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

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : IOSharp: MicroFramwork a Linux

MASTER DEGREE: Grau en Telemàtica

AUTHOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rúbio

DATE: December 13, 2013

Títol : IOSharp: MicroFramwork a Linux

Autor: Gerard Solé i Castellví

Director: Juan López Rúbio

Data: 13 de desembre 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: MicroFramework in Linux

Author: Gerard Solé i Castellví

Director: Juan López Rúbio

Date: December 13, 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 HomeSense

HomeSense, as it has been introduced above, is a Wireless Healthcare Sensor Platform created with the aim of control and take care of elderly and relatives, actually it is composed by a Netduino Mini which makes the function of the gateway controlling the sensor network, receiving all the data and uploading to aaaida platform through internet.
In the house the communication is carried on using little sensors capable of fetching data in different situations for example a drawer or a medicine cabinet where medicines are located, in different doors in order to know if it is opened or closed or any application interesting on acquiring information from he environment, house or person. This sensors make use of low-power 2.4GHz ISM band from Nordic Semiconductor.
The communication protocol is similar to a mesh network with multi-hop communication, but is interesting to view it also as a star where the important information is sent to the gateway which is in charge of make the information available.

1.1.2 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.

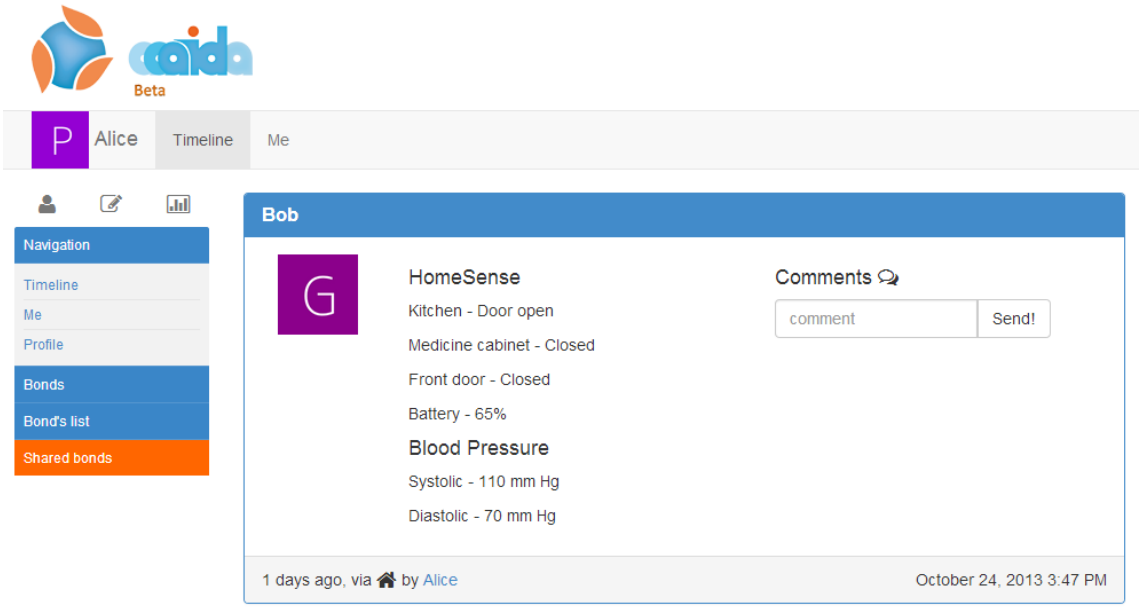

Figure 1.1: Case of use representation in aaaida

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 and are open to any company that wants to build products using those schematics.

2.2.2 NETMF on Linux

After doing some research about implementations of Micro Framework on other devices or running on top of other operating systems such as Linux. It was found a project that is currently porting NETMF to Linux, but for a very specific device called Eddy.

Eddy is an ARM embedded device board which uses Linux, the port named above has been made as a demonstration of writing NETMF applications using a port on top of other operating systems. One of the majors problems of this port is that some drivers are not working at all, for example UART, SPI and I2C which are 3 interesting I/O protocols and ports for HomeSense.
Although this port is for the Eddy board, it can be ported to other devices using the appropriate cross-toolchain, anyway it seems that there is a lack of possibilities to run Micro Framework code in other devices or operating systems.

