El Robot Butiá

Segunda parte

Temas

- Módulos de usuario.
- Protocolos.
- Drivers.
- Plug and play.
- Butiá en Tortugarte.

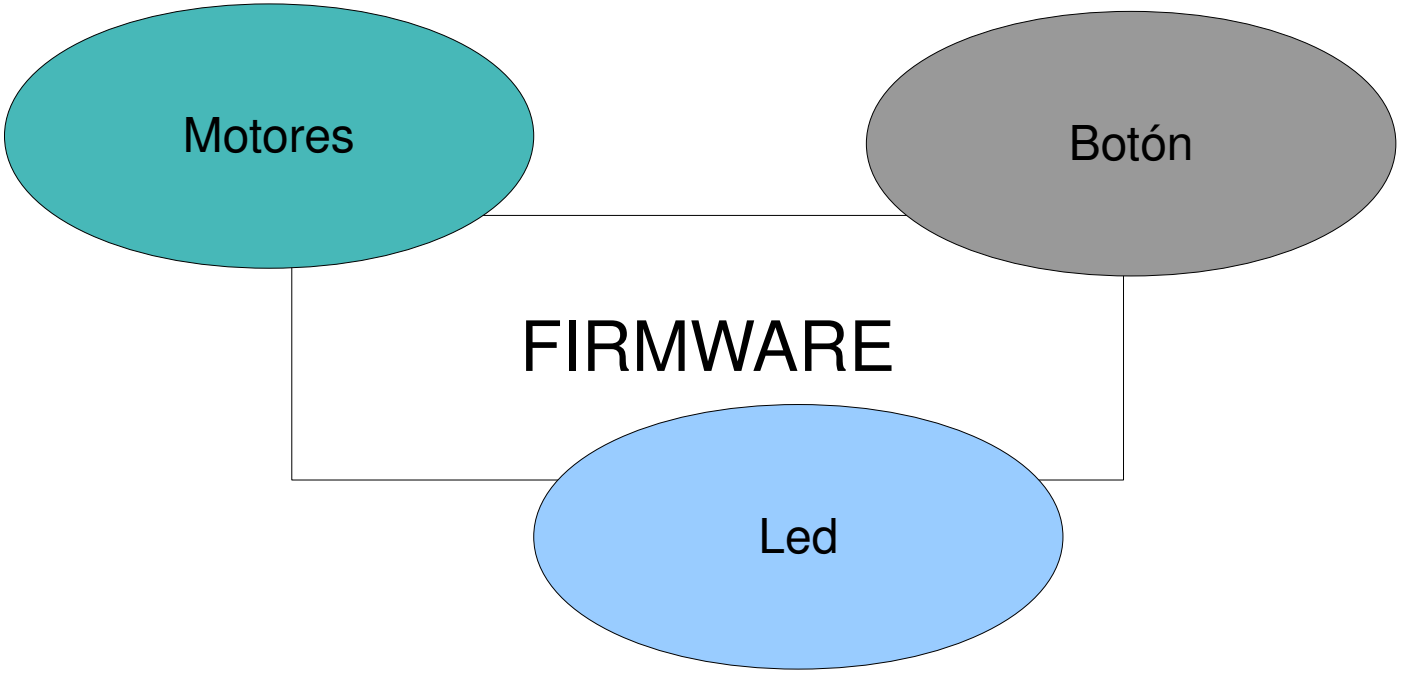
Módulos de usuario

- Componentes de software implementados en el firmware, encargados de encapsular la lógica específica para el manejo de determinado dispositivo o conjunto de dispositivos.
- El usuario solo debe concentrarse en la lógica propia del sensor/actuador a controlar, sin preocuparse de la comunicación con el PC/SBC (host).
- Lógica independiente del mecanismo de comunicación.
- Permite extender de forma muy simple las funcionalidades del firmware. No es la única forma de extenderlo.

Módulos de usuario (2)

- Fomenta un diseño modular de la solución.
- Fomenta el reuso de los módulos para resolver diferentes problemas.
- Se identifican por su nombre (7 carácteres).
- Exponen funciones al host, por ejemplo un motor: detenerse, avanzar, retroceder.
- Fácil desarrollo. No es necesario conocer cómo se gestiona el firmware para desarrollar un módulo de usuario.

Módulos de usuario (3)



Ejemplo: Módulo usuario “botón”

```
//módulo de usuario que maneja el sensor "botón"
byte boton (byte* buffer, byte largo, byte handler_id) {
   byte instruccion = buffer [0];                                          // obtengo el opcode
   switch (instruccion) {
    case 0: {                                                              // get_value
      boolean value = conector[handler[handler_id].num_conector].input(0); //leo de la entrada digital
      byte data [2] = {instruccion, !value};                               // armo el paquete de respuesta (opcode, data)
      sendMsg (handler_id, data, 2);                                       // respuesta
      }
      break;
  }
  return 0;

}
```

