Programmeerproject 2013-2014

XBEE interface

Devices simulation

Titularis: Prof. Theo D'Hondt
Begeleidende Assistenten:
Kevin Van Vaerenbergh (kevvaere@vub.ac.be)
Lode Hoste (lhoste@vub.ac.be)
Yves Vandriessche (yvdriess@vub.ac.be)
project2ba@dinf.vub.ac.be

Context

Voor deel 2 van het tweede bachelor programmeerproject 2013-2014 moeten jullie zorgen voor communicatie tussen de steward en de devices. Hiervoor krijgen jullie van ons een XBEE interface die samen met een C-library zal zorgen voor de communicatie tussen DrRacket/Slip en XBEE-devices.
Om deze communicatie te verzekeren gaan jullie best in 2 fase werken. Bij de eerste fase gaan jullie de devices simuleren zodat de communicatie tussen XBEE en device kunnen aantonen zonder in de buurt te zijn van devices. Om dit te verduidelijken is er meer uitleg te vinden in het laatste paragraaf. Eerst gaan we beginnen met de XBEE interface uitleggen.

XBEE-interface

In dit paragraaf gaan we uitleg geven over hoe de XBEE-communicatie werkt gebruik makende van onze Foreign Function Interface (FFI).
We geven jullie 7 functies ter beschikking die voor de basis functionaliteiten van XBEE-communicatie staan.

```
(provide
 xbee-initialise
 xbee-tick
 xbee-ready?
 xbee-read-frame
 xbee-discover-nodes
 xbee-list-nodes
 xbee-write)
```

*

De hierboven vermelde functies zijn beschikbaar voor jullie om te gebruiken. Een voorbeeld van hoe deze functies te gebruiken word hieronder vermeld:

We beginnen met je lokale Xbee radio te bepalen door te refereren naar de file-descriptor. Deze file-descriptor is een file die automatisch door de OS wordt aangemaakt wanneer het Xbee toestel op je machine wordt aangesloten. vb.: #"/dev/tty.usbserial-13WRQNPX" op Mac OSX, #"/dev/ttyUSB0" op LINUX/Raspberry Pi. Let op, Racket verijst dat er hier gebruik wordt gemaakt door een "byte-string" (#"" ipv de gewoonlijke ""), in slip kan je een gewone Scheme string gebruiken. Hiermee zullen we de Xbee initialiseren, het tweede argument aan de initialisatie is de standard baudrate van 9600.

```
> (define xbee (xbee-initialise XBEE 9600))
```

Elke Xbee device in een netwerk heeft een uniek 64-bit adres, wat in Scheme voorgesteld wordt met een vector van bytes. Dit kan je manueel ingeven gebaseerd op de handleiding, zoals bijvoorbeeld:

```
> (define PLUG #(0 19 162 0 64 155 139 45))
> (define SENSOR #(0 19 162 0 64 148 36 184))
```

Slip heeft hiervoor een ingebouwd bytevector datatype: #u8(0 19 162 0 64 155 139 45) Om de adressen van devices automatisch bepalen hebben we hulpfunctie geïntroduceerd om de devices in de buurt te vinden. Hiervoor gebruiken we de discover functie:

```
> (xbee-discover-nodes xbee)
```

De bovenstaande functie stuurt een speciaal bericht rond waar elke device in de buurt op zal antwoorden. Omdat dit asynchroon is, moet je zelf met een aparte functie de lijst opvragen van alle devices die tot nu toe geantwoord hebben. Deze onderstaande functie geeft een associatieve lijst terug met de ID's van de devices (een string), samen met hun 64-bit adres (een bytevector).

```
> (xbee-list-nodes)
'(("ZBS110V2120895" #(0 19 162 0 64 155 139 45)))
```

Nu we de meeste info hebben kunnen we de devices toespreken door de "xbee-write" functie. Deze functie verwacht 3 input parameters:

1. het Xbee radio object : resultaat van een xbee-initialise
2. het extern device adres : een vector van bytes
3. het door te sturen bericht : een vector van bytes

Het bericht die we in dit geval willen doorsturen is de string "PUT POW=ON\n" wat moet doorgestuurd worden als een vector van ruwe bytes:

```
> (list->vector (map char->integer (string->list "PUT POW=ON\n")))
#(80 85 84 32 80 79 87 61 79 78 10)
```

*

Na elke write naar een device verwachten we een ACK of NACK antwoord om te kunnen nagaan dat het bericht goed is toegekomen. Hiervoor wachten we even (“sleep 5”), doen we een “xbee-tick” om de buffer van berichten uit te lezen en printen we hierna de gekregen frames. Hieronder is een voorbeeld te vinden van hoe je een stopcontact “aan” zet. De hulpfunctie “print-frames” is een zelf geschreven functie de frame inleest dat op de buffer staan en die print. Het inlezen van een frame gebeurt via de “xbee-read-frame” functie.

```
> (xbee-write xbee PLUG #(80 85 84 32 80 79 87 61 79 78 10))
> (sleep 5)
> (xbee-tick xbee)
> (print-frames (xbee-read-frame xbee))
```

