14 de marzo de 2019

Laura Aguilera Checa

Ignacio Aguilera Gómez

Grado en ingeniería informática

Universidad de Almería

Práctica 2. Concurrencia de tareas y sincronización

Sistemas de tiempo real

Contenido

[1. Autores 2](#_Toc5124876)

[2. Introducción 2](#_Toc5124877)

[3. Actividades a realizar 3](#_Toc5124878)

[3.1. Descripción de la planta 3](#_Toc5124879)

[3.2. Red de Petri temporal 3](#_Toc5124880)

[3.3. Implementación en ADA sin tareas concurrentes 4](#_Toc5124881)

[3.4. Implementación en ADA con tareas concurrentes 4](#_Toc5124882)

[3.5. Conclusión 4](#_Toc5124883)

[Anexo. Descripción del problema: Control de una instalación de destilación por membranas 4](#_Toc5124884)

# 1. Autores

|  |  |  |
| --- | --- | --- |
| Apellidos | Aguilera | Checa |
| Nombre | Laura |  |
| Titulación | Grado de Ingeniería Informática | |

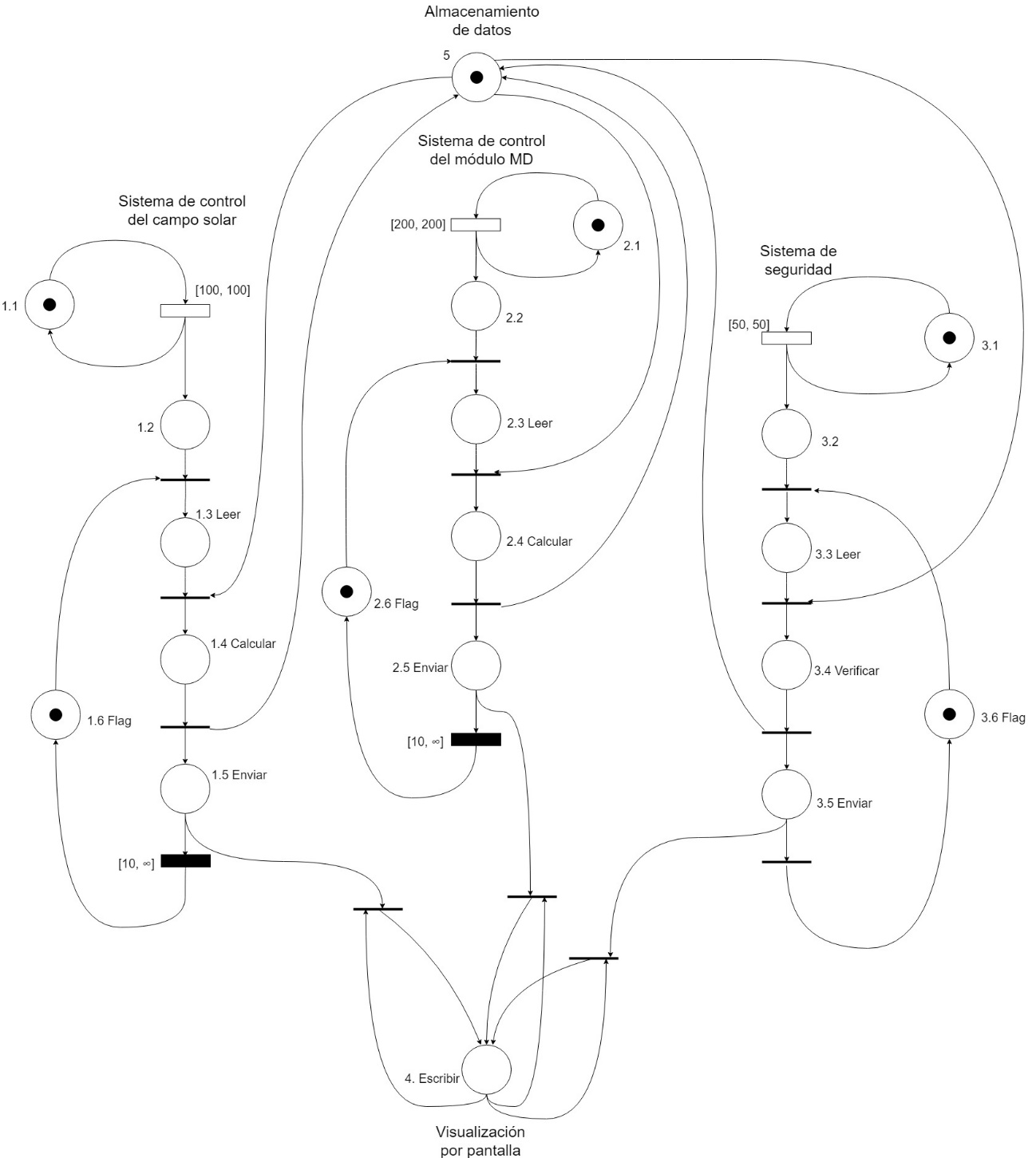
|  |  |  |
| --- | --- | --- |
| Apellidos | Aguilera | Gómez |
| Nombre | Ignacio |  |
| Titulación | Grado de Ingeniería Informática | |