Ejemplo2: Módulo usuario “butiá”

```
//este módulo se encarga de implementar funcionalidades propias del robot
byte butia (byte* buffer, byte largo, byte handler_id) {                    // administrador del robot
  byte instruccion = buffer [0];
  switch (instruccion) {
    case 2:                                                 // get_version
      buffer [1] = VERSION;
      sendMsg (1, buffer, 2);
      break;
    case 3:                                                 // get_voltage
      motor[0].readInfo (PRESENT_VOLTAGE);
      buffer [1] = motor[0].status_data;
      sendMsg (handler_id, buffer, 2);
      break;
  }
  return 0;           // ACK
}
```

Comunicación

- Admin Protocol
- User Protocol
- Handler Protocol
- Butiá Protocol

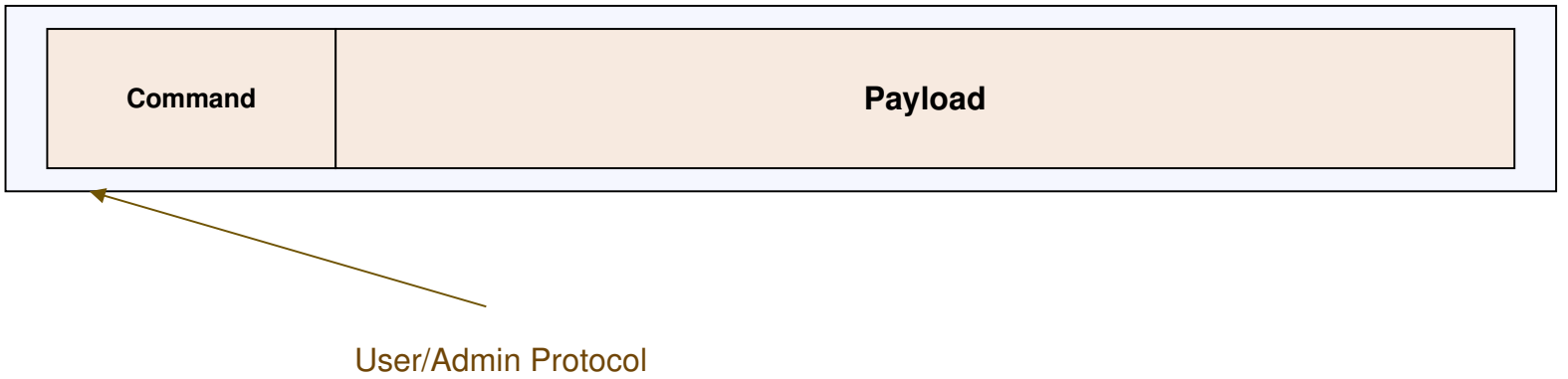
Host – Placa E/S

- Bobot-server Protocol

Cliente - Host

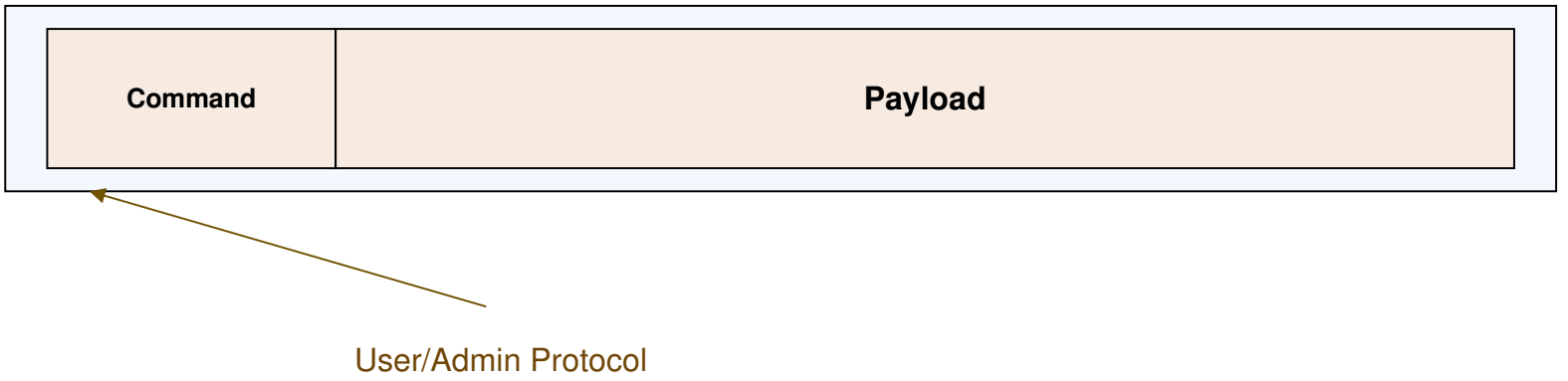
User Protocol

- Libre para que el usuario especifique las funcionalidades que el sensor/actuador expone.
- Command: Funcionalidad expuesta por el módulo
- Payload: Parámetros



