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

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : IOSharp: Implementació de Micro Framework a Linux

MASTER DEGREE: Bachelor degree in Telematics

AUTHOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rubio

DATE: November 4, 2013

Títol : IOSharp: Implementació de Micro Framework a Linux

Autor: Gerard Solé i Castellví

Director: Juan López Rubio

Data: 4 de novembre 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 : IOSharp: Linux Micro Framework implementation

Author: Gerard Solé i Castellví

Director: Juan López Rubio

Date: November 4, 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. INITIAL PROPOSAL

This project was proposed by AlterAid a company which is working on several ways to help in taking care of the health of our elderly, or in general, anyone that is relevant to our lives.

This company is working on two different projects that combine together, the first one is called aaaida which consists of a social network where people can stay alert about its relatives, upload information about its health or watch recommendations from doctors. On the other side, and more hardware oriented development, they are creating a Sensor Network called HomeSense that once deployed in a house will be able to collect relevant information from those sensors in the home and allow other people to know if the daily life is going normal, or something is happening.

1.1 AlterAid products in depth

As explained before, AlterAid has two relevant products, aaaida which is the unifying social network that shows or even analyzes the data uploaded there. Then comes HomeSense, a Wireles Healthcare Sensor Platform, which is interesting on terms of sensing and daily life control. HomeSense idea is be capable of pick information using distributed sensors around a house and then upload to aaaida.

1.1.1 Aaaida

1.1.2 HomeSense

1.1.3 Case of use

Alice is a young teenager whose grandfather, Bob, is ill and she wants to know if all is fine in Bob's home life. Alice will sign up in http://www.aaaida.com, there she will create a bond called Bob. A bond is a entity that represents a person, this entity can be configured with different measures. Then Alice will create 2 measures, the first one will be blood pressure while the other one will be bob's house. In blood pressure, Bob will use a simple elderly-oriented mobile application in order to upload his blood pressure every day, in this manner Alice can be aware of its health. Apart from this, Alice will buy a product from AlterAid, called HomeSense, which consists of a set of sensors that must be installed in doors, walls, drawers..., a small centralized system that must be plugged to the power and an application to configure HomeSense. This application will facilitate the linking between Alice account and Bob's bond, in addition, Alice will be able to setup a internet link to HomeSense using grandfather's Internet or a GSM connection to some mobile phone provider. Once the system is configured and working, Alice will be able to take care of Bob's home life, for example if he must take a pill at the morning, she can control at least if the medicine cabinet has been opened. Or if all doors are closed if Bob is out of home.

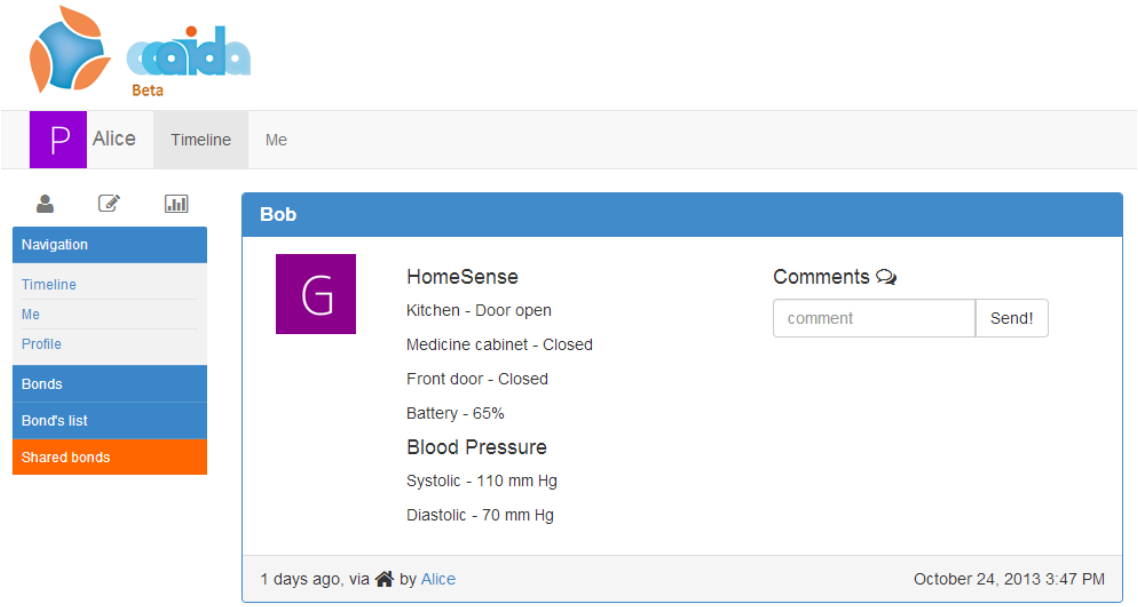

Figure 1.1: Case of use representation in aaaida

1.2 AlterNative

blablabla AlterNative

1.3 Thesis Proposal

CHAPTER 2. STATE OF THE ART

This chapter sketches out briefly the state of the art of the embedded operating systems and its capabilities. Then according to this thesis it will be explained what is the current operating system running on bottom of HomeSense and finally why has been chosen the RaspberryPi as the target device.

2.1 Embedded Systems

Embedded Systems now a days are taking relevance again with the Internet of the Things, environment sensing, Wireless Sensor Networks and all new coming technologies that require low power consumption, small size, mobility environments, ...