# 2. Introducción

# 3. Actividades a realizar

## 3.1. Descripción de la planta

## 3.2. Red de Petri temporal



|  |  |
| --- | --- |
| Nombre | Descripción |
| 1.1 | Inicialización del sistema de control del campo solar |
| 1.2 | Nodo auxiliar |
| 1.3 | Leer temperatura del campo solar (ST2) |
| 1.4 | Controla SC1 actuando sobre el caudal de la bomba 1 en función a la temperatura leída anteriormente |
| 1.5 | Enviar los datos obtenidos al sistema de visualización por pantalla |
| 1.6 | Flag |
| 2.1 | Inicialización del sistema de control del módulo de destilación |
| 2.2 | Nodo auxiliar |
| 2.3 | Leer el caudal de agua de mar (SC2) |
| 2.4 | Controlar el caudal de destilado (SD1) en función a la temperatura del campo solar (ST2) y regular el caudal actuando sobre la bomba 2 |
| 2.5 | Enviar los datos obtenidos al sistema de visualización por pantalla |
| 2.6 | Flag |
| 3.1 | Inicialización del sistema de seguridad |
| 3.2 | Nodo auxiliar |
| 3.3 | Leer la temperatura la salida del campo solar (ST2) y la temperatura a la entrada del módulo de destilación (ST3) |
| 3.4 | Verificar que ST2 es inferior a 100°C y que ST3 es inferior a 85°C, y, en caso de que no sea así, actuar sobre la bomba 1 y 2 respectivamente |
| 3.5 | Enviar los datos obtenidos al sistema de visualización por pantalla |
| 3.6 | Flag |
| 4 | Escribir por pantalla la información de los sensores del sistema y el estado de los actuadores |
| 5 | Almacenamiento de datos |

## 3.3. Implementación en ADA sin tareas concurrentes

En primer lugar, se describirán los paquetes utilizados.

tipos\_datos.ads

Se definen los tipos de datos para la temperatura, el caudal, la radiación y las bombas.

package Tipos\_Datos is

type Temperatura is new float;

type Caudal is new float;

type Radiacion is new float;

type EstadoBomba is (On, Off);

end Tipos\_Datos;

modulos.ads

Se definen los procedimientos para los módulos de almacenamiento de datos, del campo solar, del módulo de destilación, del sistema de seguridad y de la visualización por pantalla.

with tipos\_datos,

io,

control;

use tipos\_datos;

package modulos is

m\_temp : Temperatura := 30.0;

m\_caudal: Caudal := 25.0;

m\_rad : Radiacion := 1.5;

t\_bomb : estadoBomba := Off;

procedure AlmacenamientoDeDatos;

procedure CampoSolar;

procedure ModuloDestilacion;

procedure SistemaDeSeguridad;

procedure Visualizador;

end modulos;

modulos.adb

with Tipos\_Datos,

Ada.Text\_IO;

use Tipos\_Datos,

Ada.Text\_IO;

package body modulos is

procedure AlmacenamientoDeDatos is

begin

    null;

end AlmacenamientoDeDatos;

procedure CampoSolar is

begin

    Put\_Line("CS: " & io.Leer (m\_temp));

    Put\_Line("CS: " & control.ControlarSC1 (m\_temp, m\_temp, m\_temp, m\_temp, m\_rad, t\_bomb));