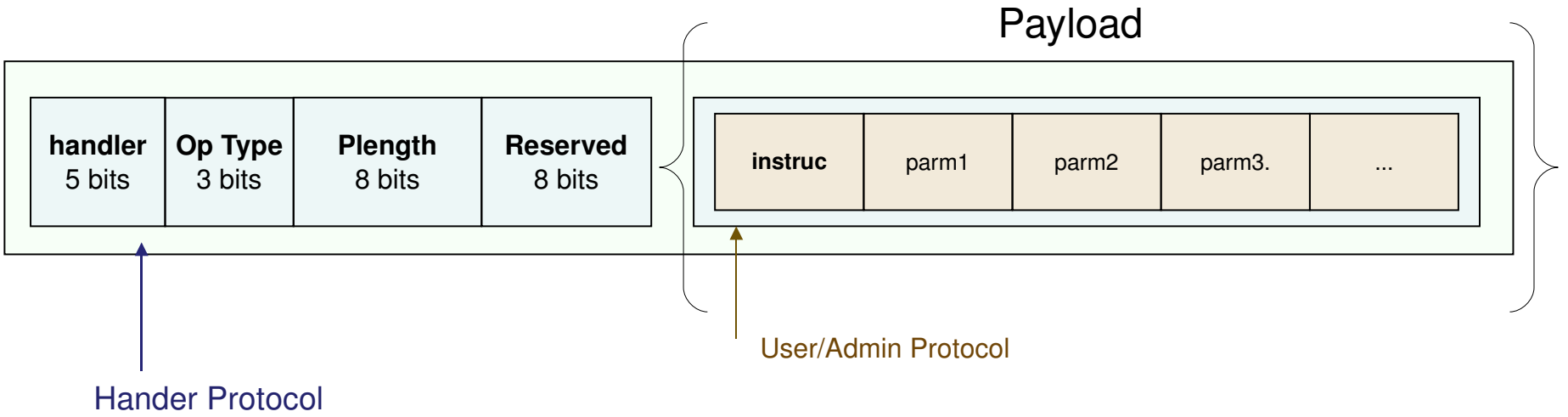
Admin Protocol

- Es utilizado para tareas de gestión de los módulos de usuario.
- Command: OPEN, CLOSE_ALL, GET_USER_MODULES_SIZE, GET_USER_MODULES_LINE.
- Payload: Argumentos para los comandos



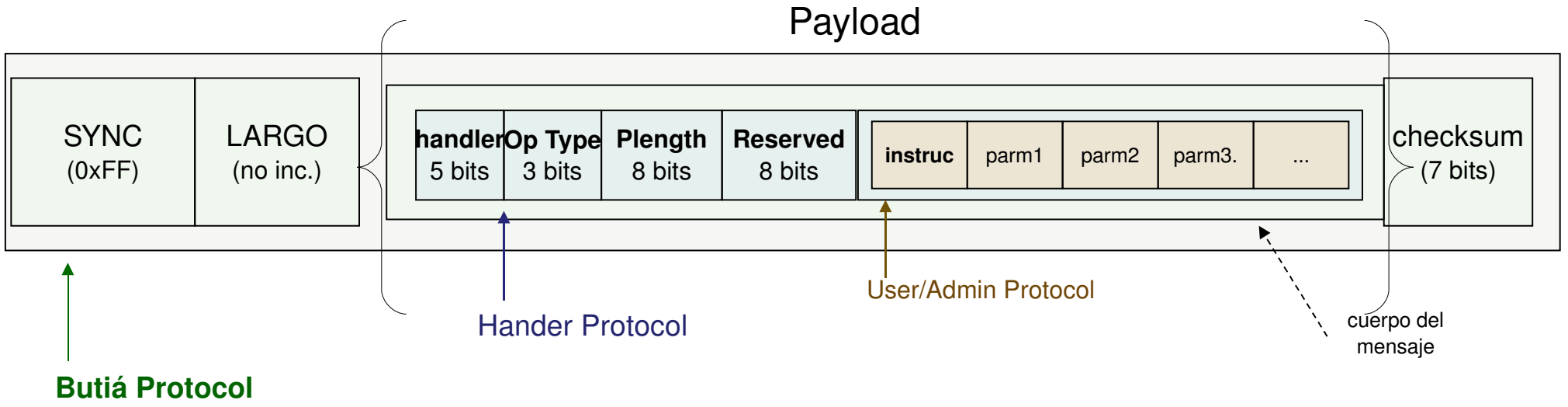
Handler Protocol

- Su objetivo es brindar mecanismos que permitan comunicar una aplicación con un módulo de usuario
- OpType: determina el tipo de operación que se desea ejecutar (send/configure)
- HandlerNumber: identifica el módulo de usuario destinatario del paquete
- Plength: especifica el largo del paquete enviado (incluye cabecera)
- Reserved: reservado para uso futuro.
- Payload: encapsula el paquete correspondiente al *admin protocol* o al *user protocol*

