.Title.String='';
    app.Altura1erTanqueButton.Enable='on';
    app.Altura2ndoTanqueButton.Enable='on';
    app.SimularTanques.Enable='on';

end
```

Figura 13 simularTanques

- *esquemaProcesoTanques* y *cerrarEsquemaProcesoTanq*: abrir y cerrar el esquema del proceso del archivo *Tanques_alvaro.slx* para que el usuario edite las variables modificables. Se mostrará un mensaje cuando se abre y se cierra el esquema, además de activar el botón de cerrar esquema cuando se abre el esquema y desactivarlo cuando se cierra.

```
% Button pushed function: EsquemaProcesoTanqButton
function esquemaProcesoTanques(app, event)
    open_system('Tanques_alvaro.slx');
    app.cerrarButton_2.Enable='on';
    app.esquemaProcesoMsj_2.Text='Se ha abierto el Esquema';
    pause(7);  % Pausar la ejecución durante 7 segundos
    app.esquemaProcesoMsj_2.Text = '';
end

% Button pushed function: cerrarButton_2
function cerrarEsquemaProcesoTanq(app, event)
    % Cerrar el modelo de Simulink
    close_system('Tanques_alvaro.slx', 0);
    app.esquemaProcesoMsj_2.Text='Se ha cerrado el Esquema';
    app.cerrarButton_2.Enable='off';
    pause(3);  % Pausar la ejecución durante 3 segundos
    app.esquemaProcesoMsj_2.Text = '';
end
```

Figura 14 esquemaProcesoTanques y cerrarEsquemaProcesoTanq

- *graficaAlturah1* y *graficaAlturah2*: estos dos callbacks funcionan muy similar a los de las gráficas de voltaje e intensidad del Circuito RC. Toman los valores que necesitan representar de la base de MATLAB, cambian el nombre de la gráfica que se vaya a representar y representan los datos con el plot en la gráfica: *app.UIAxes3*.

```
% Value changed function: Altura1erTanqueButton
function graficaAlturah1(app, event)
    t = evalin('base', 't');
    h1 = evalin('base', 'h1');
    app.UIAxes3.Title.String='Gráfica 1er Tanque';
    plot(app.UIAxes3, t,h1,'R');
    grid(app.UIAxes3, 'on');
    app.Altura2ndoTanqueButton.Value=false;
end

% Value changed function: Altura2ndoTanqueButton
function graficaAlturah2(app, event)
    t = evalin('base', 't');
    h2 = evalin('base', 'h2');
    app.UIAxes3.Title.String='Gráfica 2ndo Tanque';
    plot(app.UIAxes3, t,h2);
    grid(app.UIAxes3, 'on');
    app.Altura1erTanqueButton.Value=false;
end
```

Figura 15 graficaAlturah1 y graficaAlturah2

- *bloqueoBotonesGraficasTanques*: esta función tiene la misma funcionalidad que *bloqueoBotonesGraficasCircuito*, usada para que cuando hay un cambio en el tiempo de simulación el usuario deba ejecutar la simulación de nuevo.

```
% Value changed function: tfinalTanques, tinicialTanques
function bloqueoBotonesGraficasTanques(app, event)
    app.Altura1erTanqueButton.Value=false;
    app.Altura2ndoTanqueButton.Value=false;
    app.Altura1erTanqueButton.Enable='off';
    app.Altura2ndoTanqueButton.Enable='off';
end
```

Figura 16 bloqueoBotonesGraficasTanques

2.2.5. Pestañas Modelos Básicos

Este último callback relacionado con los modelos básicos, consiste en dejar cada modelo básico de forma predeterminada cuando el usuario pulsa en su pestaña. Es decir, limpiar las variables de la base de MATLAB, limpiar la gráfica, desactivar los botones de visualización de gráficas, etc.
Está diseñada para que no haya errores en los cambios de pestaña y que cada vez que el usuario pulsa en una, el sistema actúe como si fuera la primera vez que lo hace, de esta manera la base de MATLAB no tendrá errores relacionados con las variables añadidas.