    Put\_Line("CS: " & io.Escribir (m\_temp));

    Put\_Line("CS: " & io.Enviar(m\_temp));

    Visualizador;

end CampoSolar;

procedure ModuloDestilacion is

begin

    Put\_Line("MD: " & io.Leer (m\_caudal));

    Put\_Line("MD: " & control.ControlarSC2 (m\_temp, m\_temp, m\_caudal, t\_bomb));

    Put\_Line("MD: " & io.Escribir (m\_temp));

    Put\_Line("MD: " & io.Enviar (m\_temp));

    Visualizador;

end ModuloDestilacion;

procedure SistemaDeSeguridad is

begin

    Put\_Line("SS: " & io.Leer (m\_temp));

    Put\_Line("SS: " & io.Leer (m\_temp));

    Put\_Line("SS: " & io.Verificar);

    Put\_Line("SS: " & io.Enviar (m\_temp));

    Visualizador;

end SistemaDeSeguridad;

procedure Visualizador is

begin

    Put\_Line("V: " & io.Escribir (m\_temp));

    Put\_Line("V: " & io.Escribir (m\_caudal));

    Put\_Line("V: " & io.Escribir (m\_rad));

end Visualizador;

end modulos;

control.ads

Define los métodos para el control de SC1 y SC2 para el campo solar y el módulo de destilación.

with tipos\_datos

, Ada.Text\_IO;

use tipos\_datos

, Ada.Text\_IO;

package control is

function ControlarSC1 (t\_oSt1 : temperatura;

             t\_oSt2 : temperatura;

             t\_st2 : temperatura;

             t\_st4 : temperatura;

             t\_sr1 : radiacion;

             t\_bomb : out estadoBomba) return string;

function ControlarSC2 (t\_oSt2 : temperatura;

             t\_st3 : temperatura;

             t\_sd1 : caudal;

             t\_bomb : out estadoBomba) return string;

end control;

control.adb

with Ada.Text\_IO;

use Ada.Text\_IO;

package body control is

t\_bomb : estadoBomba := Off;

function ControlarSC1 (t\_oSt1 : temperatura;

             t\_oSt2 : temperatura;

             t\_st2 : temperatura;

             t\_st4 : temperatura;

             t\_sr1 : radiacion;

             t\_bomb : out estadoBomba) return string is

begin

    t\_bomb := On;

    return ("Controlando SC1("

     & "), estado de la bomba: " & t\_bomb'image);

end;

function ControlarSC2 (t\_oSt2 : temperatura;

             t\_st3 : temperatura;

             t\_sd1 : caudal;

             t\_bomb : out estadoBomba) return string is

begin

    t\_bomb := On;

    return ("Controlando SC2("

     & "), estado de la bomba: " & t\_bomb'image);

end;

end control;

io.ads

Define las funciones para leer, escribir, enviar y verificar los distintos tipos de datos.

with tipos\_datos;

use tipos\_datos;

package IO is

function Leer (t\_temp : out Temperatura) return string;

function Leer (t\_caud : out Caudal) return string;

function Leer (t\_rad : out Radiacion) return string;

function Escribir (t\_temp : in Temperatura) return string;

function Escribir (t\_caud : in Caudal) return string;

function Escribir (t\_rad : in Radiacion) return string;

function Enviar (t\_temp : in Temperatura) return string;

function Enviar (t\_caud : in Caudal) return string;

function Enviar (t\_rad : in Radiacion) return string;

function Verificar return string;

end IO;

io.adb

with Ada.Text\_IO;

use Ada.Text\_IO;

package body IO is

----------------Leer--------------------

function Leer (t\_temp : out Temperatura) return string is

begin

    t\_temp := 10.0;

    Return ("Leyendo temperatura: " & t\_temp'image);

end;

function Leer (t\_caud : out Caudal) return string is

begin

    t\_caud := 10.0;

    Return ("Leyendo caudal: " & t\_caud'image);

end;

function Leer (t\_rad : out Radiacion) return string is

begin

    t\_rad := 10.0;

    Return ("Leyendo radiacion: " & t\_rad'image);

end;

