

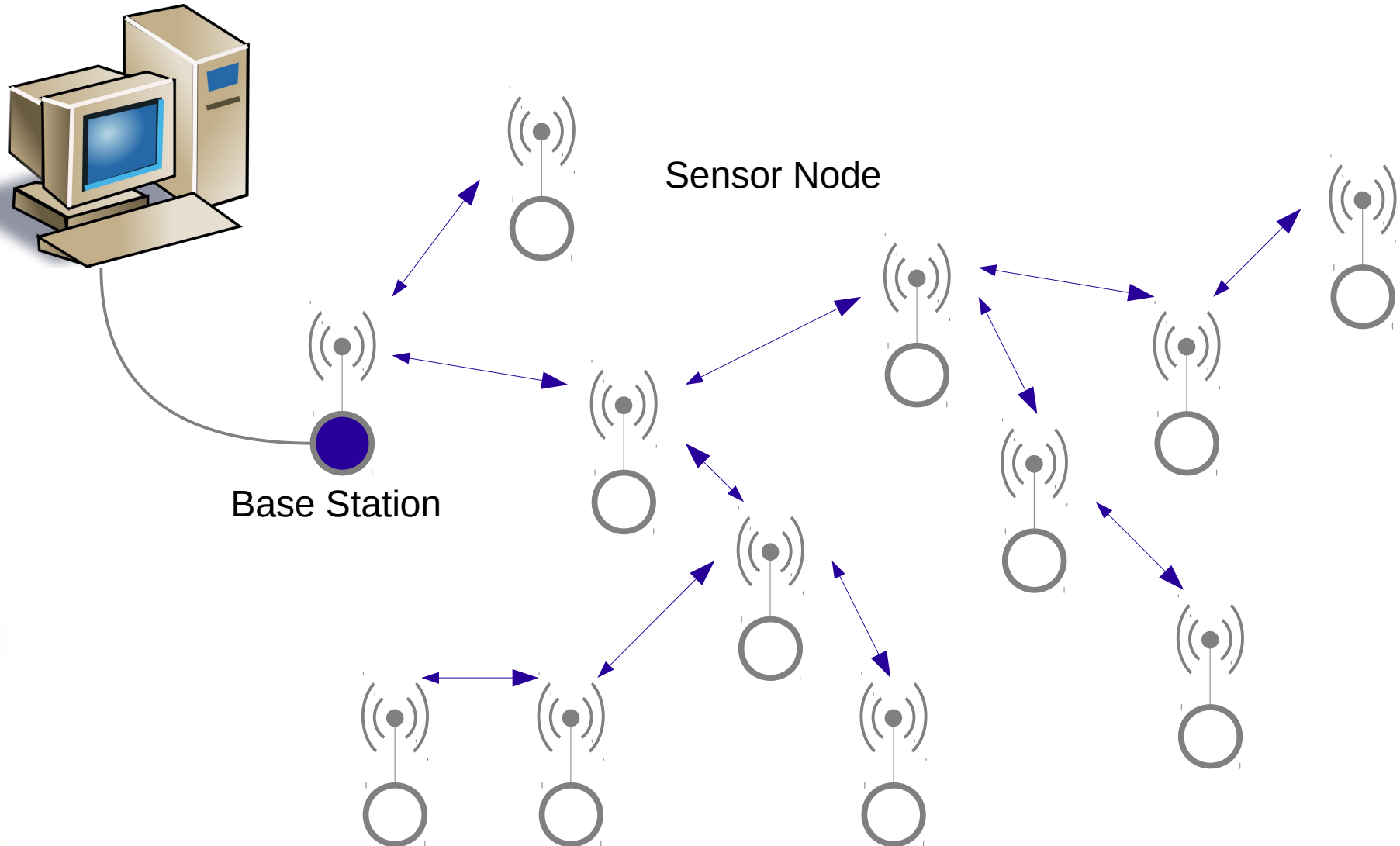
Terra System
Low abstraction level built-in functionalities
(TerraNet v0.2)