```
% Selection change function: Basicos
function cambioVentana(app, event)
    selectedTab = app.Basicos.SelectedTab;
    titulotab=selectedTab.Title;
    switch titulotab
        case 'Circuito RC'
            evalin('base', 'clear');
            cla(app.UIAxes);
            app.UIAxes.Title.String='';
            app.VoltajeButton.Enable='off';
            app.IntensidadButton.Enable='off';
            app.VoltajeButton.Value=false;
            app.IntensidadButton.Value=false;
        case 'Péndulo'
            evalin('base', 'clear');
            cla(app.UIAxes2);
            cla(app.UIAxes2_2);
        case 'Tanques Acoplados'
            evalin('base', 'clear');
            cla(app.UIAxes3);
            app.UIAxes3.Title.String='';
            app.Altura1erTanqueButton.Enable='off';
            app.Altura2ndoTanqueButton.Enable='off';
            app.Altura1erTanqueButton.Value=false;
            app.Altura2ndoTanqueButton.Value=false;
    end
end
```

Figura 17 cambioVentana

2.2.6. Gestionar Depuradora de Agua

En este apartado se explicarán todos los callbacks relacionados con el modelo de la Depuradora de Agua. Las funciones están divididas en función de la ventana que las llama, es decir.
Las primeras funciones son las que se utilizan en la ventana inicial de la Depuradora de Agua, las que están en el recuadro verde son las de la ventana de Gráficas, las del recuadro azul las de la ventana de Índices de Desempeño y las que están en el recuadro naranja las de la ventana de Comparar Resultados.

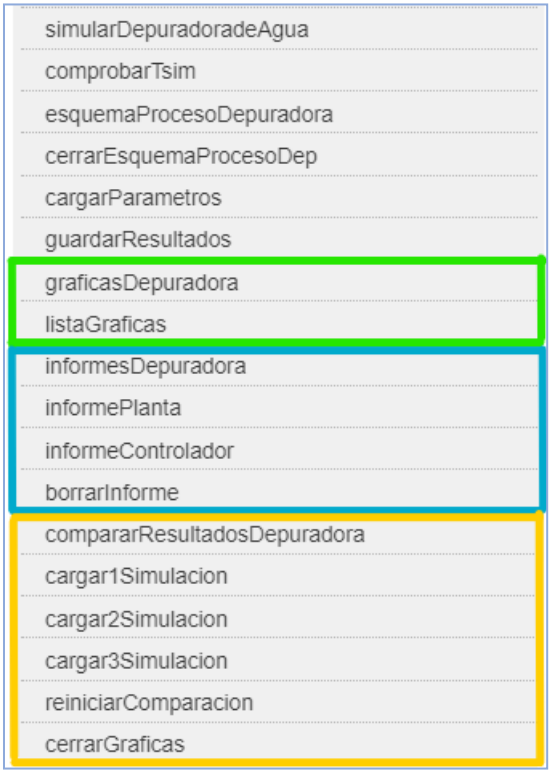

Figura 18 Callbacks Gestionar Depuradora de Agua

De esta forma se identifican fácilmente y permiten al programador ahorrar tiempo buscado las funciones.
A continuación, comenzaremos con los callbacks de la ventana inicial de la Depuradora de Agua.

Ventana Inicial Depuradora de Agua

Las funciones que se utilizan en la ventana principal son las siguientes:

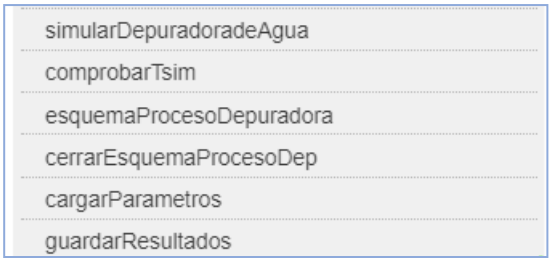

Figura 19 Callbacks Ventana Inicial Depuradora de Agua

- *simularDepuradoradeAgua*: este callback es el encargado de ejecutar la simulación de la Depuradora de agua.
  En primera instancia bloquea los botones que no se pueden utilizar hasta que se termine la ejecución de la misma, después ejecuta el archivo *benchmarkinit.m* que inicializa las variables necesarias para la simulación y añade las variables starttime y stoptime con los valores introducidos por el usuario en los campos editables.
  Una vez hechos los preparativos, se ejecuta la simulación con el archivo *benchmark.mdl* y empieza a añadir todas las variables resultantes de la simulación a la base de MATLAB que son necesarias para su posterior uso.
  Después, se empiezan a activar todos los botones que ya pueden ser utilizados por el usuario: guardar resultados, visualizar los resultados gráficamente, ver los índices de desempeño.
  Además, limpia la ventana de Índices de Desempeño de posibles datos anteriores para que no exista confusión cuando el usuario entre a visitar esta ventana.
  De esta forma la simulación del modelo más avanzado terminaría y el usuario podría acceder a las funcionalidades restringidas.

```
% Button pushed function: SimularButton
function simularDepuradoradeAgua(app, event)
    app.SimularButton.Enable='off';
    app.GuardarresultadosButton.Enable='off';
    app.GraficasButton.Enable='off';
    app.IndicesdedesempeoButton.Enable='off';
    app.GraficasMenu.Enable='off';
    app.InformesMenu.Enable='off';
    pause(1);

    evalin('base', ['run ' 'benchmarkinit.m']);
    assignin('base','starttime', (app.tinicialDepuradora.Value));
    assignin('base','stoptime', (app.tfinalDepuradora.Value));

    ti=app.tinicialDepuradora.Value;
    tf=app.tfinalDepuradora.Value;
    % Iniciar la simulación utilizando la función sim con las opciones de simulación
    simulacion=sim('benchmark', 'StartTime', num2str(ti), 'StopTime', num2str(tf));

    assignin('base','ASinput',simulacion.ASinput);
    assignin('base','Qintrreg',simulacion.Qintrreg);
    assignin('base','SNO2sensor',simulacion.SNO2sensor);
    assignin('base','SO5reg',simulacion.SO5reg);
    assignin('base','SO5sensor',simulacion.SO5sensor);
    assignin('base','carbon1in',simulacion.carbon1in);
    assignin('base','carbon2in',simulacion.carbon2in);
    assignin('base','carbon3in',simulacion.carbon3in);
    assignin('base','carbon4in',simulacion.carbon4in);
    assignin('base','carbon5in',simulacion.carbon5in);
                    • • •
```

Figura 20 simularDepuradoradeAgua 1

```
    %Si hemos simulado podemos: Guardar Resultados, Ver las
    %Ver Graficas e Indices de Desempeño
    app.SimularButton.Enable = 'on';
    app.GuardarresultadosButton.Enable='on';
    app.GraficasButton.Enable='on';
    app.IndicesdedesempeoButton.Enable='on';
    app.GraficasMenu.Enable='on';
    app.InformesMenu.Enable='on';
    %Ventana de Índices de Desemepño
    app.tinicial_inf.Value=app.tinicialDepuradora.Value;
    app.tfinal_inf.Value=app.tfinalDepuradora.Value;
    app.InformedeControladorButton.Enable='off';
    app.InformedePlantaButton.Value=false;
    app.InformedeControladorButton.Value=false;
    app.textoInforme.Value='';
end
```

Figura 21 simularDepuradoradeAgua 2

- *comprobarTsim*: esta función está creada para avisar al usuario de que no puede poner un tiempo inicial de simulación superior a 7 días ni un tiempo final de simulación inferior a 14 días.
  Si el usuario lo incumple, el sistema bloqueará el botón de simulación y le mostrará un mensaje por pantalla.

