Ada for Automation Demo Application

142 a4a_k0_S7

Stéphane LOS

Version 2022.05, 2022-05-31

Table of Contents

Chapter 1. Description

1.1. Ada for Automation

Ada for Automation (A4A in short) is a framework for designing industrial automation applications using the Ada language.

It makes use of the libmodbus library to allow building a ModbusTCP client or server, or a Modbus RTU master or slave.

It can also use Hilscher communication boards allowing to communicate on field buses like AS-Interface, CANopen, CC-Link, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, PROFINET, Sercos III, POWERLINK, or VARAN.

With the help of GtkAda, the binding to the Graphic Tool Kit, one can design Graphical User Interfaces.

Thanks to Gnoga, built on top of Simple Components, it is also possible to provide a Web User Interface.

Nice addition is the binding to the Snap7 library which allows to communicate with SIEMENS S7 PLCs using S7 Communication protocol ISO on TCP (RFC1006).

Of course, all the Ada ecosystem is available.

Using Ada bindings, C, C++, Fortran libraries can also be used.

And, since it is Ada, it can be compiled using the same code base to target all major platforms.

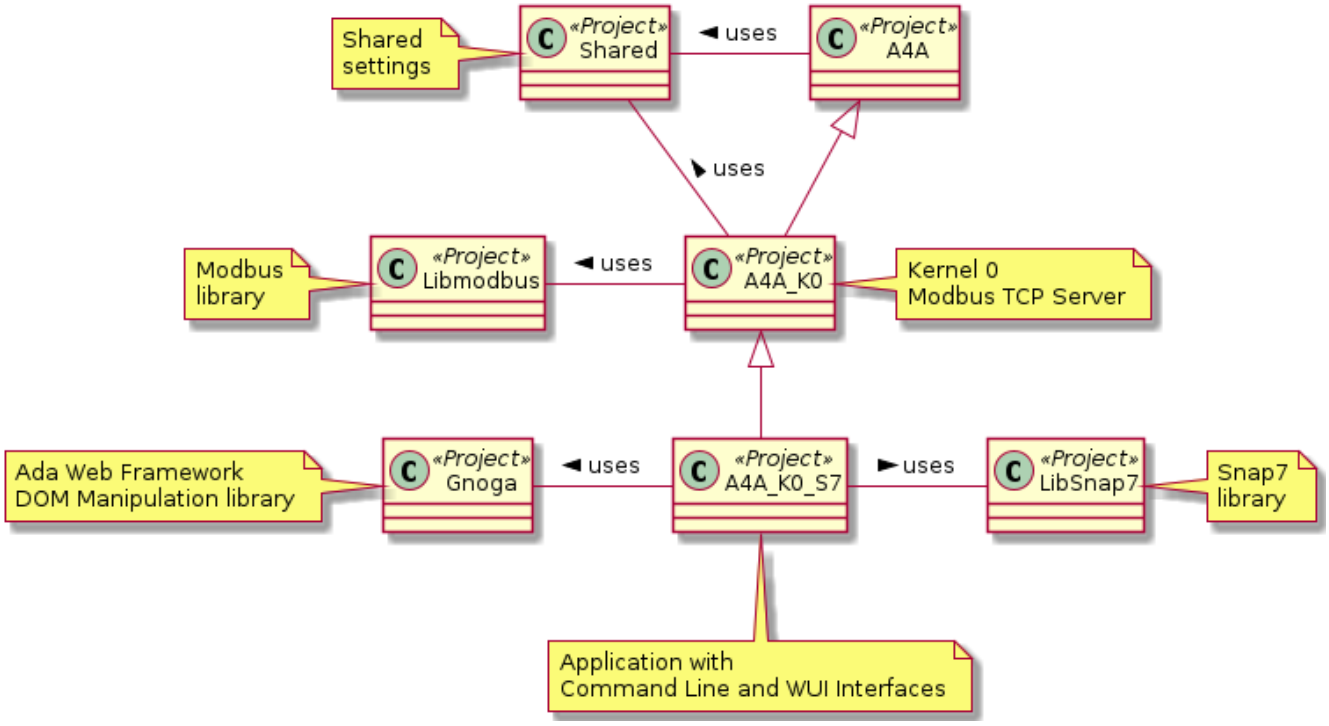
1.2. This demo application

This is a demo application featuring:

- a basic command line interface,
- a basic web user interface making use of Gnoga,
- a kernel with a Modbus TCP Server (K0),
- a gateway application with a web interface that connects to a SIEMENS S7 PLC.

Chapter 2. Projects diagram

The following picture shows the diagram of projects :



Chapter 3. License

Those files are included in the **Ada for Automation** root folder :

COPYING3
The GPL License you should read carefully. GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

COPYING.RUNTIME
GCC RUNTIME LIBRARY EXCEPTION Version 3.1, 31 March 2009

Hence, each source file contains the following header :

```
------------------------------------------------------------------------------
--                            Ada for Automation                            --
--                                                                          --
--                   Copyright (C) 2012-2023, Stephane LOS                  --
--                                                                          --
-- This library is free software;  you can redistribute it and/or modify it --
-- under terms of the  GNU General Public License  as published by the Free --
-- Software  Foundation;  either version 3,  or (at your  option) any later --
-- version. This library is distributed in the hope that it will be useful, --
-- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
-- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
--                                                                          --
-- As a special exception under Section 7 of GPL version 3, you are granted --
-- additional permissions described in the GCC Runtime Library Exception,   --
-- version 3.1, as published by the Free Software Foundation.               --
--                                                                          --
-- You should have received a copy of the GNU General Public License and    --
-- a copy of the GCC Runtime Library Exception along with this program;     --
-- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
-- <http://www.gnu.org/licenses/>.                                          --
--                                                                          --
------------------------------------------------------------------------------
```

Chapter 4. Building

The provided makefile uses GPRbuild and provides six targets:

- all : builds the executable,
- app_doc : creates the documentation of the source code,
- clean : cleans the space.