Hieronder vind je een voorbeeld hoe we bulk informatie over devices verkrijgen.

```
> (list->vector (map char->integer (string->list "GET\n")))
#(71 69 84 10)
> (xbee-write xbee PLUG #(71 69 84 10))
> (xbee-write xbee SENSOR #(71 69 84 10))
> (sleep 5)
> (xbee-tick xbee)
> (print-frames (xbee-read-frame xbee))
> (print-frames (xbee-read-frame xbee))
```

*

Device simulation

In dit paragraaf geven we wat meer uitleg over hoe jullie een device simuleren. De communicatie met de devices gebeurt via API frames die elk een specifiek type hebben dat hun gedrag identificeert. Hieronder vinden jullie een tabel dat 3 types weergeeft dat jullie gaan moeten implementeren als simulatie.

Frame Type (Hex)	Frame Name
0x10	Zigbee Transmit Request
0x8B	Zigbee Trasmit Status
0x90	Zigbee Receive Packet

We zullen beginnen met een voorbeeld om alles te verduidelijken.
Als we dit sturen:

```
> (xbee-write xbee PLUG #(71 69 84 10))
```

Moeten we een data frame terugkrijgen met daarin het juiste type, het 64bit adres, het 16bit adres en de inhoud. Hier een voorbeeld,

```
#(144 0 19 162 0 64 155 138 139 247 116 1 80 79 87 61 79 78 10 70 82 69
81 61 52 57 46 56 49 50 53 72 122 10 86 82 77 83 61 50 50 55 86 10 73 82
77 83 61 49 57 56 56 109 65 10 76 79 65 68 61 52 52 51 87 10 87 79 82 75
61 48 46 48 52 54 107 87 104 10 10)
```

Als we deze frame even ontleden zien we het volgende:

Bytes	Hex / String	
144	90	Frame type
0 19 162 0 64 155 138 139	00 13 a2 00 40 9b 8a 8b	64-bit adres
247 116 1	F7 74 01	16-bit adres
80 79 87 61 79 78 10 70 82 69 81 61 52 57 46 56 49 50 53 72 122 10 86 82 77 83 61 50 50 55 86 10 73 82 77 83 61 49 57 56 56 109 65 10 76 79 65 68 61 52 52 51 87 10 87 79 82 75 61 48 46 48 52 54 107 87 104	P O W = O N F R E Q = 4 9 . 8 1 2 5 H z V R M S = 2 2 7 V I R M S = 1 9 8 8 m A L O A D = 4 4 3 W W O R K = 0 . 0 4 6 k W h	Data packet : een ASCII string in dit geval
10 10	A A	Twee line feeds om frame mee af te sluiten

Na het versturen van een bericht naar de devices, zal het device ons een ACK/ NACK sturen om te bevestigen of het bericht al dan niet is doorgevoerd. Hier zien

*

we een voorbeeld. Dit datapacket is hetzelfde als hierboven in detail uitgelegd met als verschil dat de data er zo uitziet als string:

```
ack: set pow=on
```

Elke device dat jullie moeten simuleren heeft verschillende operaties dat hij kan uitvoeren. In de test kamer gaan verschillende devices staan waaronder deze twee:

1. Smart Energy Meter / Smart Plug (ZBS-110V2): http://pointcarre.vub.ac.be/index.php?application=weblcms&course=2905&tool=document&go=course_viewer&publication_category=23674&browser=table&tool_action=viewer&publication=222730

2. ZigBee Multi Sensor (ZBS-121): http://pointcarre.vub.ac.be/index.php?application=weblcms&course=2905&tool=document&go=course_viewer&publication_category=23674&browser=table&tool_action=viewer&publication=222712

Bij deze twee devices moeten jullie zeker enkele basis operaties kunnen simuleren (oa. “SET POW=ON”, “GET”, “SET TXT=1800”, etc…)

APPENDIX:
Om de communicatie tussen xbee en end-device te testen kunnen jullie de xbee aansluiten op jullie laptop/pc. De library dat wij gemaakt hebben voor jullie kan gebruikt worden op UNIX, MacOSX en Rasp. Pi. Studenten die gebruik maken van UNIX moeten het volgende commando invoeren op command-line (terminal) om de vereiste permissies toe te voegen zodat jullie het usb apparaat (xbee) kunnen aanspreken:

```
sudo gpasswd -a mijnusername dialout
```

Eens dit uitgevoerd moeten jullie uitloggen en opnieuw inloggen.

*