# bobot-server v2: Documentación Técnica

Guillermo Apollonia (gapollo@fing.edu.uy)
Jorge Visca (jvisca@fing.edu.uy)

Marzo de 2011

# Capítulo 1

# Descripción

BOBOT-SERVER (VERSION 2) es un servicio que permite acceder a aplicaciones y usuarios interactuar con dispositivos USB4ALL. Consiste en agente altamente portable y liviano, que exporta la funcionalidad de los dispositivos USB4ALL presentes de una forma facil de usar. Ofrece dos métodos de acceso, uno optimizado para aplicaciones, basado en un socket y un protocolo facilmente parseable, y otro optimizado para ser usado por humanos, mediante un sitio web hosteado en el propio agente.

# Capítulo 2

# Diseño

BOBOT-SERVER accede a los dispositivos USB4ALL mediante la biblioteca BOBOT, y es un reemplazo drop-in al servidor BOBOT-SERVER. El servicio consiste en un servidor de sockets que acepta conexiones TCP en dos puertos (cada uno en un protocolo distinto). Es facilmente extensible a más protocolos. Cada vez que acepta una conexión, crea una sesión TCP que es mantenida hasta que el cliente la cierre.

Los dos protocolos soportados son BOBOT-PROCESS y BOBOT-HTTP.

## 2.1. bobot-process

Es un protocolo simple orientado a comunicaciones entre procesos. Por defecto se realiza sobre el puerto 2009, y es el único que soportaba la versión anterior del servicio, BOBOT-SERVER. Consiste en un intercambio de comandos, generados por el cliente, y respuestas a estos, generados por el servicio. Ambos consisten en strings, donde cada comando o respuesta ocupa una linea (están delimitados por el caracter "$\backslash n$"). Es sensible a mayúsculas. Si bien el protocolo soporta la transmisión de caracteres no imprimibles, para simplificar el debugging se recomienda que las clientes (y los módulos USB4ALL asociados) usen algun esquema de codificación que transmita solo texto plano.

Por la especificación del protocolo, consulte el manual de usuario. La implementación de los comandos del protocolo se encuentra en el módulo *bobot-server-process.lua*.

## 2.2. bobot-http

Es un protocolo orientado a que los usuarios puedan facilmente acceder a las funcionalidades de los dispositivos USB4ALL. Esto puede ser útil durante las etapas de prototipado, desarrollo, y debugging. Consiste en una sitio web disponible por defecto sobre el puerto 2010. Esta sitio permite navegar de forma

interactiva los servicios USB4ALL disponibles y la operaciones asociadas, así como ejecutar acciones y recuperar los resultados.

El sitio web es facilmente modificable mediante *templates*, y puede ser extendido con páginas de propósito específico. Las páginas son generadas por el módulo *bobot-server-http.lua*.

# Capítulo 3

# Implementación

## 3.1. Gestión de conexiones

BOBOT-SERVER está escrito en Lua. La única dependencia externa, además de la librería BOBOT, es LuaSocket. Por lo tanto, se puede ejecutar sin modificaciones en la mayoría de las plataformas con soporte para Sockets POSIX, incluyendo Linux, BSD y Windows.

El programa principal reside en el archivo *bobot-server.lua*. Contiene el bucle principal que escucha los sockets, y el despachador que da soporte a los protocolos.

La tabla *socket_handlers* contiene las funciones handler para cada conexión TCP abierta. Al iniciar el sistema, contiene únicamente los handlers para los socket que aceptan conexiones en los puertos asignados a los protocolos (vea Figura 3.1).

Para agregar soporte para un nuevo protocolo, hay que agregar el socket en el puerto correspondiente, y asignarle un handler en la tabla *socket_handlers*.

Cada socket_handler asignado a un socket-server debera abrir ua sesión TCP y obtener el socket asociado, y con este realizar al menos dos tareas (como ejemplo vea Figura 3.2).

