

Take Off!



Ada for Automation

Freedom and Power for Control Engineers

Ada for Automation Demo Application

001 a4a-k0b-gui

Stéphane LOS

Version 2022.05, 2022-05-31

Table of Contents

1. Description	1
1.1. Ada for Automation	1
1.2. This demo application	1
2. Projects diagram	2
3. License	3
4. Building	4
5. Running	5
6. Directories	6
7. Application	7
7.1. Deployment diagram	7
7.2. Activity diagram	8
7.3. Modbus RTU Slave Configuration	11
7.4. User objects Definition	12
7.5. User Functions	13
7.6. User Application	14

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

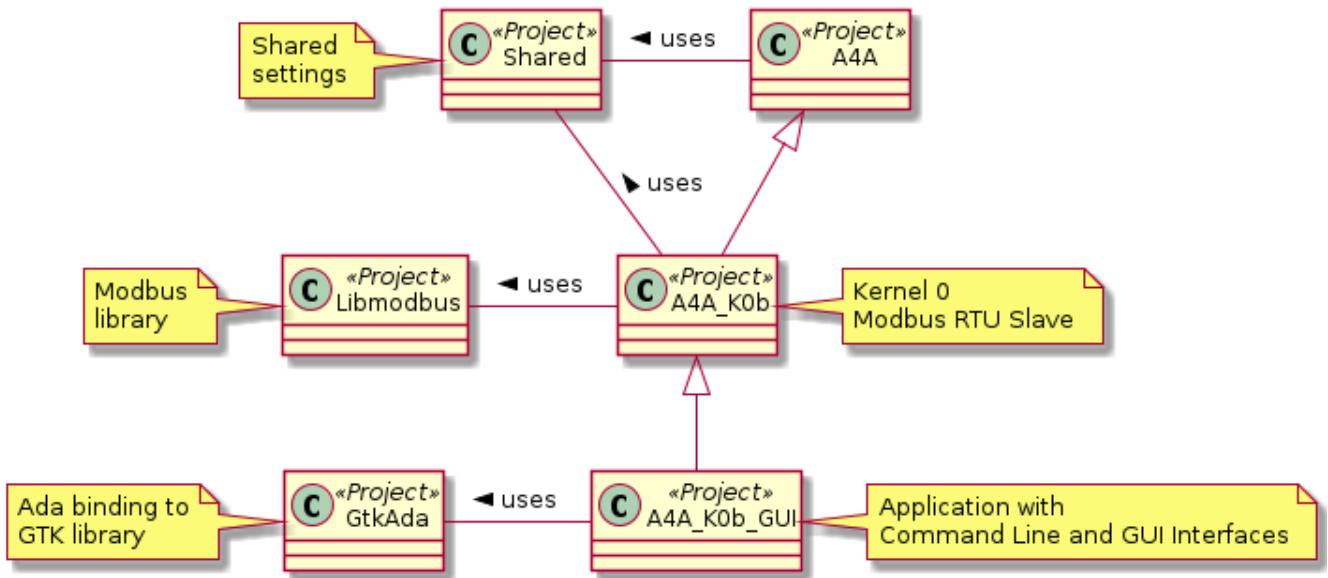
This is a demo application featuring:

- a basic command line interface,
- a basic graphic user interface making use of GtkAda,
- a kernel with a Modbus RTU Slave (K0b),
- a trivial application mirroring inputs in outputs.

It shares most of the source code of "000 a4a-k0b-cli".

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 RTU Master 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 :

- 080 app3-cli,
- 081 app3-gui,
- 082 app3-wui,
- your own.

In a console:

Build the application:

```
make
```

Optionally create the documentation:

```
make app_doc
```

Run the application:

```
make run
```

Use "Quit" button 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.

