Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : IO#: Linux Micro Framework .NET implementation

MASTER DEGREE: Bachelor degree in Telematics

AUTHOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rubio

DATE: October 19, 2013

Títol : IO#: Linux Micro Framework .NET implementation

Autor: Gerard Solé i Castellví

Director: Juan López Rubio

Data: 19 d'octubre de 2013

Resum

Aquest document conté les pautes del format de presentació del treball o projecte de final de carrera. En tot cas, cal tenir en compte el que estableix la "Normativa del treball de fi de carrera (TFC) i del projecte de fi de carrera (PFC)" aprovada per la Comissió Permanent de l'EPSC, especialment l'apartat "Requeriments del treball".

Title : IO#: Linux Micro Framework .NET implementation

Author: Gerard Solé i Castellví

Director: Juan López Rubio

Date: October 19, 2013

Overview

This document contains guidelines for writing your TFC/PFC. However, you should also take into consideration the standards established in the document Normativa del treball de fi de carrera (TFC) i del projecte de fi de carrera (PFC), paying special attention to the section Requeriments del treball, as this document has been approved by the EPSC Standing Committee

Escriure aquí opcionalment la dedicatòria.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The aim of this thesis is port an existing operating system called Micro Framework developed by Microsoft. Micro Framework is the smallest version of .NET for very resource-constrained devices. This port should let applications using this framework run on any Linux device capable of use Input/Output ports and SPI, UART communication standards. There is no official implementation of this framework for complete operating systems, in example, Windows or Linux.

The reason of this port is try to migrate a Wireless Sensor Network (WSN) Gateway that currently uses MicroFramework and Netduino Plus which is a constrained-device. But this gateway it's getting out of system resources, so in order to keep the existing code a solution has been proposed, make able to run this software on a Linux device.

In this case, the deployment device will be a RaspberryPi which is a low end computer similar to a Pentium II in terms of computing power. This computer offers a set of interesting things in terms of this thesis, basically it has exposed Input/Output ports as a GPIO. Over this GPIOs it's possible to use SPI, I^2C and UART communication buses.

After achieving this, there is a second goal which consists of help and test the development of a code translating tool called AlterNative which is being developed by Alex Albalá and Juan López. This translator is capable to get a source code from a C# binary and translate it to C++ trying to get better performance than C#. On the other hand, taking the benefit of the C++, the translated code should be able to execute between different operating systems (Windows, Linux and MacOSX and also mobile devices as Android).

CHAPTER 1. STATE OF THE ART: EMBEDDED OPERATING SYSTEMS

This chapter sketches out briefly the state of the art of the existing operating systems for embedded devices. The first part enumerates the different operating systems, explaining its important features. Next, focusing on MicroFramework, the different devices will be listed, which ports exist and finally, the existing implementations to use the Input/Output ports on a RaspberryPi.

1.1 Embedded Operating Systems

An operating system (OS) offers an interface with the hardware to make it independent from the applications that the device runs, making easy the interactions between hardware or other running programs.

An OS is an important program that makes easy to develop applications, but it is important to maintain the features that the processor offers, avoiding performance or capabilities degradation. As this bachelor thesis is focused on constrained-resource devices, where the processing capabilities and memory resources are limited, is fundamental to respect the above criteria.

In general, there are three types of operating system architectures based on how applications are executed:

- **Monolithic:** The OS and the applications are combined in a single program, being an end to end task without the possibility to include new functions without rewriting much of the code.
- **Modular:** The OS is running as a standalone program in the processor and has de ability to load programs to it self as modules. In terms of the development, it's possible to develop applications without writing in the core of the OS.
- **Virtual-Machine:** The OS creates an abstraction layer of its underlying hardware, this abstracted layer is common in every device that implements that virtual-machine. Using this type of operating system provides a helpful tool to achieve the well known slogan *write once, run anywhere*.