--------------Escribir------------------

function Escribir (t\_temp : in Temperatura) return string is

begin

    Return ("Escribiendo temperatura: " & t\_temp'image);

end;

function Escribir (t\_caud : in Caudal) return string is

begin

    Return ("Escribiendo caudal: " & t\_caud'image);

end;

function Escribir (t\_rad : in Radiacion) return string is

begin

    Return ("Escribiendo radiacion: " & t\_rad'image);

end;

---------------Enviar-------------------

function Enviar (t\_temp : in Temperatura) return string is

begin

    Return ("Enviando temperatura: " & t\_temp'image);

end;

function Enviar (t\_caud : in Caudal) return string is

begin

    Return ("Enviando caudal: " & t\_caud'image);

end;

function Enviar (t\_rad : in Radiacion) return string is

begin

    Return ("Enviando radiacion: " & t\_rad'image);

end;

-------------Verificar-----------------

function Verificar return string is

begin

    Return ("Comprobando temperaturas");

end;

end IO;

Implementación del problema

with modulos;

with Ada.Text\_IO;

use Ada.Text\_IO;

procedure mainSecuencial is

reps : integer := 10; -- Variable que indica el numero de repeticiones del bucle

i : integer := 0; -- Variable auxiliar para el bucle del numero de repeticiones

begin

for i in 1 .. reps loop

    modulos.CampoSolar; -- 10 ms

    modulos.ModuloDestilacion; -- 10 ms

    -- Como suponemos que cada iteracion del bucle son 10ms, y sabemos que

    -- el sistema de seguridad se ejecuta cada 50ms, lo llamamos cada 5 iteraciones

    if (i mod 5 = 0) then

     modulos.SistemaDeSeguridad; -- 50 ms

    end if;

end loop;

end mainSecuencial;

Resultado de la ejecución (para una sola repetición)

CS: Leyendo temperatura: 1.00000E+01

CS: Controlando SC1(), estado de la bomba: ON

CS: Escribiendo temperatura: 1.00000E+01

CS: Enviando temperatura: 1.00000E+01

V: Escribiendo temperatura: 1.00000E+01

V: Escribiendo caudal: 2.50000E+01

V: Escribiendo radiacion: 1.50000E+00

MD: Leyendo caudal: 1.00000E+01

MD: Controlando SC2(), estado de la bomba: ON

MD: Escribiendo temperatura: 1.00000E+01

MD: Enviando temperatura: 1.00000E+01

V: Escribiendo temperatura: 1.00000E+01

V: Escribiendo caudal: 1.00000E+01

V: Escribiendo radiacion: 1.50000E+00

[2019-04-25 20:40:12] process terminated successfully, elapsed time: 01.10s

## 3.4. Implementación en ADA con tareas concurrentes

with modulos;

with Ada.Real\_Time;

use Ada.Real\_Time;

procedure mainConcurrente is

reps : integer := 10; -- Número de iteraciones en la ejecucion del ejercicio

i : integer := 0; -- Variable auxiliar para el bucle del Campo Solar

j : integer := 0; -- Variable auxiliar para el bucle del Modulo de Destilacion

k : integer := 0; -- Variable auxiliar para el bucle del Sistema de Seguridad

Tiempo1, Tiempo2, Tiempo3 : Time; -- Variable para controlar el tiempo de cada modulo

Limite1, Limite2, Limite3 : Time; -- Variable para almacenar el limite temporal de cada modulo

C1 : Time\_Span := Milliseconds (10); -- Variable para almacenar el tiempo de computo de un modulo

C2 : Time\_Span := Milliseconds (10); -- Variable para almacenar el tiempo de computo de un modulo

T1 : Time\_Span := Milliseconds (100); -- Variable para almacenar el de periodo de un modulo

T2 : Time\_Span := Milliseconds (200); -- Variable para almacenar el de periodo de un modulo

T3 : Time\_Span := Milliseconds (50); -- Variable para almacenar el de periodo de un modulo

task type CS;

type REF\_CS is access CS;

-- Tarea del campo solar

task body CS is

begin

    Tiempo1 := clock;

    for i in 1 .. reps loop

     delay until (Tiempo1);

     -- Añadimos el timepo de computo

     limite1 := Tiempo1 + C1;

     select

        delay until (limite1);

     then abort

        -- Ejecutamos los pasos que sigue el campo solar

        Modulos.CampoSolar;

     end select;