1. Almacenar el socket en el array *recvt*
2. Asignarle un handler y registrarlo en *socket_handlers*.

El handler del protocolo deberá gestionar el cierre de la sesión TCP, eliminando el socket correspondiente del array *recvt* (el borrado del propio handler se realiza automáticamente).

## 3.2. Handler de bobot-process

El handler del protocolo bobot_process realiza los siguientes pasos cada vez que recibe un cambio de estado en la sesión TCP asociada:

```
local server_b = socket.bind("*", 2009) --bobot-process
local server_h = socket.bind("*", 2010) --bobot-http

local socket_handlers = {}
setmetatable(socket_handlers, { __mode = 'k' })

socket_handlers[server_b] = function()
        --handler para aceptar conexiones bobot-process
end
socket_handlers[server_h] = function()
        --handler para aceptar conexiones bobot-http
end
```

Figura 3.1: Socket handlers para aceptar conexiones

```
        local client, err=server_x:accept()
        if client then
                table.insert(recvt,client)
                socket_handlers[client] = function ()
                        --handler para procesar
                        --mensajes del protocolo
                end
        end
```

Figura 3.2: Socket handler de protocolo

```
process = {}

process["COMMAND1"] = function ( parameters )
        --interactuo con modulo bobot
         return ret
end
process["COMMAND2"] = function ( parameters )
        --interactuo con modulo bobot
end
```

Figura 3.3: Declaración de funciones del protocolo bobot-process

1. Gestiona si es un cierre de conexión.
2. Si recibe un mensaje, lo separa en palabras, donde la primer palabra es la instrucción (consultar manual de usuario)
3. Utiliza la tabla *process* para recuperar la función asociada al comando y la invoca, pasándole los parámateros (la lista de palabras)
4. El retorno de la invocación anterior es enviada de vuelta por el socket hacia el cliente.

La tabla process está declarada e inicializada en el módulo *bobot-server-process.lua* (ver Figura 3.3).

## 3.3. Handler de bobot_http

El handler del protocolo bobot_http realiza los siguientes pasos cada vez que recibe un cambio de estado en la sesión TCP asociada:

1. Delega al módulo *bobot-server-http* intentar realizar la lectura y procesado de una solicitud http.
2. Si obtiene un retorno, lo envía por el socket de vuelta hacia el cliente
3. Si obtiene un mensaje de error notificando del cierre de conexión remota, gestióna el cierre de conexión local

El procesado de las consultas http se realiza en el modulo *bobot-server-http*, con un punto de entrada en la función *serve()*. Este realiza el parseo del cabezal http (POST o GET), y extrae la URL solicitada y el string de parametros pasado (de existir). A continuación recupera la función asociada a la URL de la tabla *get_page*, y la invoca pasándole el string con parámetros. La tabla *get_page* está configurada de modo que si se requiere una URL que no aparece en la tabla, intenta localizar un archivo de igual nombre en el sistema de archivos (ver Figura 3.4).

```
local get_page={}

setmetatable(get_page, {
        __index = function(_,page)
                return find_page(page)
        end
})

get_page["/page1.htm"] = function (p)
        local params = parse_params(p)
        local page = "" --generar pagina
        return "HTTP/1.1 200/OK\r\nContent-Type:"
                .."text/html\r\nContent-Length: "
                ..#page.."\r\n\r\n"..page
end
get_page["/img.png"] = function (p)
        local params = parse_params(p)
        local page = "" --generar imagen
        return "HTTP/1.1 200/OK\r\nContent-Type:"
                .."image/png\r\nContent-Length: "
                ..#page.."\r\n\r\n"..page
end
```

Figura 3.4: Declaración de paginas a servir

El sistema es facilmente extensible. Para agregar páginas estáticas, basta con copiarlas en el sistema e archivos. Para agregar una página dinámica, hay que agregar la entrada correspondiente en la tabla *get_page*. La mayoría de las páginas dinámicas provistas se basan en un sistema de templates, y estos pueden ser modificados para alterar la apariencia del sitio.