- **TinyOS:** asdf asdf asdf.
- **FreeRTOS:** asdf asdf asdf.
- **uC/OS II:** asdf asdf asdf.
- **Contiki:** asdf asdf asdf.
- **Micro Framework .NET:** asdf asdf asdf.

1.2 Micro Framework .NET

Complexity and heterogeneity drawbacks of distributed systems could be solved or relived using a middleware. Middleware is a system software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware, which provides facilities in order to build and use distributed systems [1].

This type of software provides a transparent and abstract vision of the low-level details (e.g. network communication, encoding, concurrency, protocol handling, etc.) facilitating end user programming. Middlweware typically provides two different types of transparency to distributed systems:

- **Access transparency:** Hides differences between remote and local operations like data representation and invocation mechanisms.
- **Location transparency:** Hides where the components reside. The different components could be redistributed (e.g. moved between computers) without changing any of the other components.

1.2.1 NETMF enabled devices

MicroFramework can run on CLR enabled devices that are MicroFramework compilant with it's specifications. In this bachelor thesis a Netduino Plus (from XXXXXLabs) has been used to test, understand and code sample code in order to know how Microframework works. Apart from this Netduino there are other devices like the cerbuino,which are also capable to run CLR code.

Among this, there are more powerful devices that can run and execute simple graphics programs.

1.2.2 Software Development Kit

As Micro Framework is similar to an operating system it has

- **4.3:** This asynchronous indirect communication model uses a queue in order to exchange messages. The messages from the producer are stored into the consumer's queue after being sent. In this type of model, persistent queues are used when the reliability is required in front of performance. Quality of service (QoS) policies are also a good solution to provide reliability.
- **4.2:** In this direct communication model, the messages are sent directly to the interested parts through publish/subscribe pattern. In this pattern, the different parts register interest in receiving messages on a particular message topic. After the subscription, the consumer will receive any message corresponding to the subscribed topic.

BlaBlaBla both blablabla

1.2.3 Visual Studio

1.3 RaspberryPi and MicroFramework

Before starting with the development a search was done in order to know if there was any project involving the port of the MicroFramework to Linux devices using, for example, Mono. Nothing was found. There is currently one implementation of MicroFramework for Linux, but it only works in a resource-constrained device called Edy Linux.

RaspberryPi has many implementations in different languages involving its IO ports, many are written in C and Python, others are for example in Java. But when speaking in terms of .NET/C# there is an important lack in IO implementations, below are exposed the most important ones that where found.

- **BlaBlaBla:**
- **BlaBlaBla:**

Although the XXX library is really interesting according to it's description of functionality, it doesn't

BIBLIOGRAPHY

[1] Schmidt, D.C. and Schantz, R.E., "Middleware for Distributed System - Evolving the Common Structure for Network-centric Applications", *Encyclopedia of Software Eng.*, Wiley & Sons, New York, 2001. Also available at http://www.agentgroup.unimore.it/didattica/ingss/Lec_Middleware/Schmidt_Middleware.pdf

[2] Bagula, A.B., Denko, M.K. and Zennaro, M., "Middleware for Mobile and Pervasive Services", Chap. 7 in *Handbook of mobile systems applications and services*, Taylor and Francis Group, Kumar, A. and Xie, B., pp. 248-249, Boca Raton (FL), 2012.

[3] Khan, S., Qureshi, K. and Rashid, H., "Performance Comparison of ICE, HORB, CORBA and Dot NET Remoting Middleware Technologies", *International Journal of Computer Applications*, 3(11), 15-18 (2010). Also available at http://www.ijcaonline.org/volume3/number11/pxc3871105.pdf

[4] López, J., Royo, P., Barrado, C., Pastor, E., "Applying marea middleware to UAS communications", *In Proceedings of the AIAA Infotech@Aerospace Conference and AIAA Unmanned Unlimited Conference 2009*, Seattle (WH). Also available at http://upcommons.upc.edu/e-prints/bitstream/2117/9248/1/infotech09.pdf