Introduction & user guide

Adriano Branco
abranco@inf.puc-rio.br

October, 2014

Wireless Sensor Network



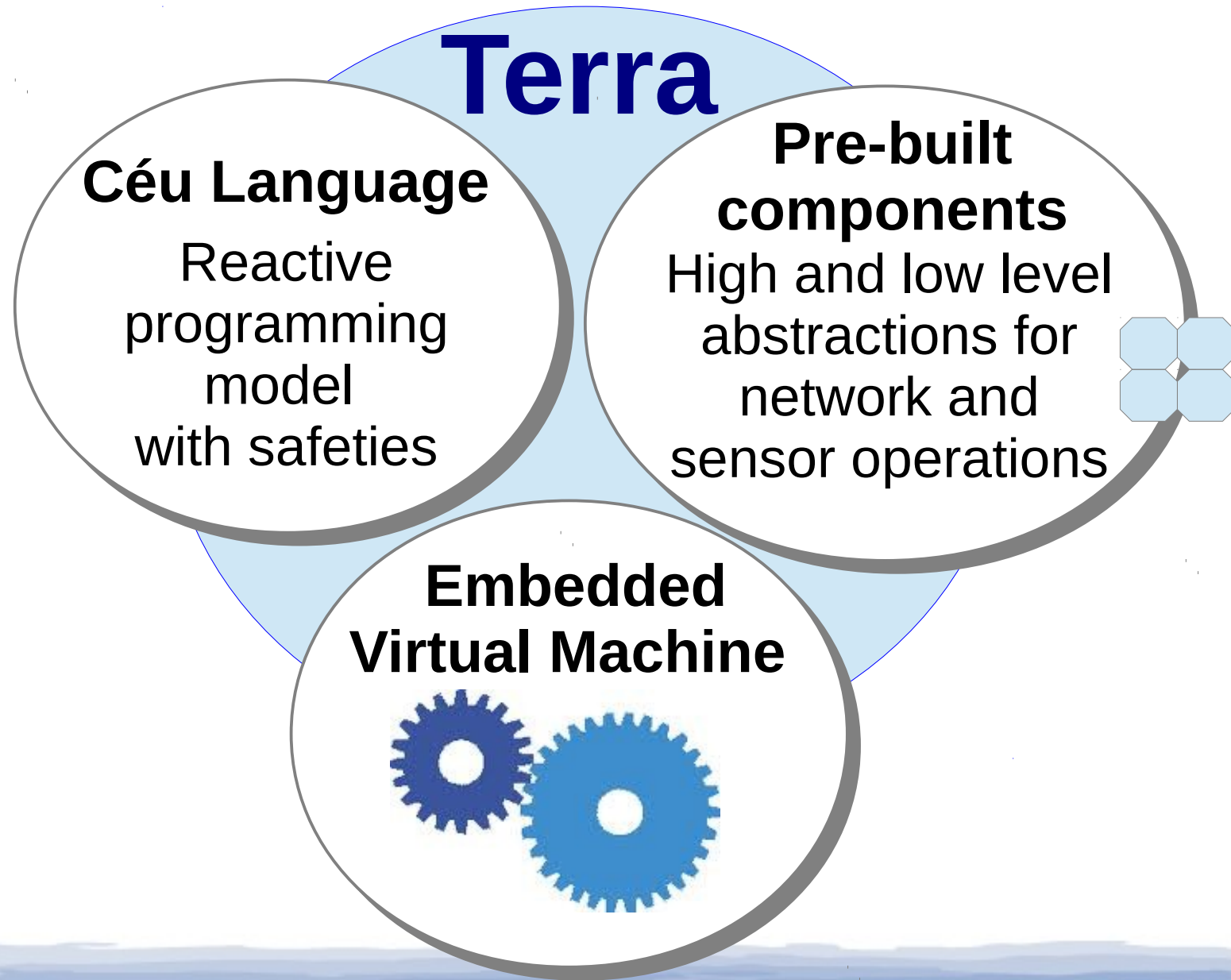
Main Challenges

- Resource scarcity
 - Battery lifetime - radio is the main battery consumer
 - Micro-controller – RAM size (4K~10K)
- Communication
 - Ad-hoc network, node volatility, noise, radio collision, etc
- Programming
 - Event driven model and distributed system
 - Remote programming