2.2.3 NETMF on RaspberryPi

If it is hard to find an implementation of NETMF in Linux it will be harder to find an implementation for RaspberryPi. In other words, people in GHI forums are asking for NETMF ports for RaspberryPi but no one exists.
aqui nombrar altres implementacions en C# que acostumen a ser bastant escaces tambÃ©

2.3 Micro Framework in other devices

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. IOSHARP

In this chapter it will be explained how was defined and architected, designed and implemented the core of IOSharp disaggregating the different parts and explaining each one.

First of all it will be explained the project design explaining which two options where at project definition, then the implementation is explained for the different IO ports and communication standards used in this project. Finally, it will be explained how is done the port mapping to work between different boards and devices.

3.1 Planning the development

At project start, the development was focused on a tiny Linux board called RaspberryPi. This device was designed by RaspberryPi Foundation in England taking in mind the kids around the world and helping them to provide cheap tools to be introduced in computer science. It is an interesting board for its features, offering basic IO using the provided GPIOs, it also provides a SPI module for peripheral communication, an I^2C and UART interface also for transmissions between external components and the device itself.
In addition to the interfaces mentioned above, it also has some desktop interesting features like USB ports which practically can accept any device that works on Linux for instance a WiFi, Bluetooth, ZigBee or any stick for wireless transmissions, HDMI for graphics and user interface and Ethernet for network communications. Apart from this IO characteristics, it also mounts a decent ARMv6 (CPU) running at 700MHz on stock frequency and being overclocked to 1GHz without problems. Together with the CPU 512MB of RAM are provided which is enough for normal desktop usage (surfing, emailing and office) and for embedded projects.

After choosing the target device, two implementing options where designed and analysed. Each one has its own benefits and problems that are going to be explained in the following sections. As an introduction to the options that were considered where the use of the specific tools and libraries for the RaspberryPi and its CPU, this should help on achieving high efficiency when deployed on the device. On the other hand, trying to develop the project as widely as possible to give the chance of running under any Linux (or similar operating systems like Unix or Android) avoiding the exclusion of other devices.

3.1.1 Focused on RaspberryPi

Initially IOSharp was started taking in mind the Raspberry Pi computer, a library written on native C was found. Using this library let native control over all the features provided by its CPU including a wide range of pins for the GPIO, different protocol communications like SPI and UART and other features like PWM in some pins.
This methodology is interesting when is important to achieve high performance on the execution of the programs that make use of its hardware, in this case the modules of the CPU are used on a low-level way by changing its registers. This normally let the programs

run faster, but in principle this is not required for HomeSense usage, although later in this thesis it will be explained that the performance of HomeSense has not been as it should be.
The library written for the bcm2835 is the one that should be used in case of this methodology is chosen. The idea is to make calls from C# to this library using a specific call methodology that will be explained on future sections on this thesis.

3.1.2 Focused on Linux in General

Since the design of IOSharp was focused on reach any device capable of execute C# and not only the RasperryPi the best solution was to build this library in a way that doing a minimal set of changes the portability statement was accomplished.

3.2 Implementation

Remember that as it was explained on the introduction, the goal of this thesis is deploy and run successfully the HomeSense platform on a RaspberryPi. Taking in mind this, some modules where required to develop in order to accomplish with this requirement. In this case the GPIO, Interrupts, SPI and UART need to work in order to use the different components of HomeSense.

3.2.1 GPIO

In order to implement the GPIO ports in NETMF it will be used the IOPorts.cs file which contain the structure for Input, Output, Tristate and Interrupt ports, the first three will be explained in this section whereas the interrupt port will have a dedicated one.
Below is explained the different options on implementing GPIOs in Linux, which has been chosen and how has been implemented in this project.

3.2.1.1 Implementation Options

GPIO acronym stands for General Purpose Input Output which are Ports on systems that are capable of generating an output or reading an input. Normally embedded systems work with ports at 3.3V, other devices had low power supply like 2V. And it is not strange that many of them are tolerant to 5V in input.
In case of the Netduino and RaspberryPi the ports on both devices run at 3.3V and also the different test hardware modules used in this thesis.

The GPIOs in Linux can be controlled in several ways, the most common and simple is use the userspace which stands for a set of directories with readable and writeable files representing the ports of the MCU or CPU. On the other hand the Linux kernel also provides a library and a module in order to control the different pins.