Additionally one can generate some documentation using Asciidoctor with :

- read_me_html : generates the README in HTML format,
- read_me_pdf : generates the README in PDF format,
- read_me : generates the README in both formats.

Chapter 5. Running

Of course, this application is of interest only if a Modbus TCP Client application is talking to it.

Good candidates are a SCADA or a PLC but, if none is at your disposal, you could use one of :

- 020 a4a-k1-cli,
- 021 a4a-k1-gui,
- 022 a4a-k1-wui,
- your own.

In a console:

Build the application:

```
make
```

Optionally create the documentation:

```
make app_doc
```

Run the application:

```
make run
```

Use Ctrl+C to exit.

Optionally clean all:

```
make clean
```

Chapter 6. Directories

bin
Where you will find the executable.

doc
The place where GNATdoc would create the documentation.

obj
Build artifacts go here.

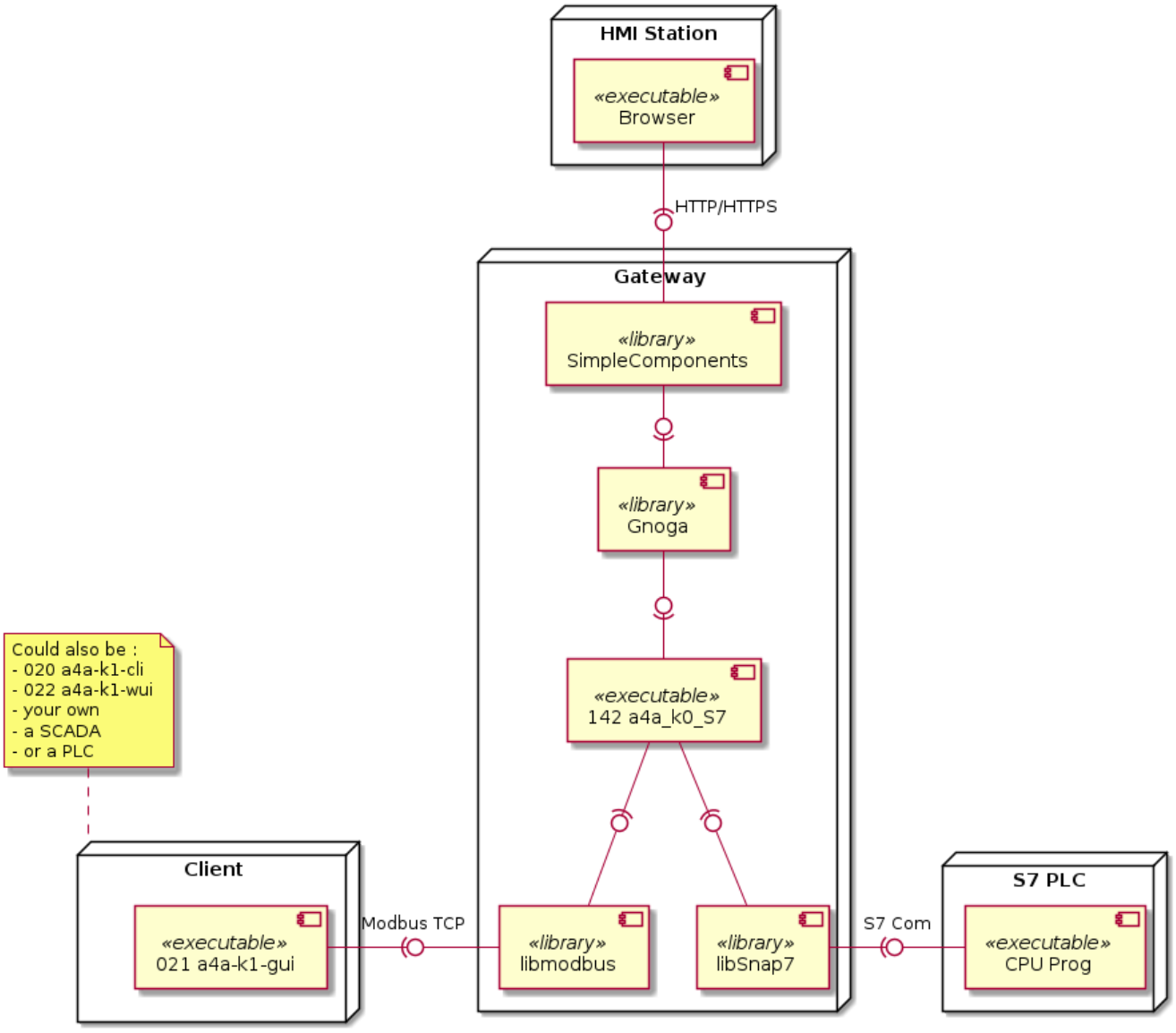
src
Application source files.

Chapter 7. Application

This is a gateway application with a web interface that connects to a SIEMENS S7 PLC a Modbus TCP Server.