Terra System Motivation

- WSN – Wireless Sensor Network
 - Small devices: μ Controller + Radio + Sensors + Battery
- Programming challenges:
 - Event-oriented
 - Distributed application – intra-coordination
 - Resource scarcity
 - Remote programming & configuration

Proposed System



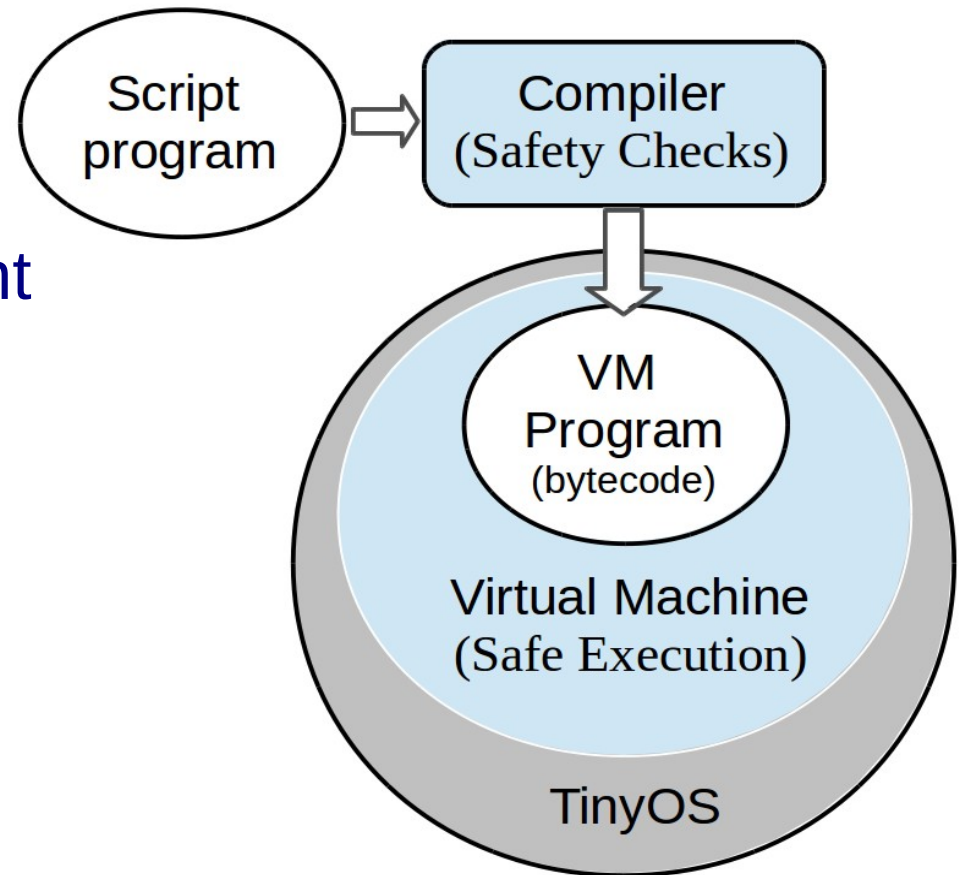
Sample Code

The diagram illustrates the flow of the sample code. It features a large box labeled "Infinite loop" on the left. Inside this box, there are two main sections. The first section, labeled "Waits for sensor reads", contains a sub-block labeled "photo" (lines 4-5) and another sub-block labeled "temp." (lines 7-8). The second section, labeled "Alarm", contains lines 10-14. The code itself is as follows:

```
1:  var ushort tValue,pValue;
2:  loop do                                // Main Loop
3:      par/and do                          // Starts two parallel blocks
4:          emit REQ_PHOTO();              // Requests PHOTO value
5:          pValue=await PHOTO;             // Waits for "sensor done"
6:      with
7:          emit REQ_TEMP();                // Requests TEMP value
8:          tValue = await TEMP;            // Waits for "sensor done"
9:      end
10:     if pValue > 200 or tValue > 300 then
11:         emit LED0(ON);
12:     end
13:     await 1min;
14:     emit LED0(OFF);
15: end
```

Terra/Céu Main characteristics

- The synchronous execution model enables race-free concurrency.
- The compiler verifies if event reactions are deterministic.
- Applications execute within bounded memory and CPU time.
- VM embedded components escape from Céu static analysis.