     -- Añadimos el periodo

     Tiempo1 := Tiempo1 + T1;

    end loop;

end;

task type MD;

type REF\_MD is access MD;

task body MD is

begin

    Tiempo2 := clock;

    for i in 1 .. reps loop

     delay until (Tiempo2);

     -- Añadimos el timepo de computo

     limite2 := Tiempo2 + C2;

     select

        delay until (limite2);

     then abort

        -- Ejecutamos los pasos que sigue el modulo de destilacion

        Modulos.ModuloDestilacion;

     end select;

     -- Añadimos el periodo

     Tiempo2 := Tiempo2 + T2;

    end loop;

end;

task type SS;

type REF\_SS is access SS;

task body SS is

begin

    Tiempo3 := clock;

    for i in 1 .. reps loop

     delay until (Tiempo3);

     limite3 := Tiempo3;

     select

        delay until (limite3);

     then abort

        -- Ejecutamos los pasos que sigue el sistema de seguridad

        Modulos.SistemaDeSeguridad;

     end select;

     -- Añadimos el periodo

     Tiempo3 := Tiempo3 + T3;

    end loop;

end;

begin

declare

    m\_cs : REF\_CS := new CS;

    m\_md : REF\_MD := new MD;

    m\_ss : REF\_SS := new SS;

begin

    null;

end;

end mainConcurrente;

Resultado de la ejecución (para una sola repetición)

CS: Leyendo temperatura: 1.00000E+01

CS: Controlando SC1(), estado de la bomba: ON

CS: Escribiendo temperatura: 1.00000E+01

CS: Enviando temperatura: 1.00000E+01

V: Escribiendo temperatura: 1.00000E+01

V: Escribiendo caudal: 2.50000E+01

V: Escribiendo radiacion: 1.50000E+00

MD: Leyendo caudal: 1.00000E+01

MD: Controlando SC2(), estado de la bomba: ON

MD: Escribiendo temperatura: 1.00000E+01

MD: Enviando temperatura: 1.00000E+01

V: Escribiendo temperatura: 1.00000E+01

V: Escribiendo caudal: 1.00000E+01

V: Escribiendo radiacion: 1.50000E+00

[2019-04-25 20:42:22] process terminated successfully, elapsed time: 01.96s

## 3.5. Conclusión

La principal desventaja de la concurrencia es que, al tener que asignar que proceso realiza en qué momento, se tarda más tiempo en terminar todas las tareas.

La ventaja que aporta es, sobre todo en sistemas en tiempo real, si tenemos un proceso que tarda mucho tiempo, digamos, 5 minutos, si no usamos concurrencia, el resto de los procesos están simplemente parados, y esto no nos conviene si tenemos que mantener constancia de muchos sensores a la vez o realizar diversos procesos.

En esta práctica es difícil de apreciar puesto que las tareas son muy rápidas, casi instantáneas, ya que solo escribiremos por consola, pero en casos reales, la concurrencia es mucho más útil.

# Anexo. Descripción del problema: Control de una instalación de destilación por membranas

En la Figura 1 se presenta el diagrama esquemático de una instalación de destilación por membranas alimentada con energía solar. Esta instalación se basa en una tecnología llamada destilación por membranas, en la cual, a través de un proceso de evaporación se produce agua desalada usando como alimentación agua de mar.

En este tipo de instalaciones se requieren dos procesos fundamentales de control. El primero consiste en el control de la temperatura de salida del campo solar (ST2 en la Fig. 1). Mantener una temperatura adecuada a la salida del campo es determinante para lograr un óptimo funcionamiento de la planta. En este tipo de sistemas la señal de control es el caudal de entrada del campo solar (SC1 en la Fig. 1), el cual se debe ir variando en función de los niveles de irradiancia medidos a través del sensor de radiación (SR1 en la Fig. 1) y las condiciones de operación del campo solar las cuales son la temperatura ambiente, la cual se mide a través del sensor ST4, y la temperatura de entrada al campo, medida a través del sensor ST1. La temperatura óptima de operación del campo solar para esta operación es de 80oC.