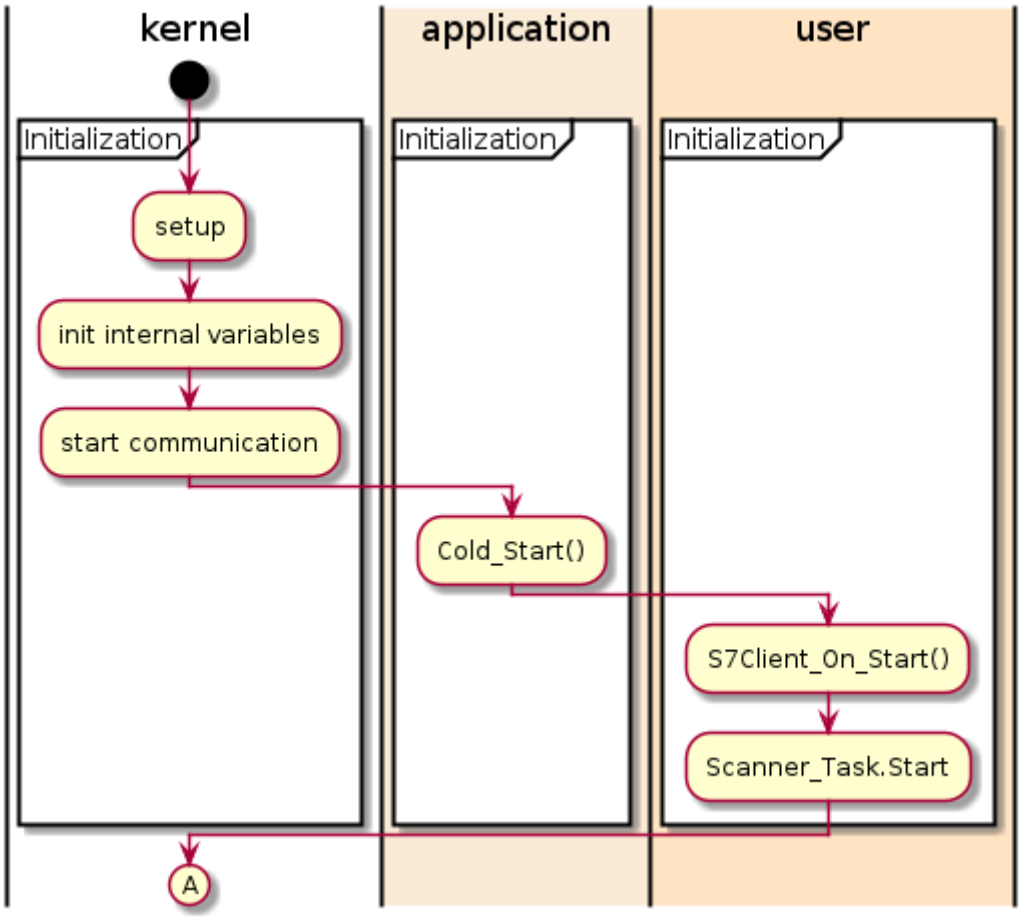
It has a Command Line and Web User Interfaces and listen to Modbus TCP requests.