```
% Value changed function: tfinalDepuradora, tinicialDepuradora
function comprobarTsim(app, event)
    ti = app.tinicialDepuradora.Value;
    tf =  app.tfinalDepuradora.Value;
    if(ti<=7)
        app.TinicialMsj.Text='';
        app.SimularButton.Enable='on';
    end
    if(tf>=14)
        app.TfinalMsj.Text='';
        app.SimularButton.Enable='on';
    end
    if(ti>7)
        app.TinicialMsj.Text='ERROR No introducir Ti mayor de 7';
        app.SimularButton.Enable='off';
    end
    if(tf<14)
        app.TfinalMsj.Text='ERROR No introducir Tf menor de 14';
        app.SimularButton.Enable='off';
    end


end
```

Figura 22 comprobarTsim

- *esquemaProcesoDepuradora* y *cerrarEsquemaProcesoDep*: al igual que el callback *esquemaProcesoTanques* y *cerrarEsquemaProcesoTanq*, esta función tiene como objetivo abrir y cerrar el archivo *benchmark.mdl* que es donde se encuentra el esquema del proceso del modelo. Tiene las mismas funcionalidades adaptadas a este modelo.

```
% Button pushed function: EsquemaProcesoButton
function esquemaProcesoDepuradora(app, event)
        % Abrir el modelo de Simulink e inicia benchamrkinit por si
        % se hace la simulación desde Simulink
        evalin('base', ['run ' 'benchmarkinit.m']);
        assignin('base','starttime', str2double(app.tinicialDepuradora.Value));
        assignin('base','stoptime', str2double(app.tfinalDepuradora.Value));
        open_system('benchmark');
        app.CargarparmetrosButton.Enable='off';
        app.cerrarButton.Enable='on';
        app.esquemaProcesoMsj.Text='Se ha abierto el Esquema';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.esquemaProcesoMsj.Text = '';
end

% Button pushed function: cerrarButton
function cerrarEsquemaProcesoDep(app, event)
    % Cerrar el modelo de Simulink
    close_system('benchmark', 0);
    app.CargarparmetrosButton.Enable='on';
    app.esquemaProcesoMsj.Text='Se ha cerrado el Esquema';
    app.cerrarButton.Enable='off';
    app.CargarparmetrosButton.Enable='on';
    pause(3);  % Pausar la ejecución durante 3 segundos
    app.esquemaProcesoMsj.Text = '';

end
```

Figura 23 esquemaProcesoDepuradora y cerrarEsquemaProcesoDep

- *cargarParametros*: esta función está desarrollada como funcionalidad alternativa a la modificación de variables a través del esquema del proceso. Permite al usuario cargar los datos de una simulación guardada anteriormente.
  Lo primero que hace es abrir el explorador de archivos para que seleccione el que quiere cargar de tipo *.mat. Si el archivo o la ruta no existen, entonces se mostrará un mensaje por pantalla diciendo que no se han podido cargar los parámetros.
  Si se selecciona un archivo, se tomará la ruta absoluta y se ejecutará el propio archivo en la base de MATLAB y se mostrará por pantalla la ruta del archivo que se ha cargado.
  Una vez cargado el archivo en la base se activarán las funcionalidades restringidas al no tener datos de simulación: visualizar los resultados gráficamente y ver los índices de desempeño.

```
% Button pushed function: CargarparmetrosButton
function cargarParametros(app, event)
    [archivo, ruta] = uigetfile('*.m', 'Seleccionar archivo para ejecutar');
    if isequal(archivo, 0) || isequal(ruta, 0)
        app.cargadosParametrosMsj.Text='¡No se han podido cargar los parámetros!';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.cargadosParametrosMsj.Text = '';
    else
        archivoCompleto = fullfile(ruta, archivo);
        evalin('base', ['run ' archivoCompleto]);
        texto=sprintf('Archivo seleccionado: \n%s ', archivoCompleto);
        app.cargadosParametrosMsj.Text=texto;
        app.GraficasButton.Enable='on';
        app.IndicesdedesempeoButton.Enable='on';
        app.GraficasMenu.Enable='on';
        app.InformesMenu.Enable='on';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.cargadosParametrosMsj.Text = '';
    end
end
```

Figura 24 cargarParametros

- *guardarResultados*: el objetivo de este callback es crear un archivo con los datos de la simulación realizada y guardarlo en un fichero elegido por el usuario para que posteriormente se pueda cargar en el propio sistema y utilizar las funcionalidades ofrecidas.
  En una primera instancia se guardarán todas las variables que se encuentran en la base de MATLAB en una variable que contendrá todas y se irá recorriendo en un bucle.
  En cada iteración del bucle se cogerá el nombre de las variables y su valor y se guardará en una cadena, de forma que se irán almacenando todas las variables, una en cada línea, con su nombre y valor.
  Una vez hecho esto, se abrirá el explorador de archivos para que el usuario seleccione la ruta de guardado y el nombre del archivo.
  Si no hay ningún error, se creará el archivo y se escribirá el contenido de todas las variables que se han almacenado en el bucle.
  Se mostrará por pantalla la ruta de guardado y finaliza la ejecución.
  Si ha habido un error a la hora de nombrar el archivo o seleccionar la ruta, se mostrará un mensaje por pantalla y finalizará la ejecución.