Céu

- Reactive
 - Execution is split in trails (lines of code)
 - A trail react to an event (timer, external, or internal)
 - Run to completion each trail (never overlap trails execution)
- Safety guarantees
 - All loops must contain an await statement
 - Avoid trails triggered from same event to share same variable.

Race-free example

```
loop do
  par/and do
    await A;
    y = 1;
  with
    await A;
    y = 2;
  end
  emit LEDS(y);
end
```

```
loop do
  par/and do
    emit REQ_SENSOR1()
    await EV1
    y = 1;
  with
    emit REQ_SENSOR2()
    await EV2
    y = 2;
  end
  emit LEDS(y);
end
```

Terra script language in one page

(Based on Céu language)

Statements:

```
var <type> name;  
event <type> name;  
await (event | time);  
emit event;  
If <cond> then <blk> [else <blk>] end  
loop do <blk> [break] <blk> end  
(par | par/and | par/or) do <blk> [with <blk>]* end
```

Var types:

```
byte, ubyte    (8bits)  
short, ushort  (16bits)  
long, ulong    (32bits)
```

Operators:

```
infix: or, and, |, ^, &  
!= , ==, <=, >=,  
<, >, <<, >>,  
+, -, *, /, %;  
prefix: not, &, -, +, ~,  
*;
```

Terra/Céu examples

Try to continue

```
var ushort a=0;
loop do
  await 1min;
  a = // do something
  If a == 0 then
    break;
  end
end
// do continue
```

Periodic action

```
loop do
  await 1min;
  // do something
end
```

Repeat[do-wait]-while

```
event ushort a;
par/or do
  loop do
    par/and do
      // do something
      await a;
      // do other-thing 1
    with
      await 1min;
    end
  end
end
with
  await 4h;
end
```

do-wait-continue

```
par/and do
  // do something
with
  await 1min;
end
// do continue
```

Time-out

```
event ushort a;
par/or do
  // do something
  await a;
  // do other-thing 1
with
  await 1min;
end
// do continue
```

Obs: We use only timers and internal events to explain the language basics.

Terra extension

Definitions

- Terra uses “Céu external events” or “functions” to access the custom components.
- TerraVM must implements all predefined external events and functions.
- The program calls a external event using “**emit** <event>(param);” Céu command.
- The program receives a external event using “[var=]**await** <event>(var);” Céu command.

TerraNet

- Implement a thin Terra version using only basic components like radio and sensors.
- The user application must implement its own communication protocol.
- Main functionalities:
 - Radio communication uses only the radio primitives SEND and RECEIVE at the radio range.
 - Support for message queue.
 - Support for radio message acknowledge.
 - Sensors read, Leds set, and a custom digital I/O.

TerraNet Functionalities

- TerraNet components use only low abstraction level
 - Radio
 - Basic send/receive - 1-hop radio range
 - Send broadcast
 - Send to specific target with option to have acknowledge
 - User defined message structure up to 20 bytes
 - Small local message queue
 - Local sensor/actuator
 - Leds
 - Temperature, Luminosity, and battery voltage sensors
 - Digital output
 - Digital input (read and interruption)

Implemented Emits and Awaits ^(1/2)

Group	emit	await
Radio	SEND(usr_msg_t)	ubyte SEND_DONE() ubyte SEND_DONE(type)
	SEND_ACK(usr_msg_t)	ubyte SEND_DONE_ACK ubyte SEND_DONE_ACK(type)
		usr_msg_t RECEIVE() usr_msg_t RECEIVE(type)
Sensor	REQ_TEMP()	ushort TEMP
	REQ_PHOTO()	ushort PHOTO
	REQ_VOLTS()	ushort VOLTS
LEDS	LED1(u8)	
	LED2(u8)	
	LED3(u8)	
	LEDS(u8)	
Internal Error		ubyte ERROR() ubyte ERROR(err_id)
Message Queue		Ubyte Q_READY()

Implemented Emits and Awaits (2/2)

Group	emit	await
Digital I/O	CFG_PORT_A(u8)	
	CFG_PORT_B(u8)	
	SET_PORT_A(u8)	
	SET_PORT_B(u8)	
	REQ_PORT_A()	u8 PORT_A
	REQ_PORT_B()	u8 PORT_B
Digital HW Interrupt	CFG_INT_A(u8)	INT_A
	CFG_INT_B(u8)	INT_B
Loop-back event	REQ_CUSTOM(u8)	u8 CUSTOM_A