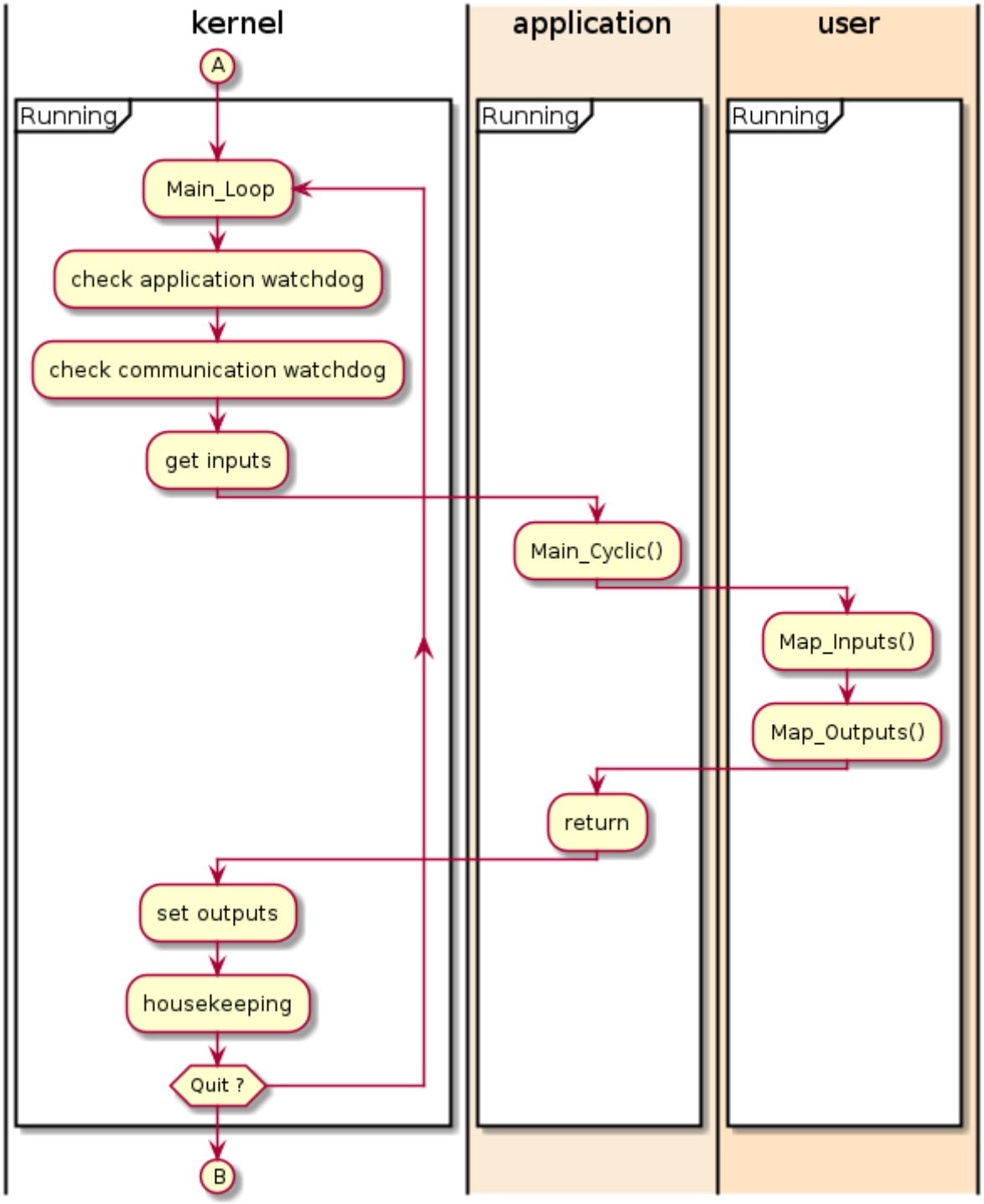
7.1. Deployment diagram

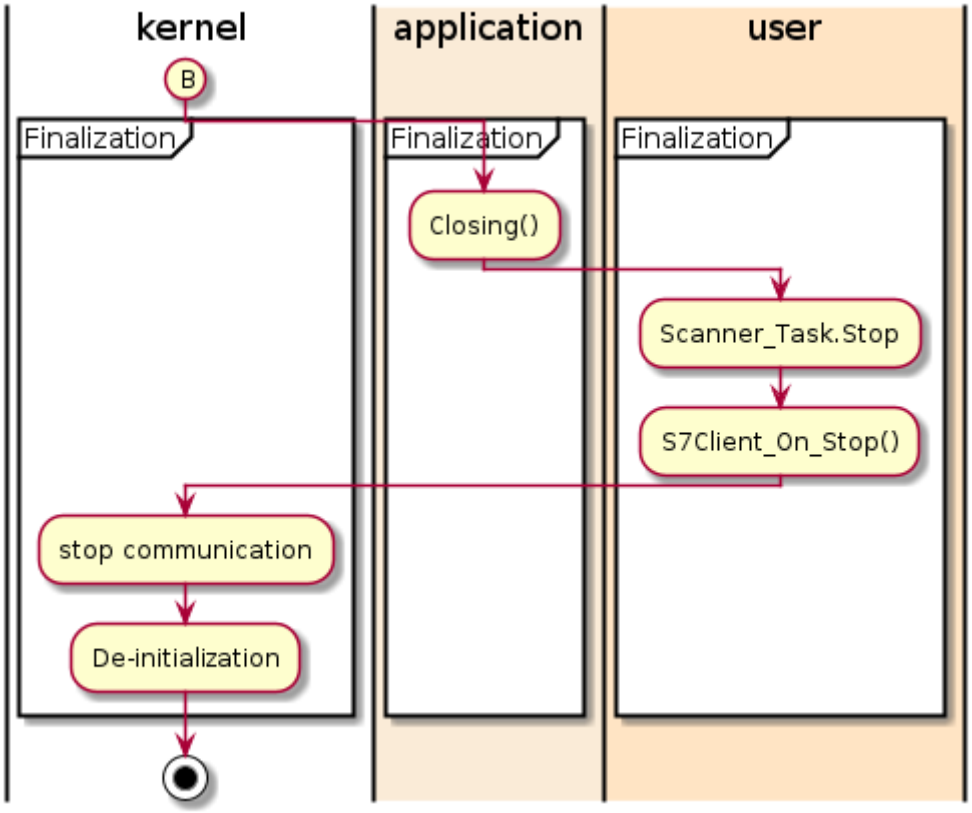


7.2. Activity diagram

The Kernel manages the communication channel and provides an interface to it, namely the package "A4A.Memory.MBTCP_IOSlave".




kernel
application
user
A
Running
Running
Running
Main_Loop
check application watchdog
check communication watchdog
get inputs
Main_Cyclic()
Map_Inputs()
Map_Outputs()
return
set outputs
housekeeping
Quit ?
B


kernel
application
user
B
Finalization
Finalization
Finalization
Closing()
Scanner_Task.Stop
S7Client_On_Stop()
stop communication
De-initialization

7.3. Modbus TCP Server Configuration

"./src/a4a-application-mbtcp_server_config.ads"

```
package A4A.Application.MBTCP_Server_Config is

   ------------------------------------------------------------------------
   --  Modbus TCP Server configuration
   ------------------------------------------------------------------------

   package Server is new A4A.MBTCP_Server
     (Coils_Number           => 65536,
      Input_Bits_Number      => 65536,
      Input_Registers_Number => 65536,
      Registers_Number       => 65536);

   Config1 : aliased Server.Server_Configuration :=
     (Server_Enabled          => True,
      Debug_On                => False,
      Retries                 => 3,
      Server_IP_Address       => To_Bounded_String ("127.0.0.1"),
      Server_TCP_Port         => 1504); --  ①

end A4A.Application.MBTCP_Server_Config;
```

① Modbus TCP Server port : 1504

7.4. S7 Client Configuration

"./src/a4a-application-s7_client_config.ads"

```
package A4A.Application.S7_Client_Config is

   ------------------------------------------------------------------
   --  S7 Client configuration
   ------------------------------------------------------------------
   Remote_IP_Address : String := "195.13.34.87";
   Local_IP_Address  : String := "192.168.253.240";

   Server_IP_Address : Bounded_String :=
    To_Bounded_String (Remote_IP_Address);    --  ①
   Server_TCP_Port   : Positive := 102;      --  ②

   CPU_Rack       : Natural  := 0;           --  ③
   CPU_Slot       : Natural  := 3;           --  ④
end A4A.Application.S7_Client_Config;
```

