

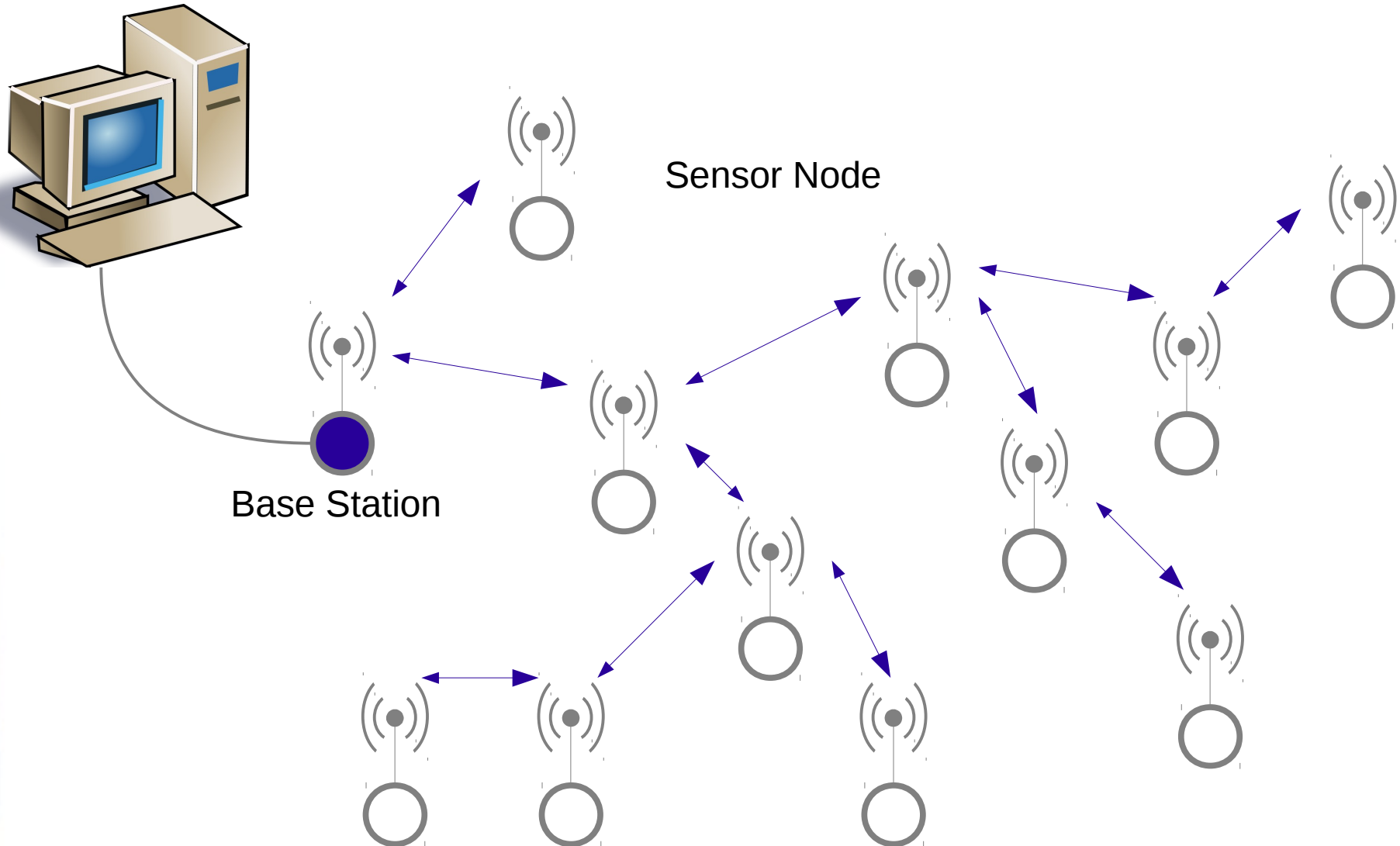
Terra System
Low abstraction level built-in functionalities
(TerraNet v0.2)