Implemented Functions

Group	Function	Description
Basic	ushort getNodeId()	Return NodeID
	ushort random()	Return 16bit Random
Message Queue	ubyte qPut(msg)	Put msg into queue
	ubyte qGet(msg)	Get msg from queue
	ubyte qSize()	Return Queue Size
	ubyte qClear()	Clear all queue entries

Basic use - Radio

```
#include "TerraNet.defs"
var ushort nodeId = getNodeId();
pkttype usrMsg from radioMsg with
    var ubyte[4] d8;
    var ushort[4] d16;
    var ulong[2] d32;
end
var usrMsg msgRadio;
msgRadio.d8[0] = 0;
if nodeId == 1 then
    msgRadio.source = nodeId;
    msgRadio.target = BROADCAST;
    loop do
        await 10s;
        inc msgRadio.d8[0];
        emit SEND(msgRadio);
        await SEND_DONE;
    end
else
    loop do
        msgRadio = await RECEIVE;
        emit LEDS(msgRadio.d8[0]);
    end
end
end
```

Include specific TerraNet configuration

Define new usrMsg type from radioMsg packet

Create a msgRadio variable of type usrMsg

Broadcast a
radio message

Waits for a
radio message

RadioMsg packet:

```
var ubyte type;
var ushort source;
var ushort target;
var payload[20] data;
```

usrMsg type:

```
var ubyte type;
var ushort source;
var ushort target;
var ubyte[4] d8;
var ushort[4] d16;
var ulong[2] d32;
```

Basic use - Queue

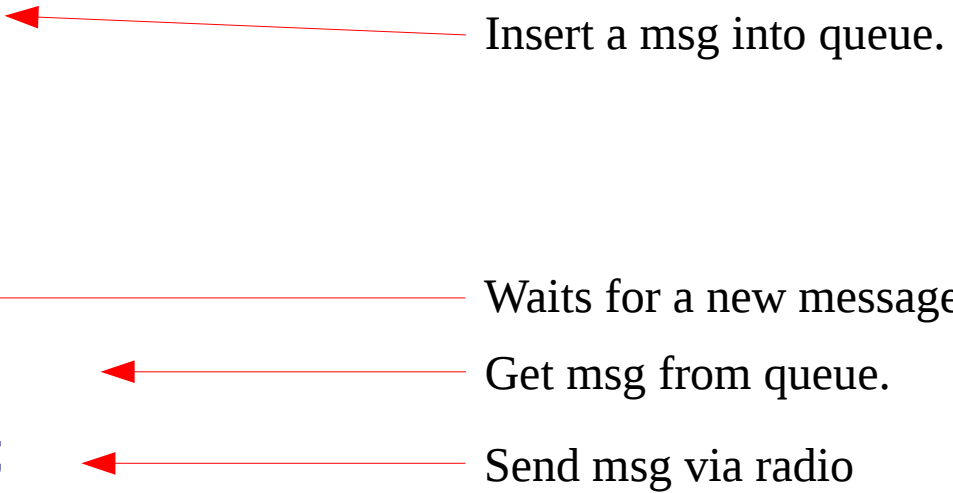
```
...  
var ubyte stat;  
par do  
    ...  
    stat=qPut(msgTemp);  
    ...  
with  
    loop do  
        await Q_READY;  
        stat = qGet(msgRadio);  
        emit SEND(msgRadio);  
        await SEND_DONE;  
    end  
end
```

Insert a msg into queue.

Waits for a new message

Get msg from queue.

Send msg via radio



Terra Local Operations

- Local operations extensions includes operations to access local inputs or outputs.
- Currently TerraNet implements:
 - TEMP – Temperature sensor
 - PHOTO – Luminosity sensor
 - LEDS – On board leds
 - VOLT – Battery voltage sensor
 - PORT_A/B – In/Out digital pin 1/2
 - INT_A/B – Interrupt pin 1/2

Terra Local Operations

Sensors

We need two steps to read a sensor. First we call an “**emit** <outEvent>();” command to start the A/D converter. Then, we wait for the results using an “xx=**await**<inEvent>;”. The 10 bits A/D converter always returns an u16 type var.