The decision must be done between this two systems, in order to use any of this solutions the GPIO must be activated in kernel, many desktop Linux distributions have GPIO disabled and that's why the kernel must be recompiled enabling this feature. Basically GPIOs and SYSFS must be activated on kernel configuration, after compiling and installing the new kernel both methods will be enabled.
In case of Linux operating systems designated for embedded devices, for example the RaspberryPi or CubieBoard, will have the Input/Output Ports enabled by default.
One of the most curious things that where found at studding this possibilities was that Android is capable to use this ports. Although it cannot seem an interesting feature nowadays android is everywhere and can run in many devices, so is another reason to try to fetch this this sector in future versions of this software.

3.2.1.2 Using GPIO from SYSFS

Since each solution can be used in this project and both are available in any Linux OS when the kernel is properly compiled the chosen option was control the GPIO through the SYSFS in order to simplify both the development and the testing.
Use the Input/Output ports using the userspace is really simple and that one of the biggest reasons, the easiness of testing and debugging the functionality. It is easier to read or write a file in order to test if the control of the GPIOs is successful rather than debugging a C function embedded in a library.

As it was said before, in userspace the control of the GPIOs is carried by several files and directories located under */sys/class/gpio* directory. In this directory there are two files which are called export and unexport, the first one is used to enable a GPIO while the second one will be used to disable it. After enabling a GPIO a new folder will be created representing the enabled port, for instance if port 2 is enabled, a folder called gpio2 will be created. Inside this new folder there will be several files, the direction file describes how port should work, if the desired function is as an input port an "in" must be written in the file whereas "out" must be write for an output port. After setting the port direction the value file comes in which will do the functions of reading the port in case of input ports, or write 0 or 1 through it if the port is described as an output. To do this, just read this file to read an incoming value, or write 1 for active-high or 0 for active-low.

3.2.1.3 Implementing in NETMF

Taking in mind that this implementation has to be done over the existing code extracted from the *IOPorts.cs* is important to design how to do it properly, in this case a **GPIOManager** has been created using a singleton pattern, in order to restrict one instantiation of this class among all the code, and simplify the use of this class.
This manager will be in charge of enabling, disabling and operating the different ports. Also it will control which ports are enabled in order to avoid problems, i.e instantiating the same port twice will make the GPIOManager throw an exception so it can prevent hardware damages.
For GPIO implementation Output, Input and Tristate ports will be required, below you can see how look the classes provided by NETMF. This classes will consume the GPIOMan-

ager explained above.