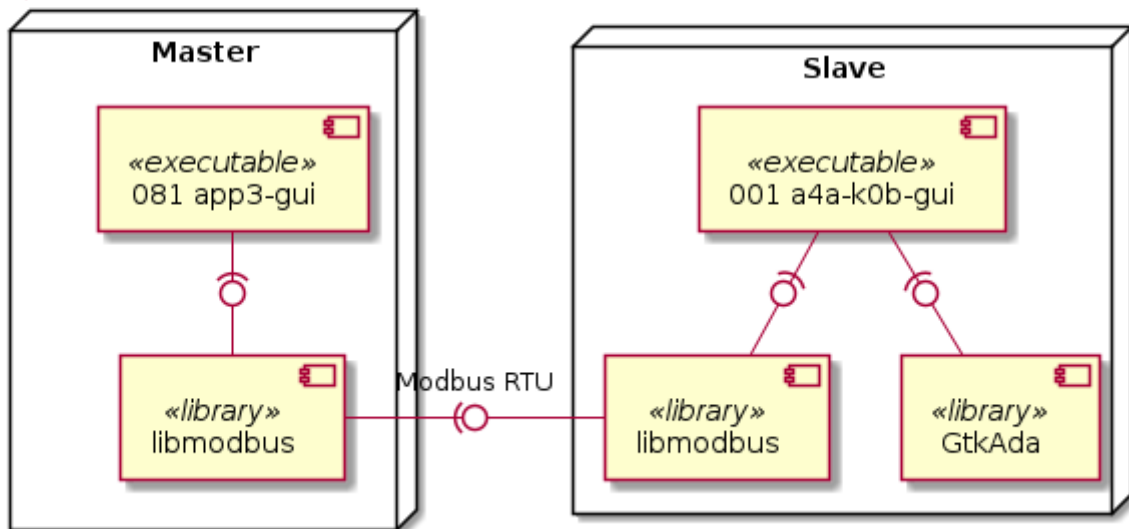
Chapter 7. Application

This is a basic **Ada for Automation** application which implements a Modbus RTU Slave.

It has a Command Line and Graphical User Interfaces and listen to Modbus RTU requests.