```
function guardarResultados(app, event)
    variables = evalin('base', 'who');
    contenido = '';
    for i = 1:numel(variables)
        nombre = variables{i};
        valor = evalin('base', nombre);
        contenido = [contenido, sprintf('%s = %s;\n', nombre, mat2str(valor))];
    end
    [archivo, ruta] = uiputfile('*.m','Guardar Archivo');
     if archivo ~= 0
        fid = fopen(archivo, 'w');
        fprintf(fid, contenido);
        fclose(fid);
        rutaActual=pwd;
        caracter='\';
        rutafinal=[rutaActual,caracter];
        disp(rutafinal);
        disp(ruta);
        if strcmp(rutafinal,ruta)
            %%no hace nada
        else
            movefile(archivo, ruta);%%mueve el archivo a la carpeta
        end
        app.GuardarresultadosButton.Enable='off';
        texto=sprintf('Se han guardado los resultados en: \n%s',ruta);
        app.guardarResultadosMsj.Text=texto;
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.guardarResultadosMsj.Text = '';
     else
        app.guardarResultadosMsj.Text='No se han podido guardar los resultados';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.guardarResultadosMsj.Text = '';
     end
end
```

Figura 25 guardarResultados

Ventana Gráficas

Las funciones que se utilizan en la ventana de las gráficas:

Figura 26 Callbacks Ventana Gráficas Depuradora de Agua

- *graficasDepuradora*: este callback llama a la función comentada anteriormente *visibleGraficas* que permite al sistema mostrar la ventana de gráficas al usuario ocultando todas las demás.
- *listaGraficas*: este callabck llama a la función comentada anteriormente *mostrarGrafica*, función que le pasa el listbox de la ventana que contiene la lista de gráficas que el usuario puede seleccionar. Permite visualizar la gráfica seleccionada dentro de ese listbox.

```
% Button pushed function: GraficasButton
function graficasDepuradora(app, event)
    visibleGraficas(app);
end

% Value changed function: ListadeGrficasListBox
function listaGraficas(app, event)
    mostrarGrafica(app,app.ListadeGrficasListBox);
end
```

Figura 27 graficasDepuradora y listaGraficas

Ventana Índices de Desempeño

Las funciones que se utilizan en la ventana de Índices de Desempeño:

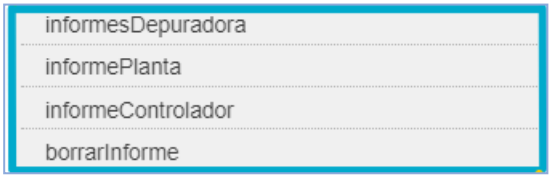

Figura 28 Callbacks Ventana Índices de Desempeño Depuradora de Agua

- *informesDepuradora*: al igual que *graficasDepuradora* llama a su función correspondiente de visualización de ventana para que al pulsar el botón el usuario sea "redirigido" a la ventana de índices de desempeño.
- *informePlanta* y *informeControlador*: estos callbacks permiten al usuario la visualización de los informes de planta y el controlador de

forma escrita en la propia ventana y de forma gráfica en ventanas emergentes.

Se pueden visualizar los informes en el rango de tiempos que el usuario considere, obviamente, estos tiempos deben de estar dentro del rango del tiempo simulado, por lo que si no están dentro de ese rango se le mostrará un error por pantalla informándole.

Si los tiempos introducidos están dentro del rango, entonces se añadirán las variables starttime_inf y stoptime_inf con esos valores introducidos a la base de MATLAB.

Se comprueba que no exista el archivo informePlanta.txt o informeControlador.txt, en el caso de que ya existiera se borra pues se trataría de un informe de otra simulación.

A continuación, se crea el archivo de texto y empieza a capturar la salida de resultados de MATLAB, para que cuando se ejecute el archivo de *perf_plant.m* o *perf_controller.m* se guarde todo en el archivo creado.

Posteriormente se carga el contenido de ese archivo en una cadena de texto y se introduce en el objetivo *app.textoInforme.Value* que es donde se va a visualizar la salida, en la propia ventana.

```
function informePlanta(app, event)
    app.InformedeControladorButton.Value=false;
    app.InformedeControladorButton.Enable='on';
    tf =  app.tfinalDepuradora.Value;
    ti_i = app.tinicial_inf.Value;
    tf_i = app.tfinal_inf.Value;

    if (tf_i>tf) || (tf_i<=7)
        cadena=sprintf('ERROR CON EL TIEMPO INTRODUCIDO \n\nNo se puede introducir
        app.textoInforme.Value=cadena;
        return;
    end
    assignin('base','starttime_inf', ti_i);
    assignin('base','stoptime_inf', tf_i);

    if exist('informePlanta.txt', 'file')
        delete('informePlanta.txt');
    end
    diary('informePlanta.txt');  % Nombre del archivo donde se guardará la salida

    evalin('base',['run ' 'perf_plant.m']);

    diary off;  % Detener la captura de la salida en el archivo

    % Leer el contenido del archivo
    contenido = fileread('informePlanta.txt');

    contenidoString = string(contenido);

    % Actualizar el contenido de la etiqueta con la salida capturada
    app.textoInforme.Value=contenidoString;
end
```

Figura 29 informePlanta

```
function informeControlador(app, event)
    app.InformedePlantaButton.Value=false;
    tf = app.tfinalDepuradora.Value;
    ti_i = app.tinicial_inf.Value;
    tf_i = app.tfinal_inf.Value;

    if (tf_i>tf) || (tf_i<=7)
        cadena=sprintf('ERROR CON EL TIEMPO INTRODUCIDO \n\nNo se puede introducir un
        app.textoInforme.Value=cadena;
        return;
    end
    assignin('base','starttime_inf', ti_i);
    assignin('base','stoptime_inf', tf_i);

    if exist('informeControlador.txt', 'file')
        delete('informeControlador.txt');
    end

    diary('informeControlador.txt');  % Nombre del archivo donde se guardará la salida

    evalin('base',['run ' 'perf_controller.m']);

    diary off;  % Detener la captura de la salida en el archivo

    % Leer el contenido del archivo
    contenido = fileread('informeControlador.txt');

    contenidoString = string(contenido);

    % Actualizar el contenido de la etiqueta con la salida capturada
    app.textoInforme.Value=contenidoString;
end
```

Figura 30 informeControlador

- *borrarInforme*: esta función está escrita para borrar el informe escrito en la ventana cuando se cambian los tiempos de visualización.