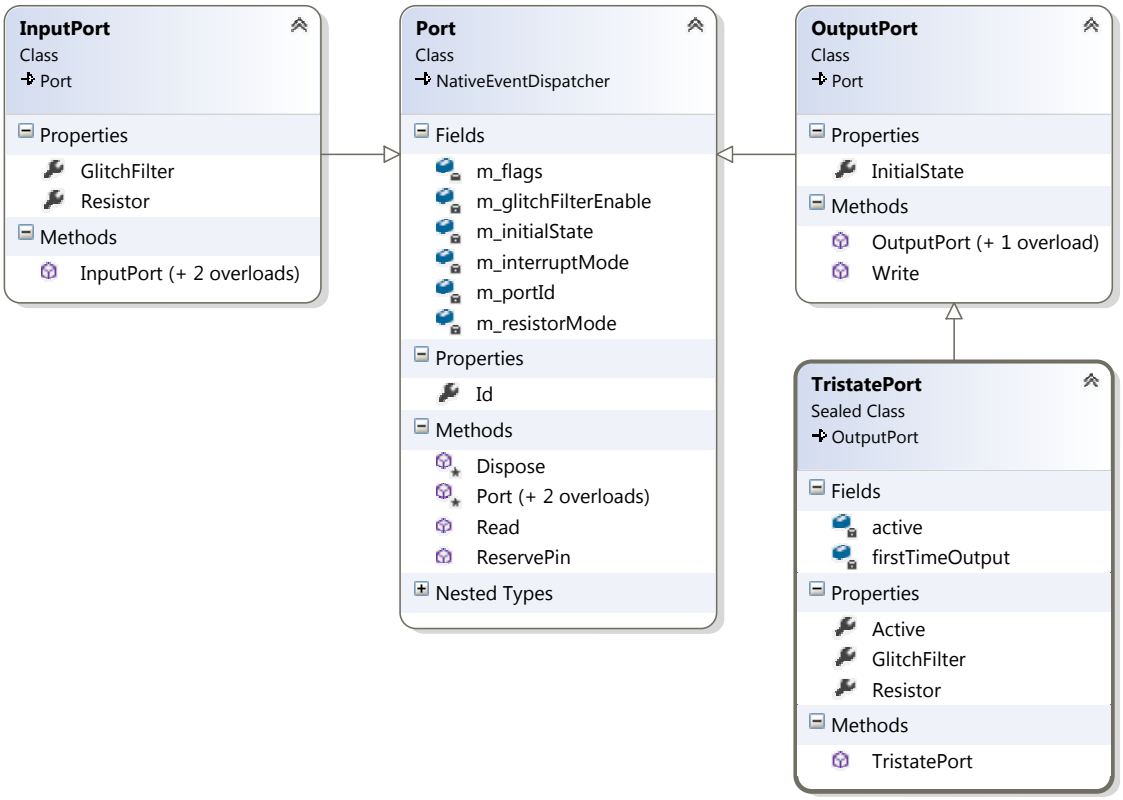

Figure 3.1: UML Diagram of NETMF Port and its inheritance

As is shown each Port type inherit from Port object which implements the methods to enable or disable a port (Port and Dispose methods), Read which is used to obtain the current state of the port i.e read an input value or know in which state is configured the output port. Finally it also has a ReservePin and as its name says, it is used to reserve a pin for future usage. Taking a look on the left box it can be seen the InputPort which inherits from Port. It does not have any special method and its constructor suppers to Port class. On the right side there is an OutputPort also inheriting from Port which implements a new method called Write that its used to write a state through the port, active high or active low. TristatePort also inherits from OutputPort, a NETMF TristatePort is a type of port capable of commuting between input or output port, i.e with a TristatePort is possible to control 2 leds which one of them is connected in pull-up and the other one in pull-down, so with this TristatePort is possible to light one, the other, if it is configured as an output port either on active high or low, or turning the TristatePort to an input port will avoid the lightning of any led.

3.2.2 Interrupt

After implementing the GPIO and taking in mind the requirement of HomeSense an interrupt system was needed for the future SPI code block. Although the BCM2835 supports native interruptions via IRQ at the time of this project the RaspberryPi did not support GPIO

interruptions from the Linux Kernel using IRQ, but when this thesis was being written this interrupt types where implemented. It was decided to maintain the current implementation to avoid problems of the same kind in other platforms and devices.

3.2.2.1 Designing the Interruptions

The interruption system will be written in C because of the simplicity to detect an interruption on a GPIO port, after doing some research a solution was found which uses a C function called poll, this is commonly used by developers that want to intercept GPIO interruptions in Linux environments where IRQ interruptions from GPIO are not available. The poll function is configured to wait certain events on a File Descriptor obtained from the GPIO file enabled on the SYSFS in a similar way as it was explained in the 3.2.1 GPIO section. The poll is configured to wait the execution of the file until the POLLPRI event is detected. This event will be triggered by the OS when the file had urgent data to be read and the poll function will collect this event and then will continue executing code block which basically will read the state of the port (active high or active low). As it is really important to notify the running program of an interruption on the port, a delegate will be used, this delegate is configured by the developer on the `OnInterrupt` using NETMF and basically it contains three parameters which are the port, the state and the time of the event.
The interruptions can be disabled at any time and the port can also be disposed.