① S7 Client IP Address,

② S7 Client IP port, 102 is ISO on TCP / RFC 1006 std

③ CPU Rack

④ CPU Slot

7.5. User objects Definition

"./src/a4a-user_objects.ads"

```
with A4A.Protocols.LibSnap7; use A4A.Protocols.LibSnap7;

with Interfaces.C; use Interfaces.C;

package A4A.User_Objects is

   ------------------------------------------------------------------
   --  User Objects creation
   ------------------------------------------------------------------

   ------------------------------------------------------------------
   --  S7 Communication
   ------------------------------------------------------------------

   Client       : aliased S7Object;
   Result       : C.int := 0;
   IsConnected  : C.int := 0;
   Connected    : Boolean := False;
   Status       : C.int := 0;
   Running      : Boolean := False;

   Order_Code   : Order_Code_Type;        --  ①
   Order_Code_Got : Boolean := False;

   CPU_Info     : CPU_Info_Type;          --  ②
   CPU_Info_Got : Boolean := False;

   MyUsrDataIn  : aliased Byte_Array (0 .. 19) := (others => 0);
   --  Input Data Buffer

   MyUsrDataOut : Byte_Array (0 .. 19) := (others => 0);
   --  Output Data Buffer

   MW0          : Word  := 0;             --  ③
   MW2          : Word  := 0;
   MW4          : Word  := 0;
   MD8          : DWord := 0;
   MD8F         : Float := 0.0;
   IW0          : Word  := 0;
   QW0          : Word  := 0;
   DB1_W0       : Word  := 0;
   T1           : Word  := 0;
   C1           : Word  := 0;

   MW4_New      : Word  := 0;
   MW4_Write    : Boolean := False;
```

```
    MD8F_New     : Float := 0.0;
    MD8F_Write   : Boolean := False;
    --  S7 PLC Data Read / Write

end A4A.User_Objects;
```

① CPU Order code.

② CPU Information.

③ Application Data.