[5] López, J., "Service Oriented Architecture for Embedded (Avionics) Applications", *The PhD Program on Computer Architecture Technical School of Castelldefels Technical University of Catalonia*, Barcelona, 2011. Also available at https://dl.dropbox.com/u/2857619/thesis-small.pdf

[6] Kiely, D., "Delegates Tutorial" in *The Microsoft Developer Network (MSDN)*. Available at http://msdn.microsoft.com/en-us/library/aa288459(v=vs.71).aspx

[7] Albahari, J. and Albahari B., "Serialization", Chap. 17 in *C# 5.0 in a Nutshell: The definitive reference*, O'REILLY, Roumeliotis, R., pp. 691-728, Sebastopol (CA), 2012.

[8] Kiely, D., "Get Closer to the Wire with High-Performance Sockets in .NET" in *The Microsoft Developer Network (MSDN) Magazine*. Available at http://msdn.microsoft.com/es-es/magazine/cc300760(en-us).aspx

[9] Books Llc, Source Wikipedia, "Software Quality: Software Crisis, Kludge, Second-System Effect, Workaround, Reliability Engineering, Fault-Tolerant System", Books Llc, Memphis (Tennessee), 2011.

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APÈNDIXS

TÍTOL DEL TFC : IO#: Linux Micro Framework .NET implementation

TITULACIÓ: Bachelor degree in Telematics

AUTOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rubio

DATA: October 19, 2013

APPENDIX A. TOOLS

A.1 Source control tools

A.1.1 Git

A.1.2 GitHub

A.1.3 Git source control provider extension

A.2 Package management system

A.2.1 NuGet

NuGet is a free, open source developer focused package management system for the .NET platform intent on simplifying the process of incorporating third party libraries into a .NET application during development.

A.2.2 Packages

A.2.2.1 Log4net

Log4net, a port of the popular Java library log4j, is an open source library that allows .NET applications to log output to a variety of sources (e.g., console, files or SMTP). The information is logged via one or more loggers which provide a the following five logging levels:

- **Debug**
- **Information**
- **Warnings**
- **Error**
- **Fatal**

A.2.2.2 NUnit

NUnit, a port from JUnit, is a unit-testing framework for all .NET languages. It is written entirely in C# and has been completely redesigned to take advantage of many .NET

language features, for example custom attributes and other reflection related capabilities.

NUnit does not support Visual Studio integration. Instead of this it provides an external program compiled either as a console app or a GUI. This program is able to runs and execute the unit tests from an assembly.