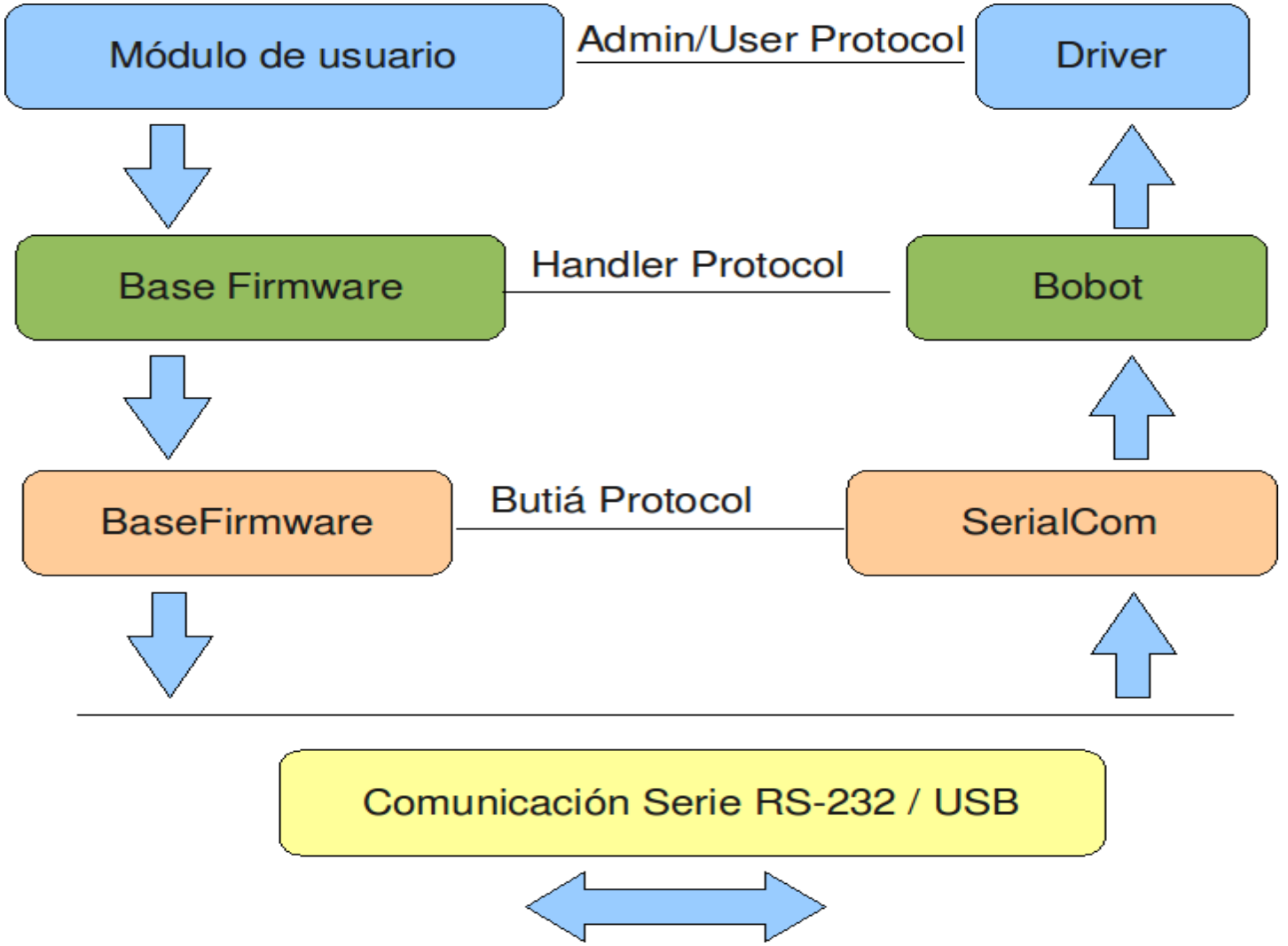


Butiá Protocol

- Ofrece un marco de confiabilidad a la comunicación. Se encarga de sincronizar la comunicación y verificar la consistencia de los datos.
- Sync: Se usa para saber donde se inicia un mensaje. Usamos el byte 0xFF. En caso de que exista el byte de Sync en el mensaje, se utilizara un carácter de escape como precedente (0xFE).
- Largo: Tamaño del mensaje, no inclusivo.
- Checksum: Es una suma de verificación. Suma de todos los bytes módulo 128.



Comunicación



Bobot-server Protocol

- Protocolo orientado a mensajes sobre TCP (Protocolo de Control de Transmisión, utilizado en Internet) por defecto en el puerto 2009.
- Para utilizarlo en las aplicaciones de usuario deben abrir una conexión TCP e implementar dicho protocolo.
- Se puede interactuar directamente desde un terminal Telnet usando TCP.

Bobot-server Protocol (2)

Los mensajes soportados son:

- LIST: Lista los módulos detectados.
- OPEN module [ep1 ep2]: Abre el módulo.
- DESCRIBE module [ep1 ep2]: Devuelve una descripción del módulo.
- CALL module function [parameters ...]: Invoca la función indicada en el módulo dado. Los parámetros dependen de la función.
- CLOSEALL: Cierra todos los módulos.