7.6. User Functions

"./src/a4a-user_functions.adb"

```
package body A4A.User_Functions is
   My_Ident : constant String := "A4A.User_Functions";

   ---------------------------------------------------------------------
   --  User functions
   ---------------------------------------------------------------------

   procedure Map_Inputs is  --  ①
   begin

      null;

   end Map_Inputs;

   procedure Map_Outputs is --  ②
   begin

      MBTCP_IOServer.Input_Registers (0) := MW0;
      MBTCP_IOServer.Input_Registers (1) := MW2;
      MBTCP_IOServer.Input_Registers (2) := MW4;

      DWord_To_Words (DWord_in => MD8,
                      LSW_Word => MBTCP_IOServer.Input_Registers (4),
                      MSW_Word => MBTCP_IOServer.Input_Registers (3));

      MBTCP_IOServer.Input_Registers (5) := IW0;
      MBTCP_IOServer.Input_Registers (6) := QW0;
      MBTCP_IOServer.Input_Registers (7) := DB1_W0;

      MBTCP_IOServer.Input_Registers (8) := T1;
      MBTCP_IOServer.Input_Registers (9) := C1;
   end Map_Outputs;

   procedure Map_HMI_Inputs is
   begin

      null;

   end Map_HMI_Inputs;

   procedure Map_HMI_Outputs is
   begin

      null;

   end Map_HMI_Outputs;
```

```
   procedure S7Client_On_Start is --  ③
   begin

      A4A.Log.Logger.Put
        (Who       => My_Ident & ".S7Client_On_Start",
         What      => "Cli_Create",
         Log_Level => Level_Info);

      Client := Cli_Create;

   end S7Client_On_Start;

   procedure S7Client_On_Stop is --  ④
   begin

      if Connected then
         Result := Cli_Disconnect (Client => Client);

         if Result = 0 then
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_On_Stop",
               What      => "Disconnected !",
               Log_Level => Level_Info);

            Connected := False;
         else
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_On_Stop",
               What      => "Not Disconnected ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);
         end if;

      end if;

      Cli_Destroy (Client'Access);

   end S7Client_On_Stop;

   procedure S7Client_On_Error is --  ⑤
   begin

      if Connected then
         Result := Cli_Disconnect (Client => Client);

         if Result = 0 then
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_On_Error",
               What      => "Disconnected !",
               Log_Level => Level_Info);
```

```
         Connected := False;
      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".S7Client_On_Error",
            What      => "Not Disconnected ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);
      end if;

   end if;

end S7Client_On_Error;

procedure S7Client_Test is --  ⑥
   use A4A.Application.S7_Client_Config;
begin

   Result := Cli_GetConnected (Client      => Client,
                               IsConnected => IsConnected);

   if Result /= 0 then
      A4A.Log.Logger.Put
        (Who       => My_Ident & ".S7Client_Test",
         What      => "Cli_GetConnected "
         & "Result = " & Cli_Error_Text (Error => Result),
         Log_Level => Level_Error);
   end if;

   if not Connected then
      Result := Cli_ConnectTo
        (Client     => Client,
         IP_Address => To_String (Server_IP_Address),
         Rack       => CPU_Rack,
         Slot       => CPU_Slot);

      if Result = 0 then
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".S7Client_Test",
            What      => "Connected !",
            Log_Level => Level_Info);

         Connected := True;
      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".S7Client_Test",
            What      => "Not Connected ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;
```

```
      end if;

      if Connected then

         Result := Cli_GetPlcStatus (Client  => Client, Status => Status);

         if Result = 0 then
            Running := (Status = 16#08#);
         else
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_Test",
               What      => "Cli_GetPlcStatus failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

      if Connected and not Order_Code_Got then

         Cli_Get_Order_Code (Client     => Client,
                             Order_Code => Order_Code,
                             Result     => Result);

         if Result = 0 then
            Order_Code_Got := True;
         else
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_Test",
               What      => "Cli_Get_Order_Code failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

      if Connected and not CPU_Info_Got then

         Cli_Get_Cpu_Info (Client   => Client,
                           Cpu_Info => CPU_Info,
                           Result   => Result);

         if Result = 0 then
            CPU_Info_Got := True;
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".S7Client_Test",
```

```
            What      => "Cli_Get_Cpu_Info : Serial_Number : "
            & CPU_Info.Serial_Number,
            Log_Level => Level_Info);
      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".S7Client_Test",
            What      => "Cli_Get_Cpu_Info failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

   Test_Mementos;

   Test_Inputs;

   Test_Outputs;

   Test_DBs;

   Test_Timers;

   Test_Counters;

end S7Client_Test;

-----------------------------------------------------------------------
--  Mementos Read / Write tests
-----------------------------------------------------------------------

procedure Test_Mementos is

begin

   if Connected then

      MyUsrDataIn := (others => 0);

      Result := Cli_MBRead (Client  => Client,
                            Start   => 0,
                            Size    => MyUsrDataIn'Length,
                            UsrData => MyUsrDataIn);

      if Result = 0 then

         Bytes_To_Word (LSB_Byte => MyUsrDataIn (1),
                        MSB_Byte => MyUsrDataIn (0),
                        Word_out => MW0);
```

```
         Bytes_To_Word (LSB_Byte => MyUsrDataIn (3),
                        MSB_Byte => MyUsrDataIn (2),
                        Word_out => MW2);

         Bytes_To_Word (LSB_Byte => MyUsrDataIn (5),
                        MSB_Byte => MyUsrDataIn (4),
                        Word_out => MW4);

         Bytes_To_DWord (Byte0 => MyUsrDataIn (11),
                         Byte1 => MyUsrDataIn (10),
                         Byte2 => MyUsrDataIn (9),
                         Byte3 => MyUsrDataIn (8),
                         DWord_out => MD8);

         MD8F := DWord_To_Float (MD8);

      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_Mementos",
            What      => "Read Mementos failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

   if Connected then

      MyUsrDataOut (0 .. 3) := MyUsrDataIn (8 .. 11);

      Result := Cli_MBWrite (Client  => Client,
                             Start   => MyUsrDataIn'Length,
                             Size    => MyUsrDataOut'Length,
                             UsrData => MyUsrDataOut);

      if Result /= 0 then
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_Mementos",
            What      => "Write Mementos failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

   if Connected and MW4_Write then
```

```
      Word_To_Bytes (Word_in  => MW4_New,
                     LSB_Byte => MyUsrDataOut (1),
                     MSB_Byte => MyUsrDataOut (0));

      Result := Cli_MBWrite (Client  => Client,
                             Start   => 4,
                             Size    => 2,
                             UsrData => MyUsrDataOut);

      if Result /= 0 then
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_Mementos",
            What      => "Write MW4 failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      else
         MW4_Write := False;
      end if;

   end if;

   if Connected and MD8F_Write then

      DWord_To_Bytes (DWord_in  => Float_To_DWord (MD8F_New),
                      Byte0 => MyUsrDataOut (3),
                      Byte1 => MyUsrDataOut (2),
                      Byte2 => MyUsrDataOut (1),
                      Byte3 => MyUsrDataOut (0));

      Result := Cli_MBWrite (Client  => Client,
                             Start   => 8,
                             Size    => 4,
                             UsrData => MyUsrDataOut);

      if Result /= 0 then
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_Mementos",
            What      => "Write MD8F failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      else
         MD8F_Write := False;
      end if;

   end if;
```

```
   end Test_Mementos;

   ----------------------------------------------------------------------
   --  Inputs Read / Write tests
   ----------------------------------------------------------------------

   procedure Test_Inputs is

   begin

      if Connected then

         MyUsrDataIn := (others => 0);

         Result := Cli_EBRead (Client  => Client,
                               Start   => 0,
                               Size    => MyUsrDataIn'Length,
                               UsrData => MyUsrDataIn);

         if Result = 0 then
            Bytes_To_Word (LSB_Byte => MyUsrDataIn (1),
                           MSB_Byte => MyUsrDataIn (0),
                           Word_out => IW0);

            MyUsrDataOut (0 .. 3) := MyUsrDataIn (0 .. 3);
         else
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".Test_Inputs",
               What      => "Read Inputs failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

      if Connected then

         Result := Cli_EBWrite (Client  => Client,
                                Start   => 10,
                                Size    => MyUsrDataOut'Length,
                                UsrData => MyUsrDataOut);

         if Result /= 0 then
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".Test_Inputs",
               What      => "Write Inputs failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);
```

```
            S7Client_On_Error;
         end if;

      end if;

   end Test_Inputs;

   ----------------------------------------------------------------------
   --  Outputs Read / Write tests
   ----------------------------------------------------------------------

   procedure Test_Outputs is

   begin

      if Connected then

         MyUsrDataIn := (others => 0);

         Result := Cli_ABRead (Client  => Client,
                               Start   => 0,
                               Size    => MyUsrDataIn'Length,
                               UsrData => MyUsrDataIn);

         if Result = 0 then
            Bytes_To_Word (LSB_Byte => MyUsrDataIn (1),
                           MSB_Byte => MyUsrDataIn (0),
                           Word_out => QW0);

            MyUsrDataOut (0 .. 3) := MyUsrDataIn (0 .. 3);
         else
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".Test_Outputs",
               What      => "Read Outputs failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

      if Connected then

         Result := Cli_ABWrite (Client  => Client,
                                Start   => 10,
                                Size    => MyUsrDataOut'Length,
                                UsrData => MyUsrDataOut);

         if Result /= 0 then
            A4A.Log.Logger.Put
```

```
            (Who       => My_Ident & ".Test_Outputs",
             What      => "Write Outputs failed ! "
             & "Result = " & Cli_Error_Text (Error => Result),
             Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

end Test_Outputs;

--------------------------------------------------------------------
--  Data Blocks Read / Write tests
--------------------------------------------------------------------

procedure Test_DBs is

begin

   if Connected then

      MyUsrDataIn := (others => 0);

      Result := Cli_DBRead (Client   => Client,
                            DBNumber => 1,
                            Start    => 0,
                            Size     => MyUsrDataIn'Length,
                            UsrData  => MyUsrDataIn);

      if Result = 0 then
         Bytes_To_Word (LSB_Byte => MyUsrDataIn (1),
                        MSB_Byte => MyUsrDataIn (0),
                        Word_out => DB1_W0);

         MyUsrDataOut (0 .. 3) := MyUsrDataIn (0 .. 3);
      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_DBs",
            What      => "Read Data Block failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

   if Connected then

      Result := Cli_DBWrite (Client   => Client,
```

```
                              DBNumber => 1,
                              Start    => 10,
                              Size     => MyUsrDataOut'Length,
                              UsrData  => MyUsrDataOut);

      if Result /= 0 then
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_DBs",
            What      => "Write Data Block failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
      end if;

   end if;

end Test_DBs;

-----------------------------------------------------------------------
--  Timers Read / Write tests
-----------------------------------------------------------------------

procedure Test_Timers is

begin

   if Connected then

      MyUsrDataIn := (others => 0);

      Result := Cli_TMRead (Client  => Client,
                            Start   => 0,
                            Amount  => MyUsrDataIn'Length / 2,
                            UsrData => MyUsrDataIn);

      if Result = 0 then
         Bytes_To_Word (LSB_Byte => MyUsrDataIn (3),
                        MSB_Byte => MyUsrDataIn (2),
                        Word_out => T1);

         MyUsrDataOut (0 .. 3) := MyUsrDataIn (0 .. 3);
         MyUsrDataOut (3) := MyUsrDataIn (3) and 16#F0#; -- Let's play !
      else
         A4A.Log.Logger.Put
           (Who       => My_Ident & ".Test_Timers",
            What      => "Read Timers failed ! "
            & "Result = " & Cli_Error_Text (Error => Result),
            Log_Level => Level_Error);

         S7Client_On_Error;
```

```
         end if;

      end if;

      if Connected and False then

         Result := Cli_TMWrite (Client  => Client,
                                Start   => 0,
                                Amount  => MyUsrDataOut'Length / 2,
                                UsrData => MyUsrDataOut);

         if Result /= 0 then
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".Test_Timers",
               What      => "Write Timers failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

   end Test_Timers;

   ----------------------------------------------------------------------
   --  Counters Read / Write tests
   ----------------------------------------------------------------------

   procedure Test_Counters is

   begin

      if Connected then

         MyUsrDataIn := (others => 0);

         Result := Cli_CTRead (Client  => Client,
                               Start   => 0,
                               Amount  => MyUsrDataIn'Length / 2,
                               UsrData => MyUsrDataIn);

         if Result = 0 then
            Bytes_To_Word (LSB_Byte => MyUsrDataIn (3),
                           MSB_Byte => MyUsrDataIn (2),
                           Word_out => C1);

            MyUsrDataOut (0 .. 2) := MyUsrDataIn (0 .. 2);
            MyUsrDataOut (3) := MyUsrDataIn (3) and 16#F0#; -- Let's play !
         else
            A4A.Log.Logger.Put
```

```
               (Who       => My_Ident & ".Test_Counters",
                What      => "Read Counters failed ! "
                & "Result = " & Cli_Error_Text (Error => Result),
                Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

      if Connected and False then

         Result := Cli_CTWrite (Client  => Client,
                                Start   => 0,
                                Amount  => MyUsrDataOut'Length / 2,
                                UsrData => MyUsrDataOut);

         if Result /= 0 then
            A4A.Log.Logger.Put
              (Who       => My_Ident & ".Test_Counters",
               What      => "Write Counters failed ! "
               & "Result = " & Cli_Error_Text (Error => Result),
               Log_Level => Level_Error);

            S7Client_On_Error;
         end if;

      end if;

   end Test_Counters;

end A4A.User_Functions;
```