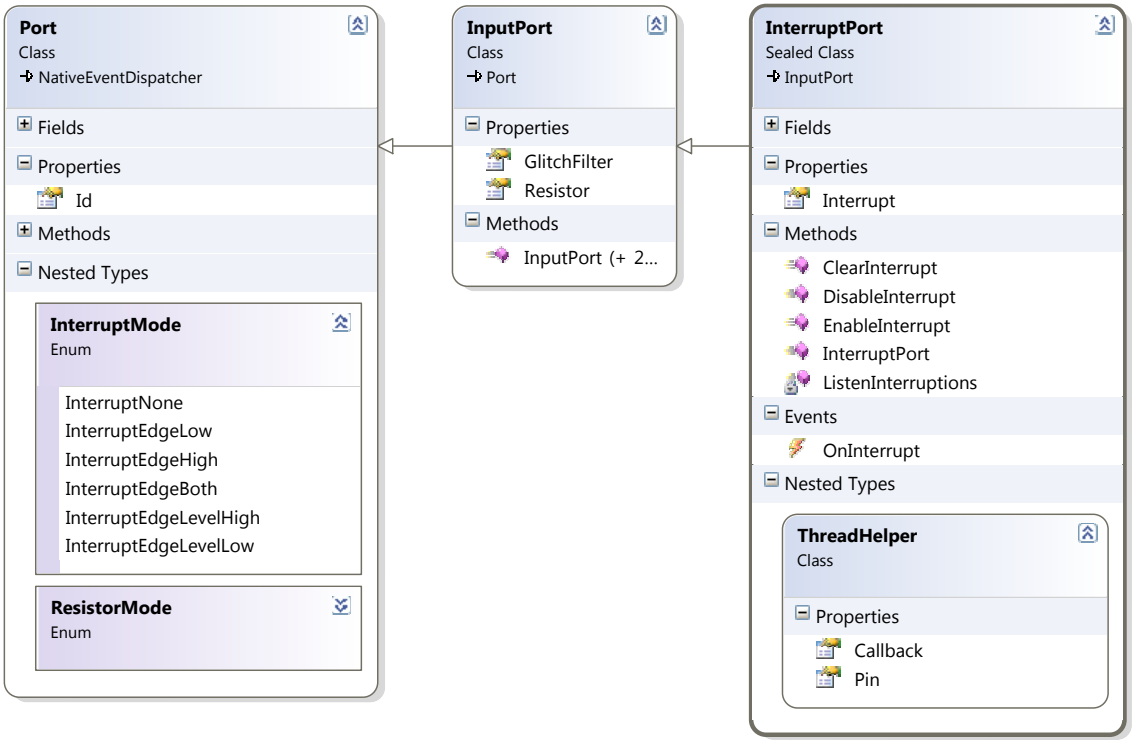

Figure 3.2: UML Diagram of NETMF Interrupt Port

3.2.2.2 Platform Invocation Services

In C# is possible to invoke external libraries which they do not have to be written in the same language, in this case the library has been written in C and then compiled into a shared library making it available to any program suitable for it, IOSharp in this case.
The Platform Invocation Services is the methodology that has been chosen to do this external calls to written libraries. This calls are known as P/Invokes and are used to call unmanaged code from managed code.

- **Static Library**
  This kind of library is that one that is imported while programming, and when the source code is the library is linked statically in the generated binary, this means that the compiler takes the library functions used along the program and then are embedded statically to the compiled binary, in this way, the functions are fiscally located with the program.
  For example, imagine that you create a library with 3 functions, called `get_password()`, `generate_token()` and `hash(char[] plain())`, but in the current project you only use two of this functions, which are `get_password()` and `get_token()`. When the compiler builds the project code will only take the functions from the library that are currently used in the program and then insert them into the binary. At this point, deleting that library won't affect the generated binary, and it will be able to use it without any dependency problems related to missing libraries (dependencies).
  The typical files for this type is .lib for Windows and .a for UNIX systems.

- **Dynamic Library**
  In order to avoid the replication of libraries that occurs in static ones, the dynamic libraries were created. This type of libraries is normally used along the operating systems to let applications use the offered functions and APIs written from the OS. In this case, and instead of the functional way of static libraries, any function from the library will be embedded on the generated binary, in this case is important to check that the dependencies are well satisfied when the binary is used, because a missing dependency will break the execution.

  In this case, the usual extension files are .dll for Windows and .so for UNIX. But .so files are also used in Windows, especially in web browsers, which use this types of library to load browser plugins such us Flash.

  There are two subtypes of dynamic libraries which are explained below:

  - **Dynamically Linked**
    These libraries must be available at compiling/link phase, because the compiler will verify that the function exists and that it is used properly. The libraries will be loaded at start time of the program. In this case, all the functions are mapped into the code.
  - **Dynamically Loaded**
    Instead of the previous library, the dynamic loading is used by programs to load or unload libraries and use its functions at run time. When the program needs to use a function it loads the library, then it uses the required functions and finally the library is unloaded again.

Pros and cons

The main problem of using static libraries is that the compiled binary takes much more memory and the library is embedded in every program that needs some functions from that library. But, on the other hand, using static libraries the access to its functions by programs is much fast than dynamic ones. Using them also avoids dependency problems, because the dependencies are embedded instead of being located in the file system like dynamic ones.
Regarding the dynamic libraries, they help to avoid replications and memory consumption, and also help to maintain the library updated in all programs that use them, although this can seem pretty good, it can have two bad effects into the generated program. First of all, if the library is missing in the system, the program will not run or will crash in execution time. Secondly, if the library is updated but some methods are changed, the program will crash because the non-existing function, and it will be necessary to readjust the code again, recompile and redistribute it.