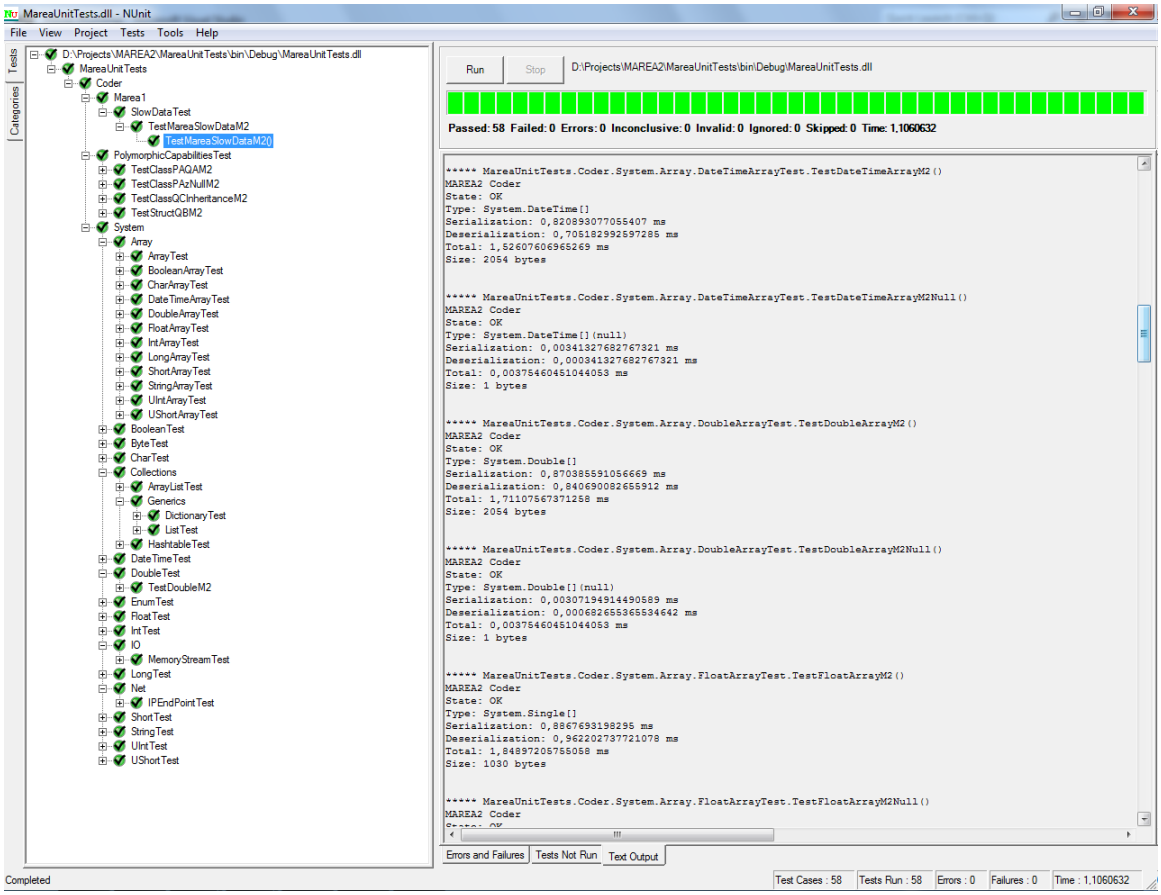

Figure A.1: MAREA unit tests executed by NUnit GUI application

This framework has been especially used to test encoder layer functionalities. The listing A.1 shows an example of a unit test to serialize and deserialize a double with two diferent pair of parameters.

Listing A.1: MAREA encoder layer unit test: serialization and deserialization of a double

```
private byte[] seralizedData = null;
private long start, serializeTicks, deserializeTicks;
private long  clock_freq = PerformanceTimer.Clock_freq();

[SetUp]
public void RunAfterAnyTest()
{
 serializeTicks = 0;
 deserializeTicks = 0;
}

[TestCase(0.100000234523, 0), NUnit.Framework.Description("Coder(double , System.Double)")]
[TestCase(double.MaxValue,0)]
public void TestDoubleM2(double oDouble, double rDouble)
```

```
{
 for (int i = 0; i < CoderTestsConstants.CODIFICATIONS; i++)
 {
  start = PerformanceTimer.Ticks();
  seralizedData = AdaptedMareaCoder.Send(oDouble);
  serializeTicks += PerformanceTimer.TicksDifference(start);

  start = PerformanceTimer.Ticks();
  rDouble = (double)AdaptedMareaCoder.Receive(seralizedData);
  deserializeTicks += PerformanceTimer.TicksDifference(start);
 }

 Console.WriteLine(CoderTestsConstants.MAREA2);
 Results results = ResultsManager.GetResults(serializeTicks, deserializeTicks, clock_freq,
 CoderTestsConstants.CODIFICATIONS, seralizedData.Length, rDouble.GetType().FullName);

 if (oDouble == rDouble)
 {
  Assert.True(true);
  Console.WriteLine(CoderTestsConstants.OK_STATE);
  Console.WriteLine(results.ToString());
 }
 else
 {
  Console.WriteLine(CoderTestsConstants.KO_STATE);
  Assert.True(false);
 }
}
```

A.2.2.3 Nuget Server

A.3 Cross platform tools

A.3.1 Mono

A.3.2 Alter Native

APPENDIX B. EXEMPLE DE PROVA D'UN APÈNDIX

Text de prova