User functions are defined to :

① get the inputs from the Modbus TCP server,

② set Modbus TCP server ouputs,

③ create the S7 Client on start,

④ disconnect and destroy the S7 Client on stop,

⑤ disconnect the S7 Client on error,

⑥ test features of the S7 Client.

7.7. S7 Client auxilliary task

"./src/a4a-application-s7_client.adb"

```
with A4A.User_Functions; use A4A.User_Functions;

package body A4A.Application.S7_Client is

   task body Scanner is
      My_Ident : constant String := "A4A.Application.S7_Client.Scanner";
   begin

      accept Start;

      loop

         S7Client_Test;    --  ①

         select
            accept Stop;
            A4A.Log.Logger.Put (Who       => My_Ident,
                                What      => "I'm dead !",
                                Log_Level => Level_Info);
            exit;
         or
            delay 0.5;     --  ②
         end select;

      end loop;

   exception

      when Error : others =>
         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => Exception_Information (Error));

         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => "Aborted !");
   end Scanner;

end A4A.Application.S7_Client;
```

This task calls the test function repeatedly :

① S7 Client test,

② with a period of 500 ms.

7.8. User Application

"./src/a4a-application.adb"

```
package body A4A.Application is

   Scanner_Task : A4A.Application.S7_Client.Scanner;

   Client_Task : A4A.Application.MQTT_Client.Client_Task_Type;

   procedure Cold_Start is
   begin

      S7Client_On_Start; --  ③

      Scanner_Task.Start;

      MQTTClient_On_Start; --  ⑤

      Client_Task.Start;

   end Cold_Start;

   procedure Closing is
   begin

      Scanner_Task.Stop;

      S7Client_On_Stop; --  ④

      Client_Task.Stop;

      MQTTClient_On_Stop; --  ⑥

   end Closing;

   procedure Main_Cyclic is
      My_Ident : constant String := "A4A.Application.Main_Cyclic";
   begin
      A4A.Log.Logger.Put
        (Who       => My_Ident,
         What      => "Yop ! **********************",
         Log_Level => Level_Verbose);

      Map_Inputs;  --  ①

      Map_HMI_Inputs;

      --  Playing with tasks interface
      Main_Outputs.X := Main_Inputs.A;
      Main_Outputs.Y := Main_Inputs.B;
```

```
      Main_Outputs.Z := Main_Inputs.C;

      Map_Outputs;  --  ②

      Map_HMI_Outputs;

   exception

      when Error : others =>
         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => Exception_Information (Error));

         Program_Fault_Flag := True;

   end Main_Cyclic;

   procedure Periodic1_Cyclic is
      My_Ident : constant String := "A4A.Application.Periodic1_Cyclic";
   begin
      A4A.Log.Logger.Put (Who       => My_Ident,
                          What      => "Hi !",
                          Log_Level => Level_Verbose);

      --  Do something useful here
      --  Could be simulate

      --  Playing with tasks interface
      Periodic1_Outputs.A := not Periodic1_Inputs.X;
      Periodic1_Outputs.B := Periodic1_Inputs.Y + 2;
      Periodic1_Outputs.C := Periodic1_Inputs.Z + 1;

   exception

      when Error : others =>
         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => Exception_Information (Error));

         Program_Fault_Flag := True;

   end Periodic1_Cyclic;

   function Program_Fault return Boolean is
   begin
      return Program_Fault_Flag;
   end Program_Fault;

end A4A.Application;
```