Introduction & user guide

Adriano Branco
abranco@inf.puc-rio.br

September, 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

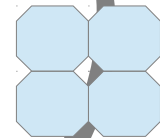
Terra

Céu Language

Reactive
programming
model
with safeties

Pre-built components

High and low level
abstractions for
network and
sensor operations



Embedded Virtual Machine



Sample Code

```
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
```

Infinite loop

Waits for sensor reads

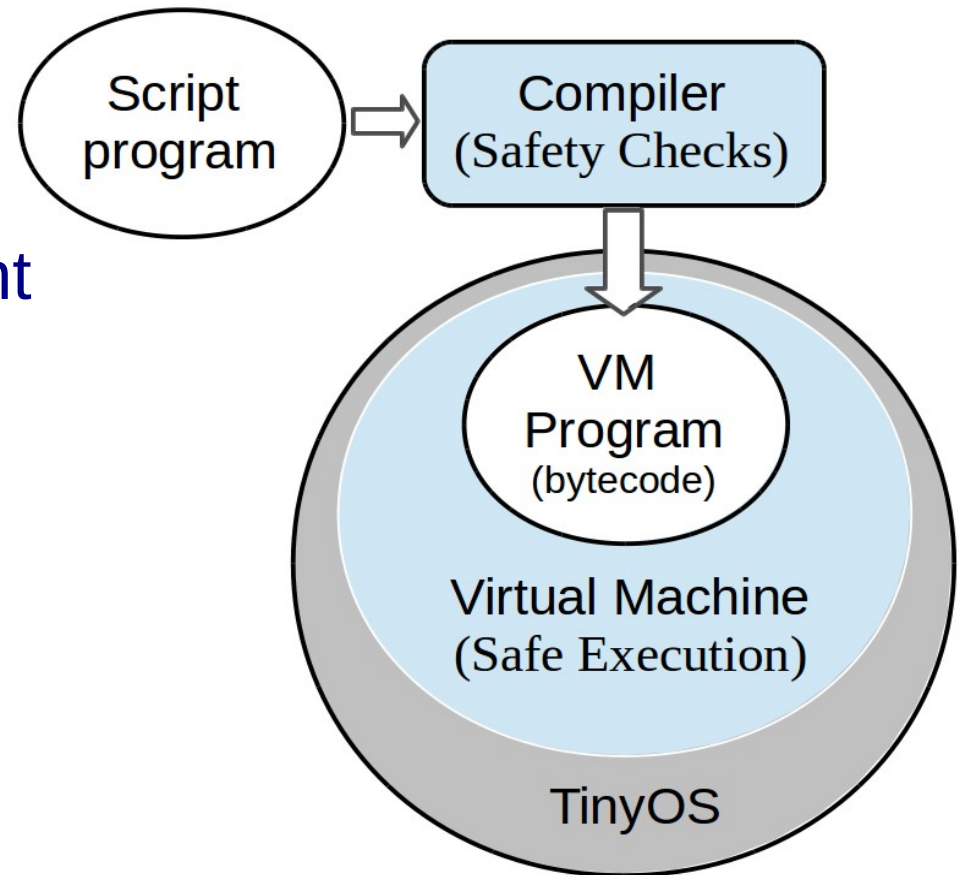
photo

temp.

Alarm

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
```

do-wait-continue

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

Periodic action

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

Time-out

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

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
with
  await 4h;
end
```

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(radioMsg)	Put msg into queue
	ubyte qGet(radioMsg)	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 – > ./terra app.terra (F5 in editor)
 - Load application using terravm java tool.
- Application Operation using an user java/lua app or terravmTool
 - Receive BaseStation Messages

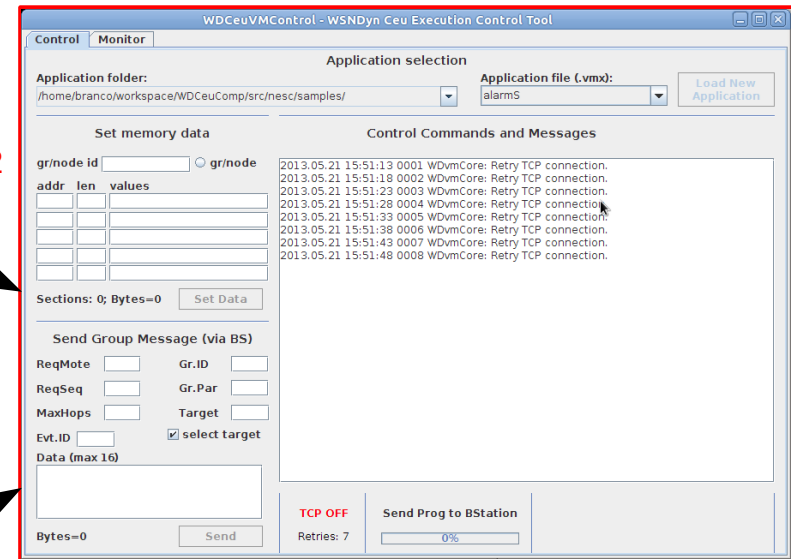
Using Terra

TOSSIM Python Script⁽³⁾

```

branco@branco-xps: ~
File Edit View Search Terminal Help
Setting up for TinyOS 2.1.2 Repository Verstion
branco@branco-xps:~$
    
```