P/Invokes in .NET makes use of dynamic loaded libraries in order to use the contained functions. The implementation difficulty of P/Invokes increases on how complex is the function to be called regarding its parameters, for basic type parameters such us `int`, `long`, `byte`, etc is really simple to make a P/Invoke call, but when passing object parameters things get much difficult because they must be passed to unmanaged code which sometimes can be impossible to do without using structs as interchange objects.

Below is shown the important parts of the implementation of the library and how P/Invoke is done in C#.

Listing 3.1: IOSharp.c - Polling function

```
uint64_t start_polling(int pin) {
    struct pollfd fdset;
    int nfds = 1;
    int gpio_fd, timeout, rc;
    char * buf[MAX_BUF], c;
    int len, count, i;
    long t;

    // Get the File Descriptor for the GPIO Port. See function on the Library.
    gpio_fd = gpio_fd_open(pin);

    // Clear any initial pending interrupts
    ioctl(gpio_fd, FIONREAD, & count);
    for (i = 0; i < count; ++i)
        read(gpio_fd, & c, 1);

    // Fill fdset which is a struct for pollfd which is used to describe the polling system.
    // In this case the File Descriptor for the GPIO port is entered, and then the POLLPRI (Data↩
        Urgent to Read) is configured as the event type.
    fdset.fd = gpio_fd;
    fdset.events = POLLPRI;

    read(fdset.fd, & buf, 64);

    // Start polling the File Descriptor. POLL_TIMEOUT variable contains (-1) which stands for ↩
        infinite blocking until event.
    rc = poll( & fdset, 1, POLL_TIMEOUT);

    // Close the GPIO Port. See function on the Library.
    gpio_fd_close(gpio_fd);
    return t;
}
```

Listing 3.2: IOSharp.h - Header file for the library

```
#ifndef IOSHARP_H_INCLUDED
#define IOSHARP_H_INCLUDED

// Define the polling function
uint64_t start_polling(int pin);

#endif
```

Listing 3.3: GPIOManager.cs - P/Invoke section

```
// The function which calls the external function
private void Listen(object obj) {
    ThreadHelper th = (ThreadHelper) obj;
    while (true) {
        int pin = (int) th.Pin;
        // Call the function. See down.
        ulong cback = GPIOManager.start_polling(pin);
        th.Callback(4, (uint) 0, DateTime.Now);
    }
}

// External function represents a function on an external library, in this case the library is ↩
    the libIOSharp-c.so. The function naming and functions parameters are equal to the original ↩
    function, but taking into account that a ulong in C# is a uint64_t on C.
[DllImport("libIOSharp-c.so", CallingConvention = CallingConvention.StdCall)]
public static extern ulong start_polling(int gpio);
```

3.2.2.3 Final implementation

After defining and writing the library, and studding how to make calls to native functions it was time to code de final implementation making work together, the library and the IOSharp code. In this case, the program flow is shown on Figure 3.3. The Interrupt Port inherits from Input Port as Figure 3.3 shows.