La segunda tarea de control consiste en mantener un nivel de producción de destilado determinado, el cual se mide con el sensor de destilado SD1. En este caso la señal de control es el caudal de entrada al módulo de destilación (SC2 en la Fig. 1), el cual debe ir variando en función de la temperatura del agua de mar, medida con el sensor ST3 y la temperatura que le llega desde el campo (ST2). El nivel deseado de producción es de 25 L/h.

Además, este proceso industrial está sujeto a algunas restricciones, las cuales deben tenerse en cuenta como medidas de seguridad. La primera consiste en que la temperatura a la salida del campo no debe ser mayor de 100oC, ya que el fluido es agua y a esa temperatura formaría vapor. Por tanto, si se detecta que la temperatura es mayor de 100oC, se debe parar la bomba 1. Del mismo modo, la temperatura a la entrada del módulo MD, no puede ser mayor de 85oC, si esto ocurre, se debe parar la bomba 2. Por último, como en casi todas las plantas industriales, se requiere una visualización por pantalla de los estados de todos los actuadores de la instalación. También se requiere un almacenamiento histórico de toda esta información.

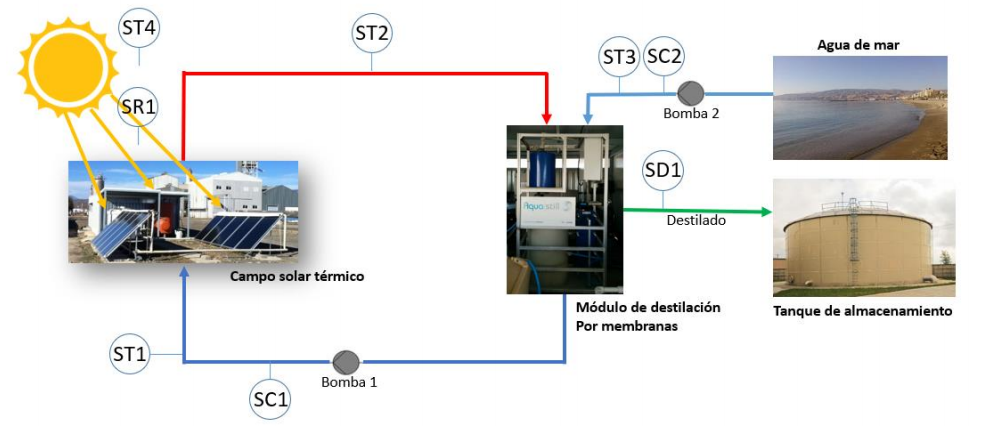


Figura 1. Diagrama esquemático de la instalación de destilación por membranas. ST son sensores de temperatura [oC], SC son sensores de caudal [L/h], SR es un sensor de radiación [W/m2], y SD es un sensor de destilado [L/h].

De la descripción anterior se deduce que las tareas específicas a realizar son:

* **Sistema de control del campo solar**: Se encarga de controlar la temperatura del campo solar (ST2) actuando sobre el caudal de la bomba 1 (SC1). Esta tarea se debe ejecutar cada 100 ms y tiene un tiempo de cómputo de 10 ms.
* **Sistema de control del módulo MD**: Se encarga de controlar la producción de destilado (SD1) actuando sobre el caudal de la bomba 2 (SC2). Esta tarea se debe ejecutar cada 200 ms y tiene un tiempo de cómputo de 10 ms.
* **Sistema de seguridad**: Esta tarea se encarga de asegurar los límites de operación del módulo MD y del campo solar. Esta tarea se debe ejecutar cada 50 ms.
* **Almacenamiento de datos**: Contiene la información de todos los sensores de la planta y se encarga de almacenar en una base de datos el estado de los sensores y actuadores utilizados. El problema que tiene dicha base de datos es que solo permite el acceso de una tarea a la vez, ya sea en modo escritura o lectura, por lo que deberá ser implementada como un recurso compartido con exclusión mutua.
* **Visualización por pantalla**: Se encarga de mostrar por pantalla la información de los sensores del sistema, así como el estado de los actuadores. Esta tarea deberá ser implementada como una tarea servidora que espera el envío de mensajes (citas) desde otras tareas para actualizar la información por pantalla. Se supone que el tiempo que consume cada actividad de la cita es despreciable.