The application cyclically :

① gets the inputs from the server,

② sets server ouputs.

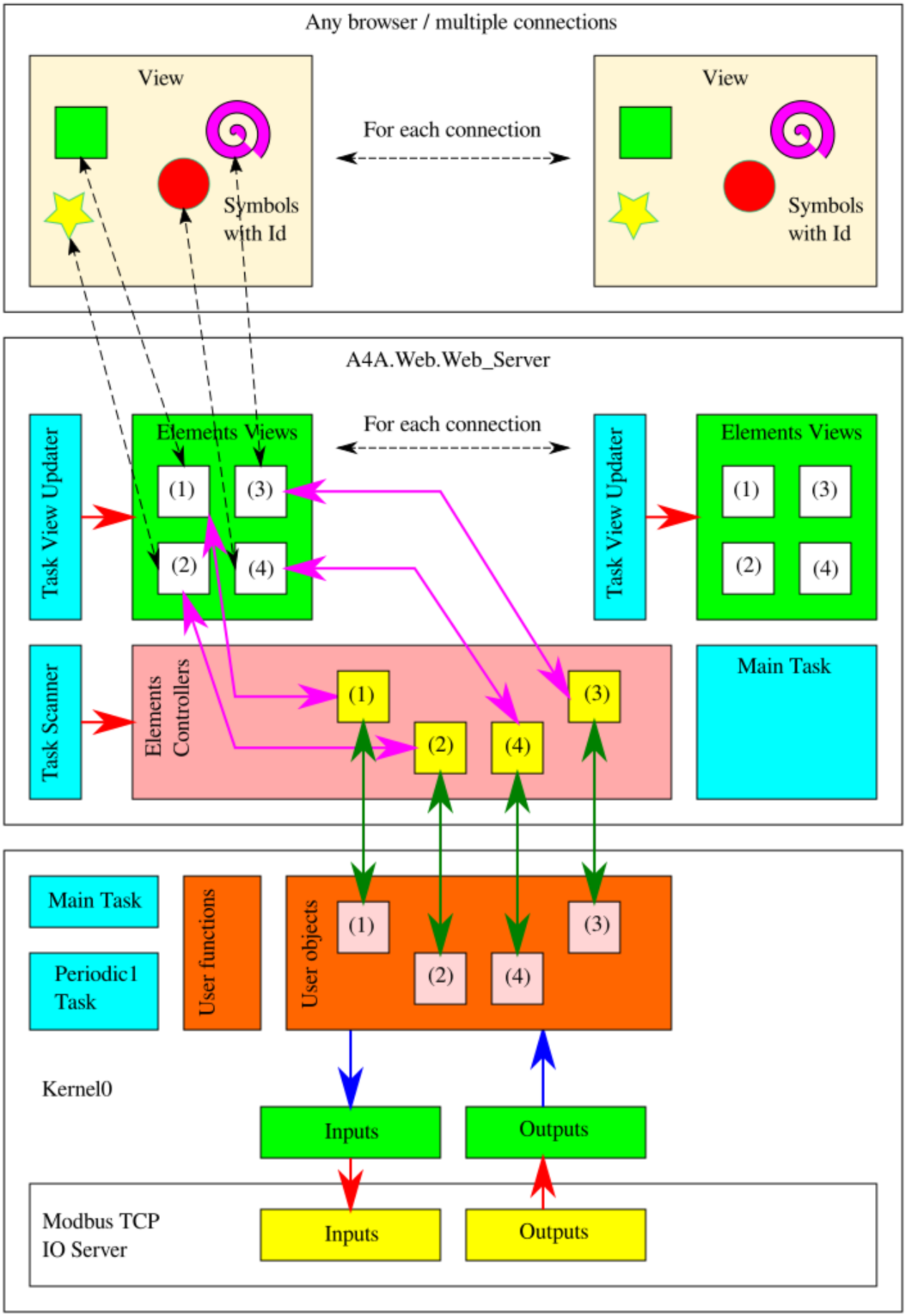
③ creates the S7 Client and starts Scanner task on start.

④ stops Scanner task, disconnects and destroys the S7 Client on stop.

7.9. Web server and User Interface

Hereafter is a diagram showing architecture and information flow for the Web UI.

An article is available, in French though :
https://slo-ist.fr/ada4automation/a4a-modbus-tcp-server-web-hmi-a4a_piano


Any browser / multiple connections
View
Symbols with Id
For each connection
View
Symbols with Id
A4A.Web.Web_Server
Task View Updater
Elements Views
(1)
(3)
(2)
(4)
For each connection
Task View Updater
Elements Views
(1)
(3)
(2)
(4)
Task Scanner
Elements Controllers
(1)
(3)
(2)
(4)
Main Task
Main Task
Periodic1 Task
User functions
User objects
(1)
(3)
(2)
(4)
Kernel0
Inputs
Outputs
Modbus TCP IO Server
Inputs
Outputs

7.10. MQTT Client

TO DO !