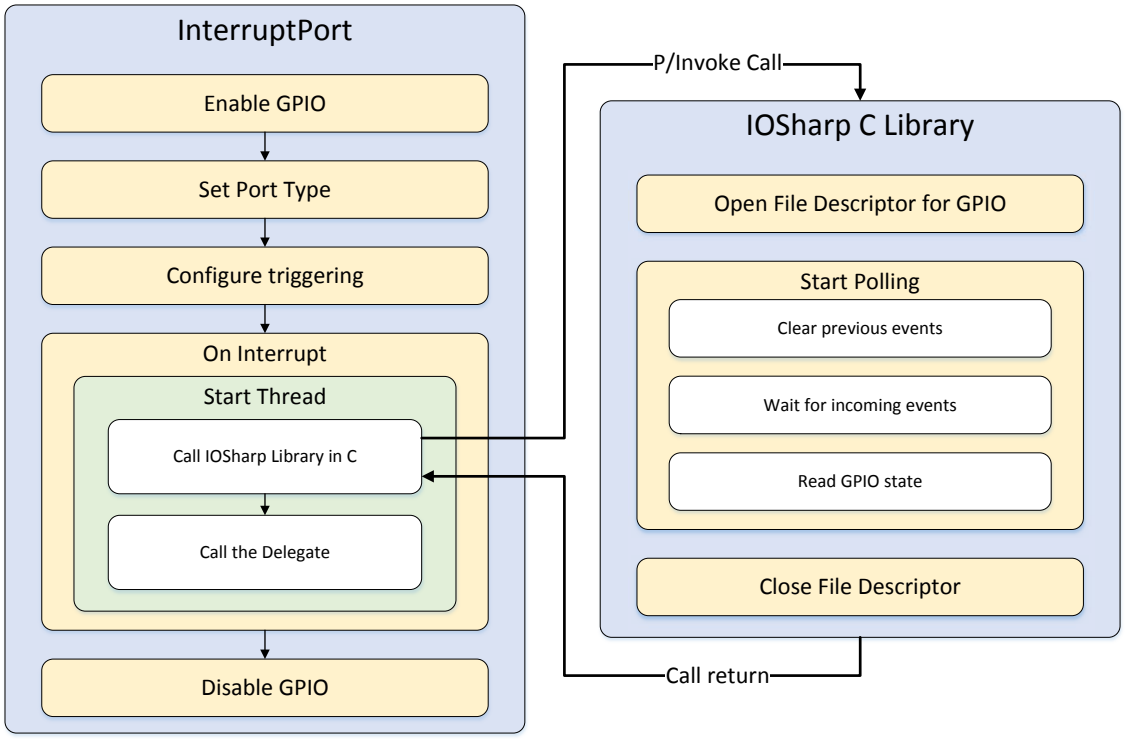

Figure 3.3: Representation of the Interrupt Port flow

The idea is to do the exact same steps like the other ports do, the GPIO enabling is done by the Port implementation which is the base class for Input Port, then the port type is set to be an Interrupt Port, after this is important to configure the triggering. NETMF boards usually supports four kinds of interruptions whereas Linux supports a few less, in the following table is shown which are supported. It is important to know that the Edges are the point where the port changes from one state to another one, and the Level is used for to trigger interruptions where there is a continuity on a certain state.

Trigger Type	Micro Framework	Linux
InterruptNone	X	X
InterruptEdgeLow	X	X
InterruptEdgeHigh	X	X
InterruptEdgeBoth	X	X
InterruptEdgeLevelHigh	X	
InterruptEdgeLevelLow	X	

Table 3.1: Interrupt Trigger Types

Like the GPIO the triggering is configured on the SYSFS on a file called edge located inside of the enabled GPIO folder, this file can be configured on three different ways without counting on None interruptions. In order to configure EdgeLow interruptions the word *falling* must be written, in case of EdgeHigh will be used *rising* and finally *both* is used for EdgeBoth.
Once the triggering is configured IOSharp will wait for a delegate to be configured by the

user in which will receive the interrupt events, once a delegate is passed to the OnInterrupt a thread will be assigned to the task of polling a GPIO using the library previously explained. This thread works as an infinite loop P/Invoking to C and waiting for the call return, when this occurs the delegated passed by the user is called and in this way the interruption event is passed straight to the program which uses IOSharp. Finally the interrupt port can be disabled as the other ones.

3.2.3 SPI

The SPI is one of the important features to be implemented on IOSharp. The SPI is a protocol used in embedded systems to communicate boards and components in a Master-Slave way. This protocol offers a full-duplex communication where the master and the slave can write and read at the same time on the channel. Normally this protocol accepts transmission frequencies in the range of 10 kHz to 100 MHz and in order to operate, the master configures its clock using a frequency less or equal to the maximum frequency supported by the slave which wants to communicate.
Both Netduino and RaspberryPi support the SPI communication protocol which makes easy the use of different modules. In case of this thesis, the SPI was required in order to make HomeSense work in the platform.
It was decided that the implementation of this component will be on a similar way as the it has been done in the interrupt port which is explained on the previous section.