```
% Value changed function: tfinal_inf, tinicial_inf
function borrarInforme(app, event)
    app.InformedeControladorButton.Enable='off';
    app.InformedePlantaButton.Value=false;
    app.InformedeControladorButton.Value=false;
    app.textoInforme.Value='';
end
```

Figura 31 borrarInforme

Ventana Comparar Resultados

Las funciones que se utilizan en la ventana de Comparar Resultados:

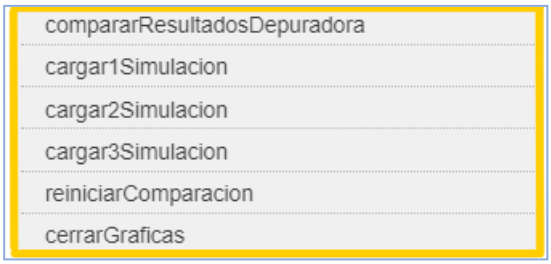

Figura 32 Callbacks Ventana Comparar Resultados Depuradora de Agua

- *compararResultadosDepuradora*: al igual que los otros dos callbacks explicados: *graficasDepuradora* e *informesDepuradora*. Muestran al usuario la ventana de comparar resultados ocultando todas las otras ventanas. Para esto llama a la función explicada al principio *visibleCompararResultados*.
- *Cargar1Simulacion, cargar2Simulacion* y *cargar3Simulacion*: estas funciones se activan cuando se pulsan los botones de cargar y se ha seleccionado previamente la gráfica donde se quieren comparar los resultados.

  En una primera instancia, se abre el explorador de archivos para que el usuario seleccione la simulación que quiere cargar, si el archivo no existe o no selecciona ninguno, se mostrará un mensaje por pantalla informándole.

  En caso contrario, ejecutará el archivo en cuestión, es decir, cargará todas las variables resultantes de la simulación en la base de MATLAB y mostrará un mensaje por pantalla indicando que se ha cargado correctamente.

  Finalmente llamará a la función explicada inicialmente *mostrarGrafica* pasándole la gráfica que ha seleccionado al principio y esta función sacará por pantalla los resultados de la simulación cargada visualizados en la gráfica seleccionada.

  A medida que se vayan cargando más simulaciones, los datos de estas se representarán en la misma gráfica, de forma que se puedan comparar los resultados.

```
% Button pushed function: cargarButton_1
function cargar1Simulacion(app, event)
    [archivo, ruta] = uigetfile('*.m', 'Seleccionar archivo para ejecutar');
    if isequal(archivo, 0) || isequal(ruta, 0)
        app.cargarCompa1Msj.Text='¡No se han podido cargar los parámetros!';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.cargarCompa1Msj.Text = '';
    else
        archivoCompleto = fullfile(ruta, archivo);
        disp(['Archivo seleccionado: ' archivoCompleto]);
        evalin('base', ['run ' archivoCompleto]);
        app.cargarCompa1Msj.Text='¡Se han cargado perfectamente!';
        app.listadegrficasDropDown.Enable='off';
        mostrarGrafica(app,app.listadegrficasDropDown);

        app.cargarButton_1.Enable='off';
        pause(7);  % Pausar la ejecución durante 7 segundos
        app.cargarCompa1Msj.Text = '';

    end
end
```

Figura 33 cargar1Simulacion

- *reiniciarComparacion*: esa función está diseñada para poder realizar otra comparación, es decir, poder seleccionar otros datos y otra gráfica donde visualizarlos.
  Activa el botón de la lista de gráficas para que se pueda seleccionar de nuevo una y activa los botones de cargar simulaciones para añadir nuevos datos.

```
% Button pushed function: ReiniciarButton
function reiniciarComparacion(app, event)
    app.listadegrficasDropDown.Enable="on";
    app.cargarButton_1.Enable='on';
    app.cargarButton_2.Enable='on';
    app.cargarButton_3.Enable='on';
end
```

Figura 34 reiniciarComparacion

- *cerrarGraficas*: simplemente llama al comando *close all* para que todas las gráficas que se hayan abierto, se cierren instantáneamente. Esta funcionalidad es muy útil ya que en esta aplicación se abren muchas ventanas emergentes con gráficas, por lo que es tedioso ir cerrando todas de una en una.