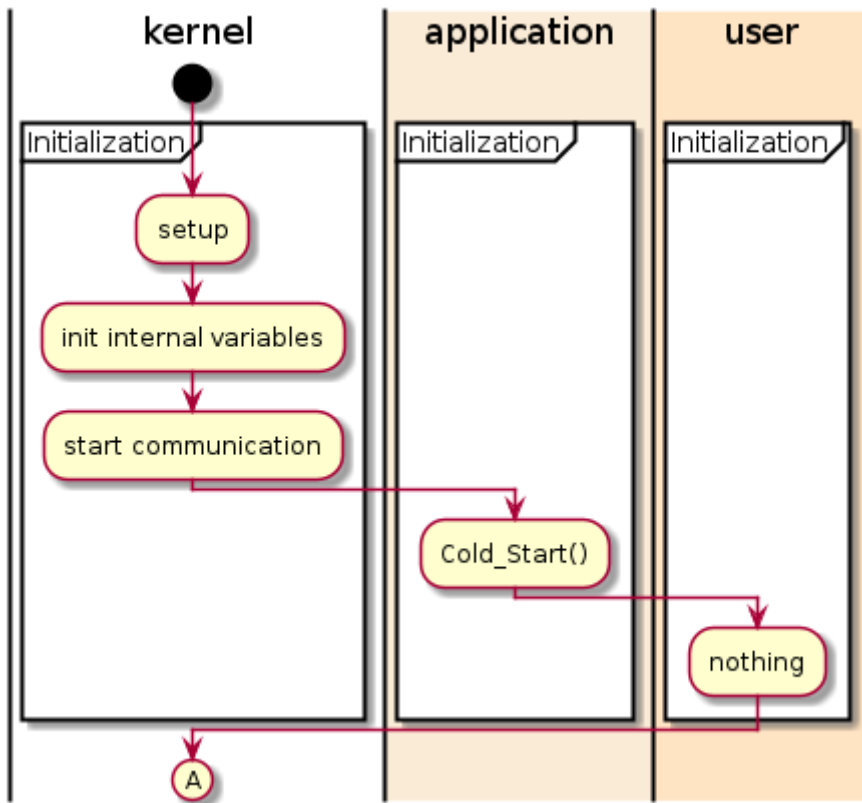
7.1. Deployment diagram

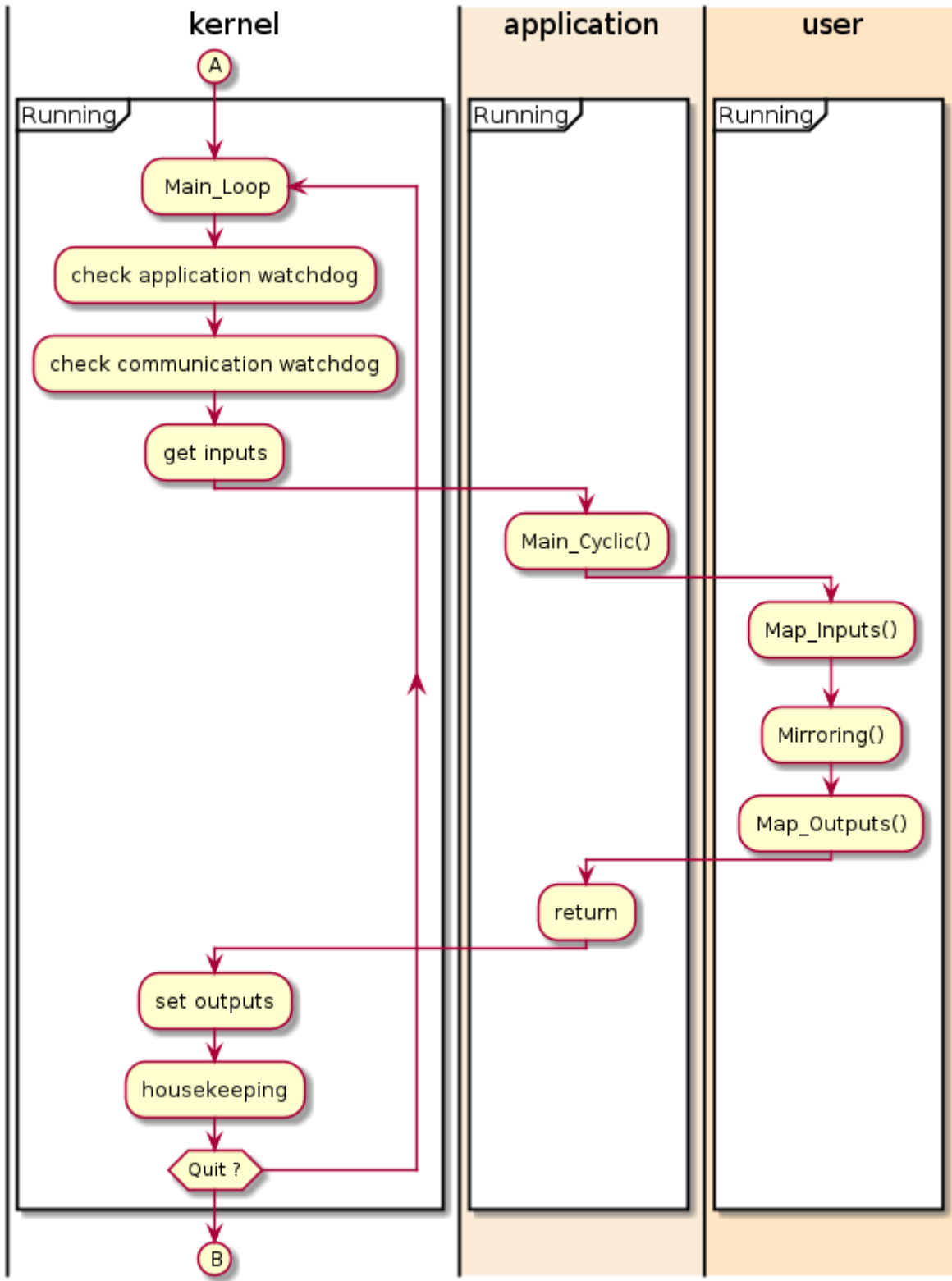
Could also be :
- 080 app3-cli
- 082 app3-wui
- your own
- a SCADA
- or a PLC