Terra sensor events: (outEvent x inEvent)

- REQ_TEMP x TEMP
- REQ_PHOTO x PHOTO
- REQ_VOLTS x VOLTS

Ex:

```
var ushort temp;  
emit REQ_TEMP();  
temp = await TEMP;
```

Terra Local Operations

Leds

It's possible to set the value for each led or all three values together. When setting a individual led value, you may write 'OFF' to have led off, 'ON' to have led on, or 'TOGGLE' to toggle the led state. The LEDS command uses the three least significant bits.

Terra Leds events: (outEvent)

- LEDS, LED0, LED1, LED2

Ex:

```
var ubyte count=0
```

```
emit LED0(ON);
```

```
...
```

```
count=count+1;
```

```
emit LEDS(count);
```

Terra Local Operations

Port A and B (only on Mica)

Currently Terra implements access to two I/O pin^(*) (port A and B). Each port has to be configured as input or output before the use. Reading a input port uses the two steps like to read a sensor. Configuring a port and setting a output port is like to set a led.

Terra Port events: (outEvent / inEvent)

- CFG_PORT_A
- CFG_PORT_B
- SET_PORT_A
- SET_PORT_B
- REQ_PORT_A / PORT_A
- REQ_PORT_B / PORT_B

Obs: Use 'OUT' and 'IN' constants to configure ports.

(*) PortA=6F and PortB=7F on MDA100CB sensor board.

Terra Local Operations

Interrupt A and B (only on Mica)

Currently Terra implements access to two interrupt pin^(*) (int A and B). Each pin has to be configured as rising or falling before the use. The interruptions are received by “await” command.

Terra Port events: (outEvent / inEvent)

- CFG_INT_A / INT_A
- CFG_INT_B / INT_B

Obs: Use 'RISING', 'FALLING', and 'DISABLE' constants to configure interrupt pins.

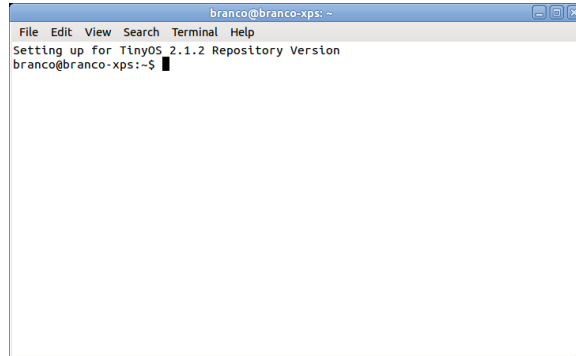
(*) IntA=5D and IntB=4D on MDA100CB sensor board.

Using Terra

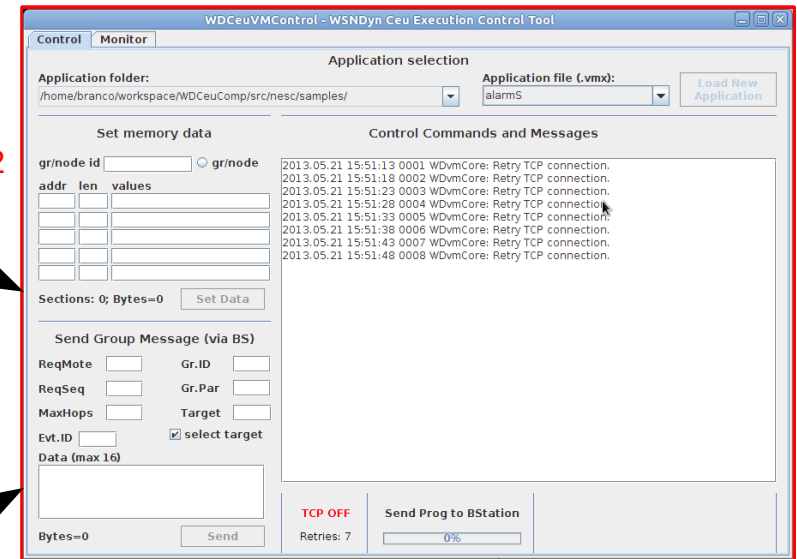
- Preparation
 - Upload TerraVM.exe to all nodes
- Application
 - Edit your Terra application
 - Compile it – > ./terracc app.terra (F5 in editor)
 - Load application using terravm java tool.
- Application Operation using an user java/lua app or terravmTool
 - Receive BaseStation Messages