```
function cerrarGraficas(app, event)
    close all;
end
```

Figura 35 cerrarGraficas

2.3. Masks

Las máscaras son una herramienta que ofrece Simulink para añadir interfaces a subsistemas del esquema del proceso y de esta forma permitir al usuario modificar ciertas variables del modelo.
En mi caso se ha utilizado en los dos únicos ficheros de Simulink que se usan en el sistema: Tanques_alvaro.slx y benchmark.mdl.

2.3.1. Tanques_alvaro

Para el desarrollo de las máscaras de este esquema primero tuve que crear los subsistemas: 1er Tanque Acoplado y 2ndo Tanque Acoplado.
De esta forma ya se podría crear una máscara para cada uno.

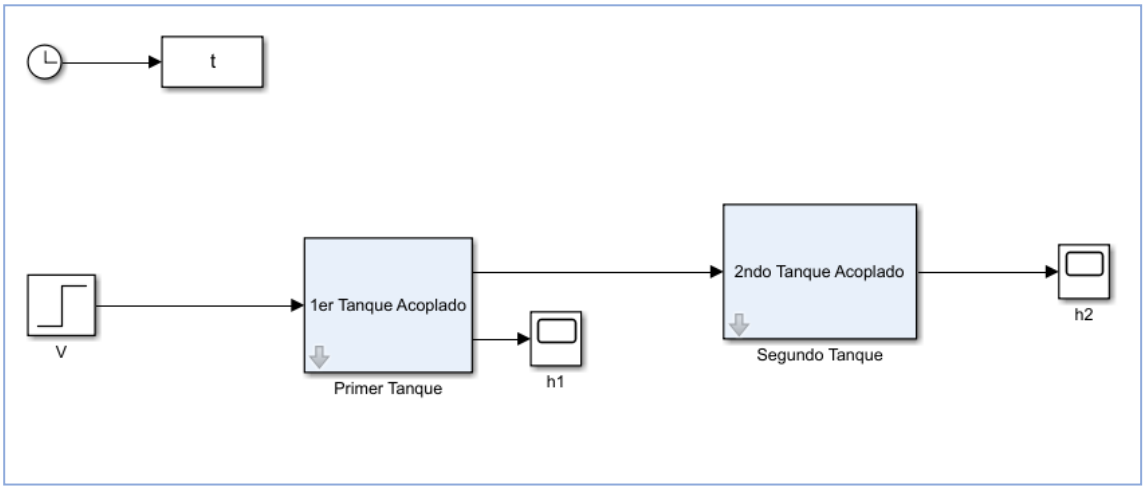

Figura 36 Esquema Proceso Tanques Acoplados

- *1er Tanque Acoplado* y *2ndoTanque Acoplado*: el sistema de diseño de código es por bloques.
  El programador va añadiendo a esta ventana: pestañas, etiquetas, campos editables, etc. Cada uno de los elementos que se añaden se mostrarán en la máscara cuando el usuario pulse en el subsistema.
  En este en concreto se añadió un campo con el nombre de la máscara, indicando que es el subsistema del primer o segundo tanque y se añadió una etiqueta con una descripción del mismo.
  Después se añadieron dos conteiner (pestañas) para dividir los parámetros entre los campos numéricos y el campo de la altura inicial que se modificará con una barra numérica dónde se señaliza la altura moviendo la flecha.

Type	Prompt	Name
	Subsistema del Primer Tanq...	DescGroupVar
A	Este el subsistema del prime...	DescTextVar
	Parámetros	Container3
	(N/A)	Container4
	Diseño	Container5
#1	Superficie del Tanque (m²)	A1
#2	Radio del Tanque (m)	r1
	Altura Inicial	Container6
#3	Altura Inicial del Agua (m)	h10

Figura 38 Máscara Primer Tanque Acoplado

Type	Prompt	Name
	Subsistema del Segundo Ta...	DescGroupVar
A	Este el subsistema del segu...	DescTextVar
	Parámetros	Container3
	(N/A)	Container4
	Diseño	Container5
#1	Superficie del Tanque (m²)	A2
#2	Radio del Tanque (m)	r2
	Altura Inicial	Container6
#3	Altura Inicial del Agua (m)	h20

Figura 37 Máscara Segundo Tanque Acoplado

2.3.2. benchmark

Al igual que el archivo de tanques acoplados, en este tuve que crear los subsistemas de los que iba a crear las máscaras, en el caso de este modelo fueron muchos más, al tratarse de un modelo más complejo con más variables involucradas.
Los subsistemas modificados son: los 5 reactores, el controlador de oxígeno, el controlador de nitratos y el decantador secundario.

Al ser tantos subsistemas tuve que reorganizar las líneas de unión para que el esquema quedara completamente entendible, además de añadir imágenes a algunos subsistemas como los reactores y el decantador secundario.