Bobot-server Protocol (3)

- Ejemplo Telnet on the fly

- Ejemplo Python on the fly

Drivers

- Componentes de software implementados en el host, encargados de encapsular la lógica específica para el manejo de determinado dispositivo o conjunto de dispositivos.
- Implementa el User Protocol.
- Utilizan primitivas brindadas por el bobot para independizarse del mecanismo de comunicación (USB/Serial rs232).
- Se identifican por su nombre y debe de existir un archivo .lua en el directorio drivers dentro del bobot.

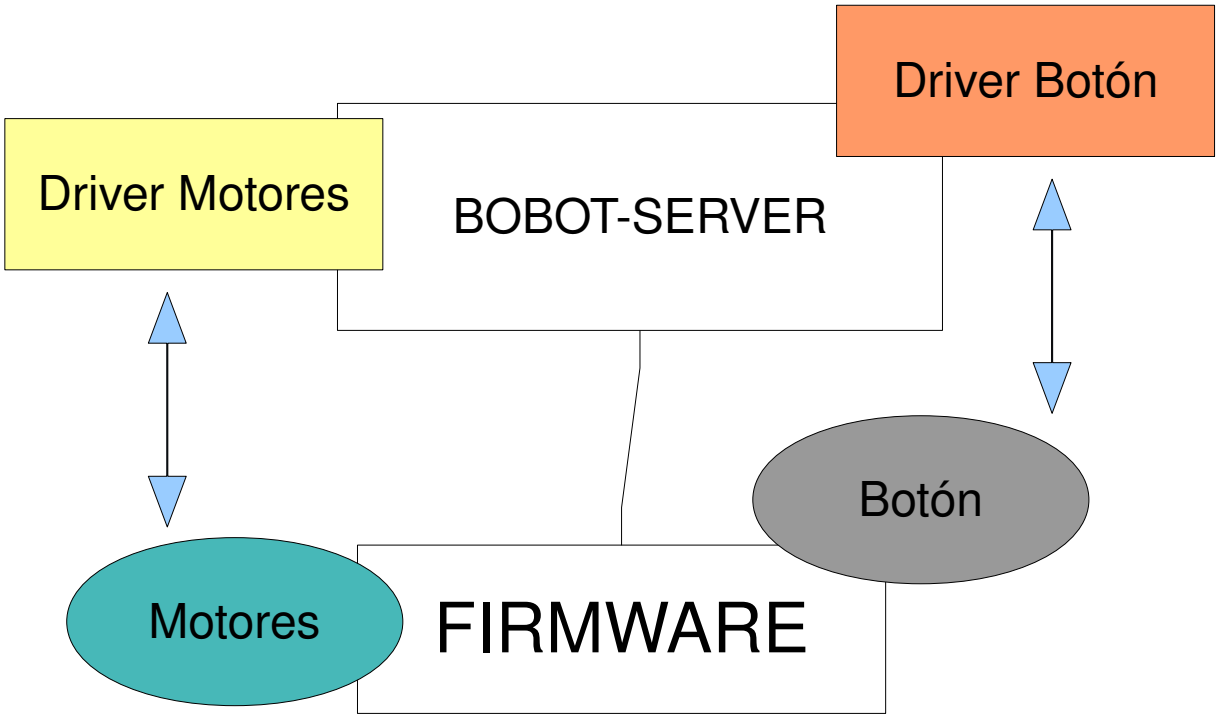
Drivers (2)

- Brindan información al host acerca de las funcionalidades que implementan.
  - Nombre de la funcionalidad.
  - Nombre y tipo de parámetros que recibe.
  - Nombre y tipo del valor de retorno.
- Esta información va a ser parseada por el bobot y brindada a los clientes.

  Ejemplo:

  - Pensar cuando agregamos una tarjeta de video en nuestra pc. ¿Qué sucede?

Drivers (3)


Driver Botón
Driver Motores
BOBOT-SERVER
Botón
Motores
FIRMWARE

Ejemplo: Driver “botón”