In Embedded Systems or Resource Constrained Systems it is interesting to take a look into the Hardware platform and its capabilities, the differences between platforms, and also which tools or unique features offers to developers.

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 and the programs running on the machine.

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. 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.

2.1.1 Operating Systems Architectures

In general, there are three types of operating system architectures for embedded devices, which are based on how applications are executed or included into the OS.

- **Monolithic:** The OS and the applications are combined into a single program. Normally in this situations the embedded device runs in the same process the OS and the program written to it. This type of architecture makes difficult 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. Normally using modules developers can expand the capabilities of its software.
- **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*. Although using virtual-machine devices simplifies the development on multiple devices, the performance of the platform normally will be reduced and in Real Time environments it isn't recommended to use it.

2.1.2 Embedded Operating Systems

There is a wide range of Embedded Operating Systems each of them has strengths and weaknesses, below different OS are described and compared.

- **TinyOS** is a popular open source OS for wireless constrained devices, many of them used in wireless sensor networks. It provides software abstractions from the underlying hardware. It is focused on wireless communications offering stacks for 6LoWPAN and ZigBee. It also supports secure networking and implements a RPL taking in mind the forthcoming routing protocol for low power and lossy networks.
  However, TinyOS changes how programs should be developed, it intended to use non-blocking programming which means that it isn't prepared for long processing functions. For example, when TinyOS called to send a message the function will return immediately and after a while the send will be processed and after then, TinyOS will make a callback to a function, for example send()'s callback will be sendDone().

- **FreeRTOS** is a free real-time OS that supports over 34 architectures and it is being developed by professionals under strict quality controls and robustness. Is used from toys to aircraft navigation and it is interesting for its real-time qualities. It has a very small memory footprint (RAM usage) and very fast execution, based on hard real-time interruptions performed by queues and semaphores. Apart from this, there are not constraints on the maximum number of tasks neither the priority levels that can be used on tasks.

- **Contiki** is similar to TinyOS in terms of portability between platforms and its code is open source. It also offers features similar to standard operating systems like threading, timers, file system and command line shell and uses modular architecture, loading or unloading programs from its kernel. Contiki is build on top of the Internet Standards supporting IPv4 and IPv6 and also the new low-power internet protocols which includes 6LoWPAN, RPL and CoAP.
  Contiki uses protothreads which are designed for event-driven systems running on top of constrained devices, which is the case of Contiki's kernel. It provides blocking without having a real multi-threading system or a stack-switching.

- **Micro Framework .NET** is a solution provided by Microsoft for constrained devices which cannot execute the full .NET stack. It is Virtual-Machine based operating system that a small implementation of the CLR making available to execute a small set of .NET classes. Its memory footprint is about 300KB and supports the common embedded peripherals like EEPROM, GPIO, SPI, UART, USB, ...
  One of its interesting features is that offers the advantages of .NET language using Visual Studio and it also offers real-time debugging directly on the device.

2.2 Micro Framework .NET

MicroFramwork, is also known as NETMF, had its roots in a project called **Smart Personal Objects Technology (SPOT)**. The first devices implementing the SPOT technology where smart-watches from Fossil and Suunto in 2004 and after them became kettles, weather stations and for traffic and map updates in Garmin devices. Microsoft wanted to create a technology for everyday devices so they launched together with SPOT the MSN Direct which was a set of network services capable of delivering information to the SPOT devices using FM radio broadcast signals.
In 2008 the production of SPOT watches was discontinued and in 2009 Microsoft released the source code of Micro Framework under Apache 2.0 license making availably to the community and shortly after this release the MSN Direct services where ceased.

2.2.1 Devices using Micro Framework

Since Microsoft released the Micro Framework source code, different companies had created different devices supporting .NET code and this stack.
There are two major vendors producing chips and development kits for this software. Secret Labs produces the netduino family which consists of the standard netduino, a netduino plus which is an enriched version. This one has better processor and memory, it includes an ethernet port and uses micro sd cards to provide storage. One of the interesting this of this two boards is the layout of the board, it is totally compatible with most of the arduino shields in the market. Secret Labs has another board called netduino go, which is similar to netduino plus but without storage, ethernet and it does not use the typical arduino layout, so a globus module is required to use arduino shields.
GHI Electronics is another hardware manufacturer that has designed and released different boards implementing Micro Framework or modules which its target platform is Micro Framework. GHI has a very wide range of products for example FEZ Cerbuino Bee and FEZ Cerbuino NET which are similar to the netduino plus in terms of performance. One interesting thing of FEZ devices over netduino is the possibility of load native code (C/Assembly) for real-time requirements which it is really interesting. For example HomeSense could run its Mesh driver in C and perform better than it performs using C#.

In addition to the mentioned manufacturers, Microsoft Research in Cambridge has defined a hardware reference platform called .NET Gadgeteer which defines how boards and modules must be in order to allow rapid prototyping of projects. Gadgeteer boards and modules share the same layout and connector schemes.

2.2.2 Linux port of Micro Framework

2.3 CLR on RaspberryPi

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

CHAPTER 3. 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.

3.1 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.

3.2 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.

3.2.1 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.

3.2.2 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

3.2.3 Visual Studio

3.3 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 : IOSharp: Implementació de Micro Framework a Linux

TITULACIÓ: Bachelor degree in Telematics

AUTOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rubio

DATA: November 4, 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