terravm Java Tool⁽²⁾



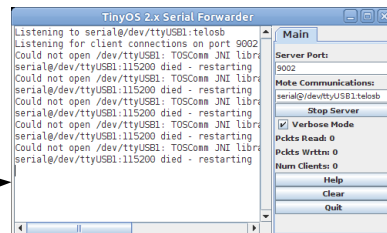
IP, porta 9002

IP, porta 9002

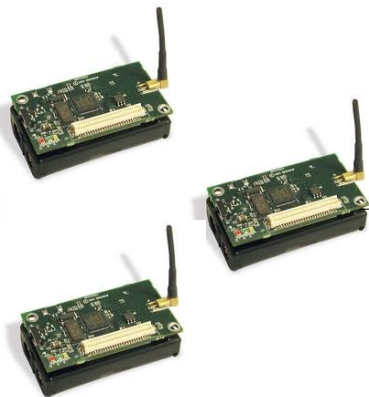
Simulator

Real nodes

SerialForwarder⁽¹⁾



USB



Network⁽⁴⁾

Terra Comp.⁽⁵⁾

Commands:
 home (1): java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB1:micaZ
 /tools (2) java -jar terravmcontrol.jar
 /sim (3) ./TerraSim..py
 /bin (4) ./load_mica.sh <USB id> TerraNet_v01_NOBS_micaZ <id>
 /terra (5) ./terrac xxx.terra

Linux environment

Simulator Viewer

VM Control Tool

The screenshot displays the TerraControl application interface. On the left is a vertical toolbar with icons for various functions: a gear, a folder, a globe, a terminal, a document with a pencil, and buttons for 'Terra Control', 'TOSSIM 2 x 1', 'Serial Forward Micaz USB0', 'src', 'Log', and a question mark. The main workspace contains two windows:

- Terra Simulation Viewer:** A window titled 'TOSSIM' showing a grid of 20 small circuit diagrams labeled 1 through 44. The text '97s of 600s' is visible at the top of the grid.
- TerraVMControl - Terra Execution Control Tool:** A window with tabs for 'Control', 'Data', and 'Monitor'. It features an 'Application selection' section with a dropdown for 'Application folder:' (set to '/home/terra/TerraNet_v0.1/src/') and another dropdown for 'Application file (.vmx):' (set to 'blink_tutorial'), with a 'Load New Application' button. Below this is a 'Control Commands and Messages' text area containing a log of system messages with timestamps and IDs. At the bottom, there are controls for 'TCP ON' (Retries: 3) and 'Send Prog to BStation' (100%).

Linux environment

Source dir

Text Editor

The screenshot displays a Linux desktop environment with a dark theme. On the left is a sidebar with various application icons, including a terminal, file manager, and Terra Control. The main window is split into three panes:

- File Manager (left):** Shows the contents of the `src` directory. Files include `blink_tutorial.terra` (highlighted), `blink_tutorial.vmx`, `noop.terra`, `noop.vmx`, `play.terra`, and `play.vmx`.
- Text Editor (middle):** A `gedit` 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
```
- Compiler Output (bottom):** A window titled `compiling code: terra` showing the results of a compilation:

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

Done.
```

At the bottom of the screen, a status bar shows `TerraNet`, `Tab Width: 4`, `Ln 15, Col 16`, and `INS`. A yellow tooltip at the bottom of the file manager indicates `"blink_tutorial.terra" selected (238 bytes)`.

Compiler output
(press F5 to compile)

TerraNet Motes

- Simulator (TOSSIM)
 - n x 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

Blink Example

- Select the 'src' icon to open source files folder.
- Open the file Blink_tutorial.terra
- Press 'F5' key to compile it.
- Start Terra simulator with 2 nodes. Right click the icon TOSSIM and select 2x1 option.
- Select TerraControl icon.
- Load Blink_Tutorial program.