boton.lua

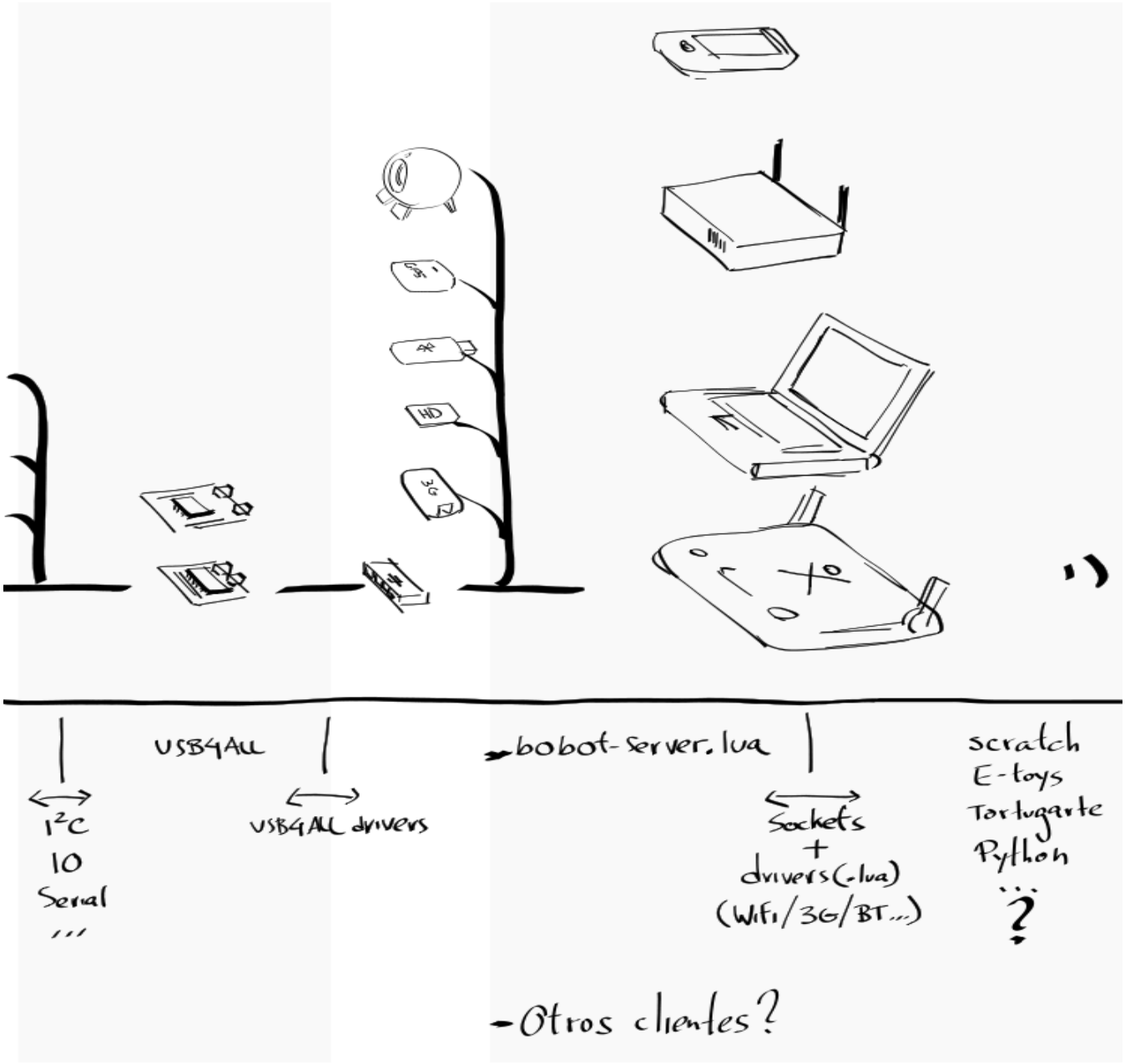
```lua
local device = _G

-- descripción: permite conocer el estado el botón en un momento dado.
-- entrada: no tiene.
-- salida: estado del botón. Posibles estados: 1 presionado, 0 libre.
api={}
api.getBoton = {}
api.getBoton.parameters = {} -- no tiene parámetros de entrada
api.getBoton.returns = {[1]={rname="par1", rtype="int"}} -- 1 = presionado, 0 = libre
api.getBoton.call = function ()
    device:send(string.char(0x00))          -- codigo de operacion = 0
    local sen_dig_response = device:read(2) -- leo 2 bytes (opcode, data)
    local raw_val
    if not sen_dig_response or string.byte(sen_dig_response or "00000000", 2) == nil
    then
        raw_val = "nil value"
    else
        raw_val = string.byte(sen_dig_response, 2) -- me quedo con los datos
    end
    return raw_val
end

```

Drivers (4)

- Ejercicio:
  - Hacer un cliente “Botón”:
    - Obtener el estado del botón que esta conectado al robot mediante el uso de las primitivas del servidor.


USB4ALL
bobot-server.lua
I²C
IO
Serial
...
USB4ALL drivers
Sockets
+
drivers(.lua)
(WiFi/3G/BT...)
scratch
E-toys
Tortugarte
Python
...
?
-Otros clientes?

Plug & Play

Qué es?

Plug & Play (2)

- Definición Plug & Play:

  Plug-and-play (conocida también por su abreviatura PnP) es la tecnología que permite a un dispositivo informático ser conectado a un ordenador sin tener que configurar (mediante jumpers o software específico proporcionado por el fabricante) ni proporcionar parámetros a sus controladores. Para que sea posible, el sistema operativo con el que funciona el ordenador debe tener soporte para dicho dispositivo.

  La frase plug-and-play se traduce como enchufar y usar. No obstante, esta tecnología en la mayoría de los casos se describe mejor por la frase apagar, enchufar, encender y listo.

Plug & Play (3)

Definición Hot Plug:

Conexión en caliente, traducido del inglés hot-plug, es la capacidad que tienen algunos periféricos de poder enchufarse o desenchufarse al ordenador, sin apagar el mismo, y funcionar correctamente.