Using Terra

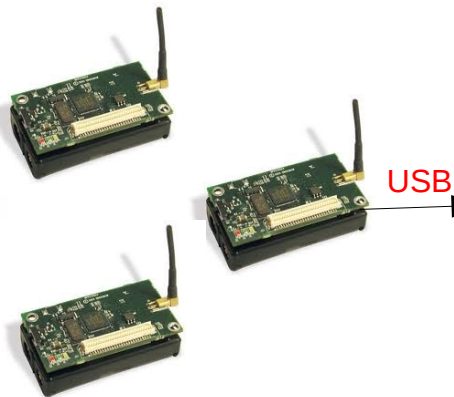
TOSSIM Python Script⁽³⁾



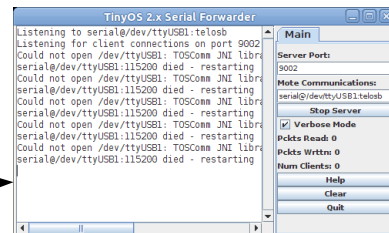
terravm Java Tool⁽²⁾



Simulator Real nodes



SerialForwarder⁽¹⁾



IP, porta 9002

Terra Comp.⁽⁵⁾

Network⁽⁴⁾

Commands:

```
home (1): java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB1:micasz
/tools (2): java -jar terravmcontrol.jar
/sim (3): ./TerraSim..py
/bin (4): ./load_mica.sh <USB id> TerraNet_v01_NOBS_micasz <id>
/terra (5): ./terrac xxx.terra
```

Linux environment

Simulator Viewer

VM Control Tool

The screenshot displays the TerraControl application window, which is divided into two main sections. On the left is the TOSSIM (Terra On-Site Simulation) window, titled 'Terra Simulation Viewer'. It shows a grid of 20 small windows, each representing a different component or process, with labels like '1', '11', '21', '31', '41', '12', '22', '32', '42', '13', '23', '33', '43', '14', '24', '34', and '44'. The status bar at the top of this window indicates '97s of 600s'. On the right is the TerraVMControl window, titled 'Terra Execution Control Tool'. It has three tabs: 'Control', 'Data', and 'Monitor'. The 'Control' tab is active, showing 'Application selection' with fields for 'Application folder' (set to '/home/terra/TerraNet_v0.1/src/') and 'Application file (.vmx)' (set to 'blink_tutorial'), and a 'Load New Application' button. Below this is a 'Control Commands and Messages' section displaying a log of system messages, including timestamps and messages like 'ControlCore: newData button.', 'ControlCore: sendNewProgVersion()', 'recReqFSMDDataMsg: Block: 2 - 1/5', etc. At the bottom right of the TerraVMControl window, there are controls for 'TCP ON' (with 'Retries: 3') and 'Send Prog to BStation' (with a '100%' progress bar).

TerraControl

TOSSIM

Terra Simulation Viewer

97s of 600s

1 11 21 31 41

12 22 32 42

13 23 33 43

14 24 34 44

TerraVMControl - Terra Execution Control Tool

Control Data Monitor

Application selection

Application folder: /home/terra/TerraNet_v0.1/src/ Application file (.vmx): blink_tutorial Load New Application

Control Commands and Messages

2014.04.29 10:17:41 0043 ControlCore: newData button.
2014.04.29 10:17:41 0044 ControlCore: sendNewProgVersion()
2014.04.29 10:17:41 0045 recReqFSMDDataMsg: Block: 2 - 1/5
2014.04.29 10:17:41 0046 ControlCore: sendNewProgBlock()
2014.04.29 10:17:42 0047 recReqFSMDDataMsg: Block: 3 - 2/5
2014.04.29 10:17:42 0048 ControlCore: sendNewProgBlock()
2014.04.29 10:17:43 0049 recReqFSMDDataMsg: Block: 4 - 3/5
2014.04.29 10:17:43 0050 ControlCore: sendNewProgBlock()
2014.04.29 10:17:44 0051 recReqFSMDDataMsg: Block: 5 - 4/5
2014.04.29 10:17:44 0052 ControlCore: sendNewProgBlock()
2014.04.29 10:17:45 0053 recReqFSMDDataMsg: Block: 6 - 5/5
2014.04.29 10:17:45 0054 ControlCore: sendNewProgBlock()