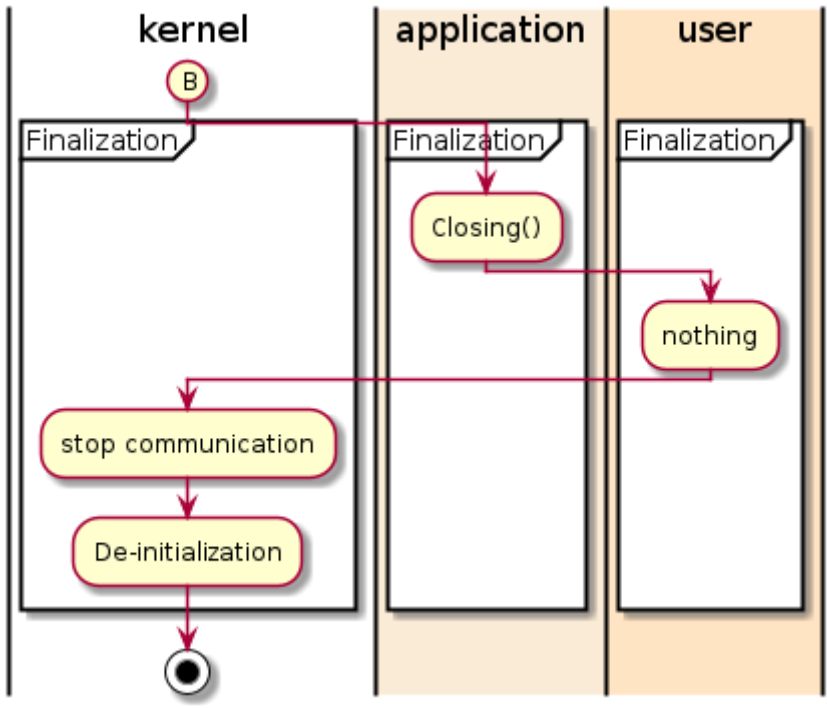


7.2. Activity diagram

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







7.3. Modbus RTU Slave Configuration

"../000 a4a-k0b-cli/src/a4a-application-mbrtu_slave_config.ads"

```
package A4A.Application.MBRTU_Slave_Config is

-----
-- Modbus RTU Slave configuration
-----

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

Config1 : aliased Slave.Slave_Configuration :=
  (Slave_Enabled          => True,
   Debug_On              => False,
   Retries                => 3,

   Device                 => To_Bounded_String ("/dev/ttyUSB1"), -- ①
   -- "COM1" on Windows
   -- "/dev/ttyS0" or "/dev/ttyUSB0" on Linux

   Slave                  => 2, -- ②

   Baud_Rate              => BR_115200,
   Parity                 => Even,
   Data_Bits              => 8,
   Stop_Bits              => 1);

end A4A.Application.MBRTU_Slave_Config;
```

① Modbus RTU Device : /dev/ttyUSB1

② Modbus RTU Slave Address : 2

7.4. User objects Definition

"../000 a4a-k0b-cli/src/a4a-user_objects.ads"

```
package A4A.User_Objects is
-----
-- User Objects creation
-----

-----
-- Inputs
-----

Input_Bits : Bool_Array (0 .. 15) := (others => False); -- ①

-----
-- Outputs
-----

Coils      : Bool_Array (0 .. 15) := (others => False); -- ②

end A4A.User_Objects;
```

- ① An array of 16 Input bits that a Modbus RTU Master can read is defined.
- ② As well, an array of 16 Coils can be written by the master.

7.5. User Functions

"../000 a4a-k0b-cli/src/a4a-user_functions.adb"

```
package body A4A.User_Functions is
-----
-- User functions
-----

procedure Map_Inputs is -- ①
begin
    Coils := MBRTU_IOSlave.Bool_Coils (Coils'First .. Coils'Last);

end Map_Inputs;

procedure Map_Outputs is -- ②
begin
    MBRTU_IOSlave.
        Bool_Inputs (Input_Bits'First .. Input_Bits'Last) := Input_Bits;

end Map_Outputs;

procedure Mirroring is -- ③
begin
    Input_Bits := Coils;

end Mirroring;

end A4A.User_Functions;
```

User functions are defined to :

- ① get the inputs from the slave,
- ② set slave outputs,
- ③ mirror the data.

7.6. User Application

"../000 a4a-k0b-cli/src/a4a-application.adb"

```
package body A4A.Application is

  procedure Cold_Start is
  begin

    null;

  end Cold_Start;

  procedure Closing is
  begin

    null;

  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

    -- Mirroring Inputs in Outputs
    Mirroring; -- ②

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

    Map_Outputs; -- ③

  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 slave,
- ② copies the Coils to the Input bits,
- ③ sets slave outputs.