Entre las conexiones con capacidad "hot-plug" se encuentran las conexiones USB, Firewire, SATA y SAS. Las conexiones en serie, en paralelo y PS/2 (ratón y teclado), podrían no estar adaptadas para conexión y desconexion con el ordenador encendido, ya que se podrían quemar los puertos o el periférico.

Plug & Play (4)

- Como los sensores/actuadores son en su mayoría dispositivos pasivos no hay posibilidad de que ellos puedan notificar su presencia ni su identidad al firmware.
- Se agregan pines de configuración digitales, independientes a las lineas que ya utiliza el sensor/actuador.
- Se organizan en conectores con polaridad, permitiendo al firmware detectar e identificar cada dispositivo.

Tablas de auto-configuración (cont.)

TIPO	SubTIPO	ID 0	ID 1	Digital 0	Digital 1	Analógico
0 - NADA / MANUAL / I2C	0 – NADA	1	1	INPUT	INPUT	INPUT
1 - Sensor Analógico	0 – (Distancia)	0	1	0	0	INPUT
1 - Sensor Analógico	1 – (Temperatura)	0	1	1	0	INPUT
1 - Sensor Analógico	2 – (Luz)	0	1	0	1	INPUT
1 - Sensor Analógico	3 – (etc.) (Acelerómetro?)	0	1	1	1	INPUT
2 - Sensor Analógico c/pin de control	0 – (Reflectividad)	1	0	OUTPUT	0	INPUT
2 - Sensor Analógico c/pin de control	1 – (etc.)	1	0	OUTPUT	1	INPUT

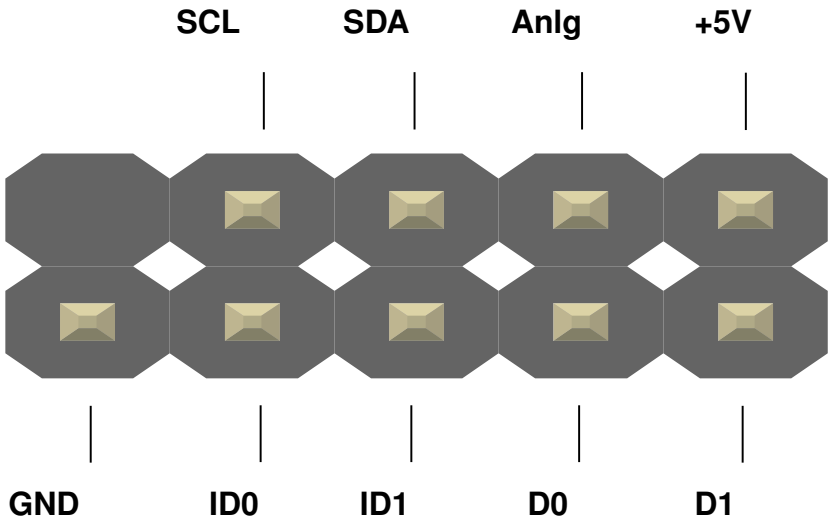
TIPO	SubTIPO	ID 0	ID 1	Digital 0	Digital 1	Resistencias	Analógico
3 - Sensor Digital	0 – (Switch)	0	0	INPUT	INPUT	GND	0
3 - Sensor Digital	1 – (Sensor de contacto)	0	0	INPUT	INPUT	68 / 10	131
3 - Sensor Digital	2 – (Inclinación)	0	0	INPUT	INPUT	68 / 15	185
3 - Sensor Digital	3 – (Encoder)	0	0	INPUT	INPUT	68 / 22	250
3 - Sensor Digital	4 – (etc.)	0	0	INPUT	INPUT	68 / 33	335
4 - Sensor Digital c/pin de control	0 – (Optoacoplador)	0	0	OUTPUT	INPUT	68 / 47	418
4 - Sensor Digital c/pin de control	1 – (etc.)	0	0	OUTPUT	INPUT	68 / 68	512
4 - Sensor Digital c/pin de control	2 – (etc.)	0	0	OUTPUT	INPUT	47 / 68	605
4 - Sensor Digital c/pin de control	3 – (etc.)	0	0	OUTPUT	INPUT	33 / 68	689
5 - Actuador Digital	0 – (Led)	0	0	OUTPUT	OUTPUT	22 / 68	774
5 - Actuador Digital	1 – (Relé)	0	0	OUTPUT	OUTPUT	15 / 68	839
5 - Actuador Digital	2 – (Servo)	0	0	OUTPUT	OUTPUT	10 / 68	893
5 - Actuador Digital	3 – (etc.)	0	0	OUTPUT	OUTPUT	5V	1023

Plug & Play (5)

- Tipos:
  - Quedan definidos por los bits ID0 e ID1.
  - Nada/I2C, s. analógico, s. digital, actuador digital, entre otros.
  - Divide en clases de sensores.
- Subtipos:
  - Digital: Switch, contacto, inclinación...
  - Analógicos: Distancia, luz, temperatura...
  - Son subclases dentro de los Tipos que separan comportamientos.