TCP ON Retries: 3 Send Prog to BStation 100%

Linux environment

Source dir

Text Editor

The screenshot displays the TerraNet Linux environment. On the left is a sidebar with icons for various system components. The main window is divided into two panes. The left pane shows a file manager with a sidebar containing 'Devices', 'Computer', and 'Network' sections. The 'src' directory is selected, showing files like 'blink_tutorial.terra', 'noop.terra', 'play.terra', 'blink_tutorial.vmx', 'noop.vmx', and 'play.vmx'. A status bar at the bottom indicates '"blink_tutorial.terra" selected (238 bytes)'. The right pane shows a text editor window titled 'blink_tutorial.terra (~/TerraNet_v0.1/src) - gedit'. It contains the following code:

```
1 @include(/home/terra/TerraNet_v0.1/terra/Terra.m4)
2
3 par do
4   loop do
5     await 1000ms;
6     emit LED0(3);
7   end
8 with
9   loop do
10    await 2s;
11    emit LED1(3);
12  end
13 with
14   loop do
15    await 4s;
16    emit LED2(3);
17  end
18 end
19
20
```

Below the text editor is a 'Compiler Output' window showing the following text:

```
running code: terra
Code size = 96 bytes.
Stack size = 0 (0 bytes).
Using 151 bytes of VM memory
Total of 3 message(s)

Done.
```

A 'Stop' button is visible in the bottom right corner of the compiler output window. The status bar at the bottom of the text editor shows 'TerraNet', 'Tab Width: 4', 'Ln 15, Col 16', and 'INS'.

Compiler output
(press F5 to compile)

TerraNet Motes

- Simulator (TOSSIM)
 - $n \times n$ MicaZ grid – neighbor radio range
 - TerraNet script – max size of 1488 bytes
 - All nodes execute the same script
- Real nodes (Testbed “alpha-test”)
 - MicaZ
 - TelosB

Tarefa Blink

- Selecionar o ícone 'src' para abrir a pasta dos arquivos fontes.
- Abrir o arquivo Blink_tutorial.terra
- Pressionar 'F5' para compilar o programa.
- Iniciar simulador Terra com 2 nós.
- Selecionar o ícone TerraControl.
- Carregar o Blink_Tutorial.
- Verificar o piscar dos Leds.
- Alterar os tempos no arquivo fonte, compilar e recarregar o programa.

Tarefa Monitor 1

- Faça um programa Terra em que o nó 11 envie periodicamente para a Estação Base (nó 1) o valor do seu sensor de temperatura.
- Opcional:
 - Teste a recepção do valor com o programa ex1.lua do diretório 'tossam'

Tarefa Monitor 2

- Assumindo uma numeração sequencial para os nós, faça um programa que:
 - O nó lê periodicamente seu valor de temperatura e envie para o nó com (NodeId-1).
 - O nó que receber uma mensagem de temperatura deve repassar para o nó (NodeId-1)
 - O nó 11 sempre deve repassar as mensagens para o nó 1 (BaseStation)
- Teste no simulador e no testbed e verifique se todas mensagens estão chegando.
- Imagine e implemente uma possível solução para evitar a perda de mensagens.

Mini projeto

Árvore geradora mínima + roteamento para raiz.

- Monte uma árvore geradora mínima com os nós da rede. Considere o nó ligado na BaseStation como nó raiz da árvore. (Nó 11 no simulador)
- A mensagem de dados de qualquer nó da rede deverá ser roteada até a BaseStation (Nó 1).
- Cada nó deverá enviar periodicamente o seu valor de Temperatura.
- Teste no simulador com redes de 4x4 e 8x8 e no Testbed.
- Tratar erros de colisão e perda de mensagens. Experimentar o uso de fila e envio aleatório ou sincronizado.

Final da Parte 3