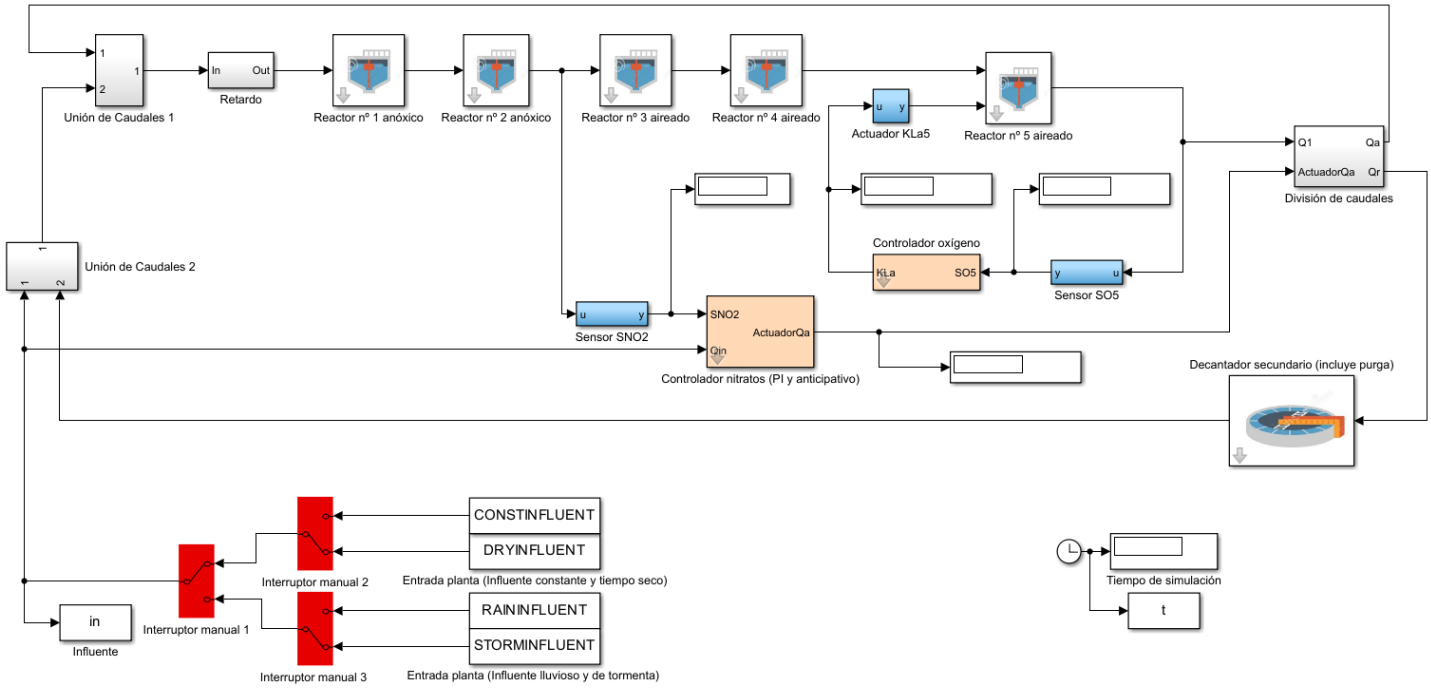

Figura 39 Esquema Proceso Depuradora de Agua

El diseño de bloques de estos subsistemas es muy similar al de los tanques acoplados, mostrando el nombre de la máscara y dividiendo los parámetros dependiendo de su funcionalidad
En este caso hay una división de dos pestañas en los Controladores de Oxígeno y Nitratos para diferenciar el campo del valor de referencia de las constantes de proporcionalidad, tiempo integral, etc.

- *Reactores*:

Type	Prompt	Name
	Reactor nº 1 anóxico	DescGroupVar
A	%<MaskDescription>	DescTextVar
	Parámetros	ParameterGroupVar
123 #1	carb1	carb1
123 #2	KLa1	KLa1

Figura 40 Máscara Reactor

- *Controlador de Oxígeno*:

Type	Prompt	Name
	Controlador oxígeno	DescGroupVar
A	%<MaskDescription>	DescTextVar
	(N/A)	Container3
	Referencia	Container4
123 #1	SO5ref	SO5ref
	Parámetros	Container5
123 #2	Kp (Constante proporcional)	KSO5
123 #3	Ti (Tiempo integral)	TiSO5
123 #4	Constante antiwindup	TtSO5

Figura 41 Máscara Controlador de Oxígeno

- *Controlador de Nitratos*:

Type	Prompt	Name
	Controlador nitratos (PI y ant...	DescGroupVar
A	%<MaskDescription>	DescTextVar
	(N/A)	Container3
	Referencia	Container4
123 #1	SNO2ref	SNO2ref
	Parámetros	Container5
123 #2	Kp (Constante proporcional)	KQintr
123 #3	Ti (Tiempo Integral)	TiQintr
123 #4	Constante antiwindup	TtQintr
123 #5	Kfeedfordward	Kfeedforward

Figura 42 Máscara Controlador de Nitratos

- *Decantador Secundario*:

Type	Prompt	Name
▾	Decantador Secundario	DescGroupVar
A	%<MaskDescription>	DescTextVar
▾	Parámetros	ParameterGroupVar
123 #1	Qin0	Qin0
123 #2	Qw (Caudal de Purga)	Qw

Figura 43 Máscara Decantador Secundario

3. Manual del Programador

Este apartado está definido para recoger las acciones necesarias para generar el producto final a partir del código fuente.
En el caso de mi aplicación al usar el App Designer de MATLAB, permite al programador compartir el proyecto convirtiendo el archivo en cuestión en un archivo de instalación para que el usuario pueda instalar la aplicación en su propio MATLAB y utilizar la aplicación como muchas de las otras herramientas que proporciona MATLAB.

- En la pestaña de diseño del propio archivo existe un botón llamado *Share*, este es el que utilizaremos para crear el archivo de instalación.

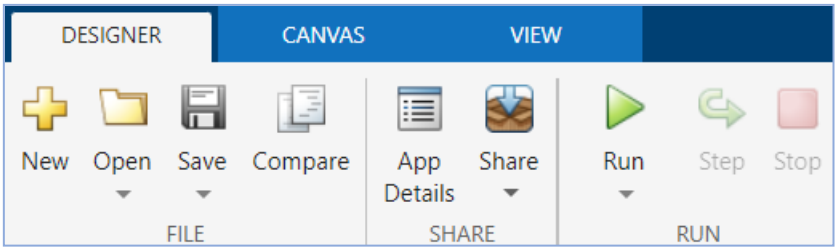

Figura 44 Manual del Programador 1

- Una vez pulsado, seleccionar la opción de MATLAB app.

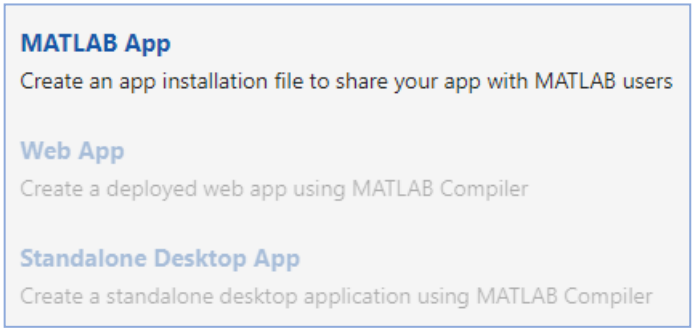

Figura 45 Manual del Programador 2

- Pulsando esa opción se abrirá una ventana emergente que te permitirá seleccionar los archivos accesorios necesarios para la ejecución de la aplicación, en mi caso se trata de la carpeta *ArchivosSim*, además de añadir un nombre, resumen y descripción a la propia aplicación.