Plug & Play (6)

- Funcionamiento
  - Cuando el firmware inicia la exploración de los conectores.
  - Se cargan los módulos de usuario correspondientes a los dispositivos conectados.
  - Se configuran los pines como E/S dependiendo de la configuración del conector.



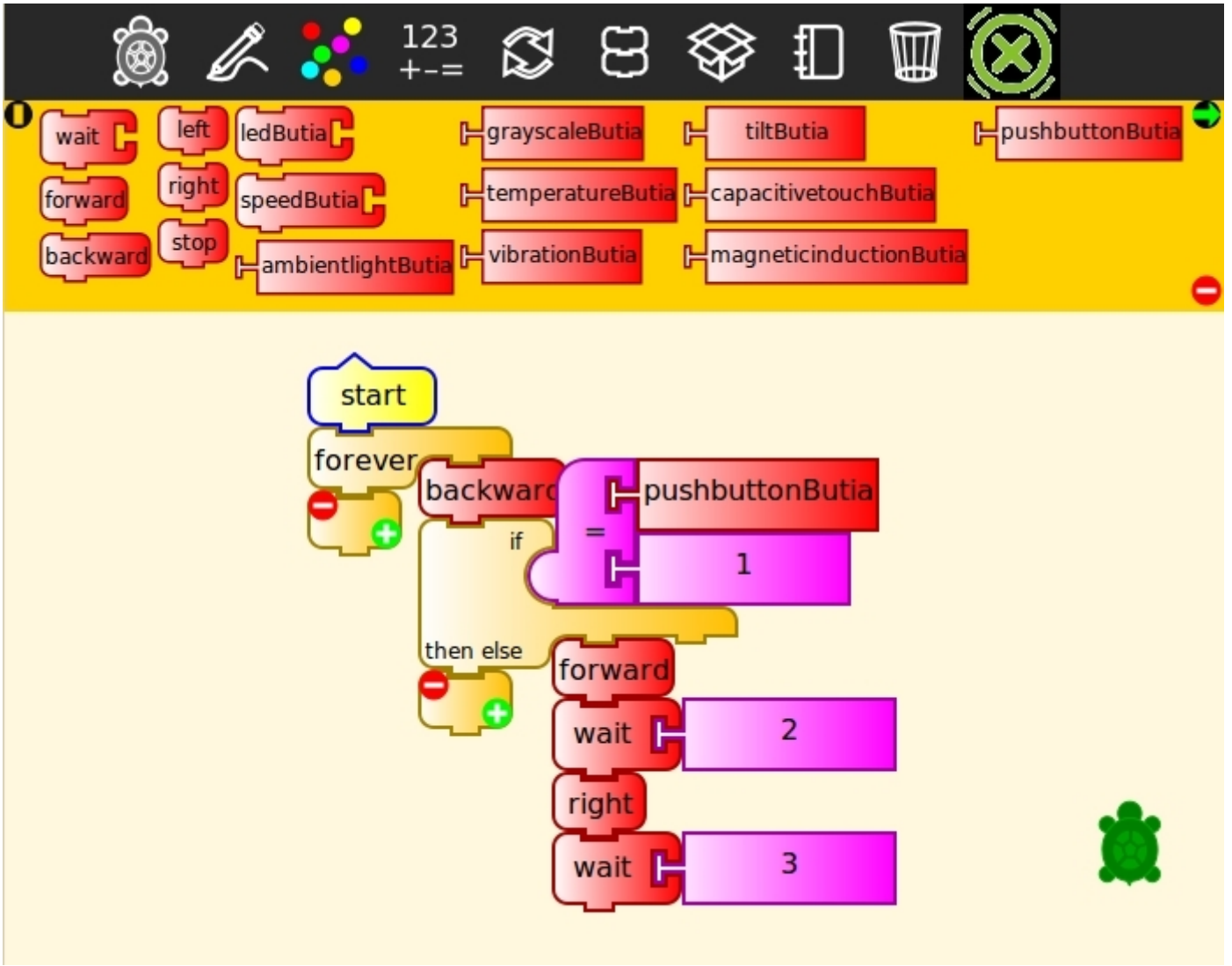
Plug & Play (7)

- Ventajas
  - Conecto los dispositivos en el puerto del shield más adecuado para resolver el problema que se plantea.
  - Facilita la configuración.
  - Brinda las primitivas para realizar descubrimiento de servicios.

Tortugarte

- Aplicación de la XO.
- Lenguaje de programación intuitivo orientado a bloques.
- Se comunica con la API del butiá.
- Facilita el uso del butia a nivel escolar.

Tortugarte (2)



Referencias

- http://www.fing.edu.uy/inco/grupos/mina/pGrado/pgusb/material.html
- http://www.lua.org/
- http://es.wikipedia.org/wiki/Conexi%C3%B3n_en_caliente
- http://es.wikipedia.org/wiki/Plug-and-play

Demo

- Las dos placas conectadas a la vez.

- ArduinoMega y USB4all.