  Indicar los productos de MATLAB de los cuales depende la aplicación, que en el caso de este proyecto sería el propio MATLAB y Simulink.
  Finalmente seleccionar la carpeta donde se guardará el archivo de instalación y pulsar el botón *Package*.

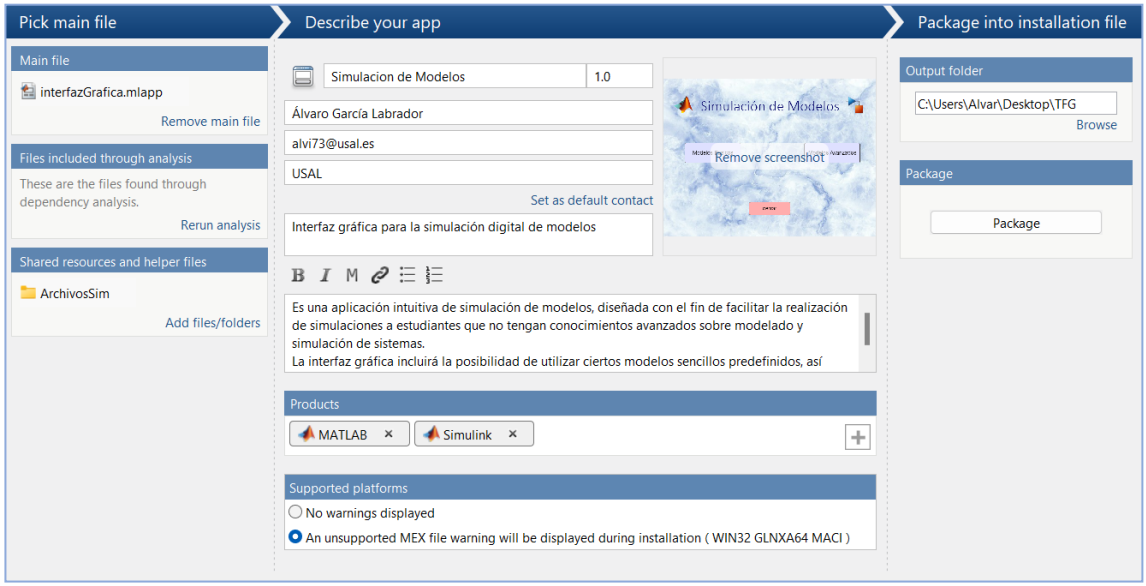

Figura 46 Manual del Programador 3

- De esta forma quedaría ya listo el archivo de instalación para compartirlo con el usuario que quiera utilizar la aplicación.

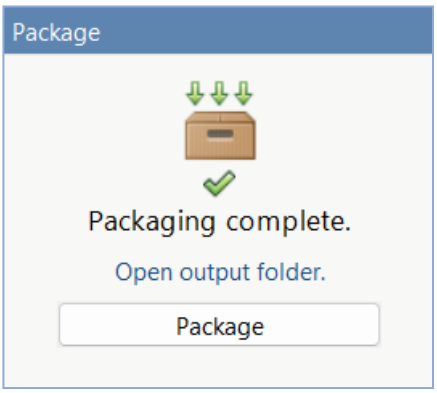

Figura 47 Manual del Programador 4

4. Pruebas Unitarias

Las pruebas unitarias han sido necesarias para el perfecto desarrollo de la aplicación. Debido al poco conocimiento inicial del entorno de desarrollo y su lenguaje de programación, las pruebas en el desarrollo de la aplicación han sido constantes, con más carga al principio que al final debido a la adaptación y aprendizaje del medio.
Las pruebas iniciales llegaron de la mano del desconocimiento y de búsquedas en la página principal de MATLAB para, poco a poco, irme adaptando a este entorno de desarrollo de aplicaciones e ir aumentando mis conocimientos de su lenguaje de programación.
Es verdad que estas primeras pruebas no supusieron mucho tiempo, en lo que a corrección de errores se trataba, pues al tratarse del comienzo del proyecto, las pruebas eran más simples y la corrección de estos errores también.

A medida que fue pasando el tiempo y desarrollándose la aplicación, mis conocimientos del lenguaje fueron aumentando, por ende, las pruebas que se ejecutaban eran más complejas y con errores más difíciles de solucionar, de no ser por las constantes búsquedas, tanto en la página de MATLAB, como en YouTube, etc.
El modelo avanzado de la depuradora de aguas residuales ha sido el objetivo al que más tiempo le he dedicado, debido a su complejidad y la cantidad de funcionalidades ofrecidas.
Sobre todo, mencionar las funcionalidades: guardar resultados, comparar resultados y los índices de desempeño.
Estas tres funcionalidades son las que más pruebas han requerido para su perfecto desarrollo final, puesto que son las funcionalidades más complejas en cuanto a código se refiere, se realizaron muchas pruebas con muchos errores que con el tiempo y la constancia se fueron puliendo.

Finalmente, las últimas pruebas de la aplicación fueron junto a las iniciales las que menos carga de trabajo tuvieron, puesto que, al tratarse ya de los últimos detalles, los cambios eran menos relevantes y más fácil de realizarlos. Estas últimas pruebas tendrían que ver más con el diseño de la aplicación, posición de los botones, de las gráficas, fondos de las ventanas, etc.