3.2.3.1 Designing the SPI

3.2.3.2 Implementation

3.2.4 UART

The UART is a really simple protocol using asynchronous serial communication between two devices. Unlike the above functions which are not implemented on the standard .NET Framework the UART has a class on it.
After comparing and evaluating the differences on the Micro Framework and .NET Framework classes it was decided to use it on the IOSharp for two reasons, the first one is that any reuse of code is better rather than writing again the same feature, and considering that the communications between devices must be really well done, a class on the standard framework will be much more stable and tested than a code written from scratch. The other reason on choosing the existing class that although it is not exactly as the NETMF class it has all the required functions that are needed for deploying HomeSense.
Is important to emphasize that IOSharp in Linux runs using the Mono implementation of the standard Framework classes, and in this case the SerialPort class has some disadvantages on the Mono version, basically they do not support events such as `DataReceived` or `ErrorReceived` because the functions have not been implemented on its internal runtime, but as it was said, this feature is not used in HomeSense so it does not influence on the execution of the platform.

3.3 Port Mapping

In order to make IOSharp portable between hardware platforms a port mapping must be done. The idea is to map the underlying hardware to variables in order to be used on the upper-layer taking into account its GPIO ports, the SPI devices and its corresponding ports and the Serial Pins.

3.3.1 HardwareProvider

In fact, the original NETMF supports a hardware descriptor called HardwareProvider which do the mentioned things. The Raspberry Pi was the target platform for this project so a hardware descriptor was written, below is represented the pins of the Raspberry Pi, on the left the revision 1.0 and on the right the revision 2.0. In this pins are located the GPIO, two SPI devices, two UART ports and an I^2C bus.

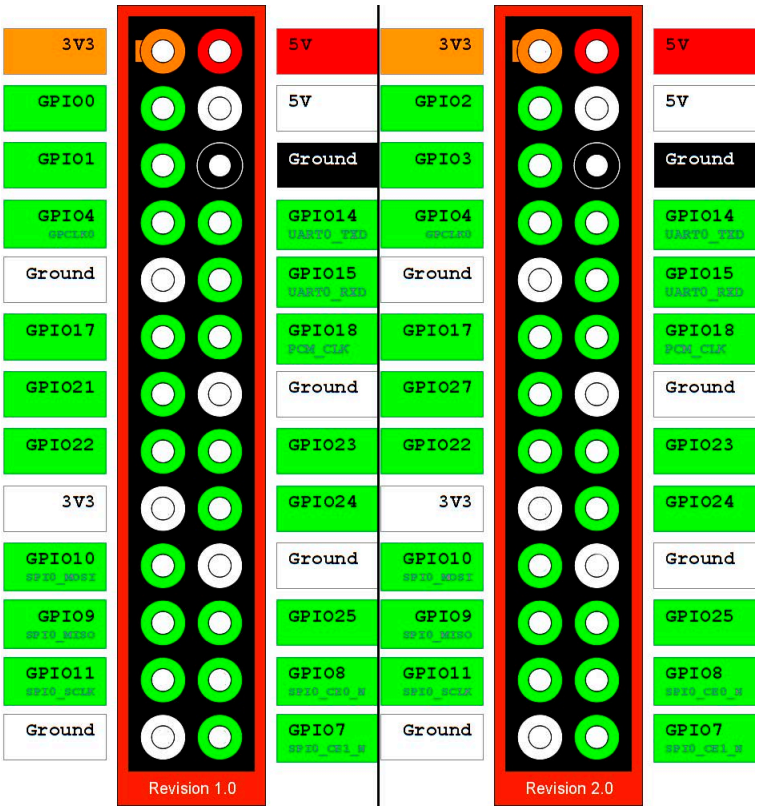

Figure 3.4: Raspberry Pi available ports

The mapping has been divided on two classes, the first one maps the Pins so it will contain the references to the platform pins, in case of this thesis, both versions are mapped. The second file is used to pass the specific pins to the program, for example it configures the pins for the SPI or the UART. It is interesting to mention that in Linux the SPI and UART are known as devices so they are located under `/dev/` directory. For the SPI the device is called spi0.0 and spi0.1 in case of the Raspberry Pi, this shows basically that

there are two SPI devices which shares the MISO, MOSI and SCLK pins it has two different chip selects so it can connect to two different slaves. In case of the UART it is also represented as a device and will have a name like `/dev/ttyAMA0` in case of Raspberry Pi.

This is not the unique case, for example the Netduino provides the same classes for the pins for the CPU.

CHAPTER 4. TESTS OF IOSHARP

4.1 Current IOSharp release

4.2 RFID and IOSharp

4.3 HomeSense over IOSharp

GLOSSARY

BCM2835 ARMv6 CPU mounted on the RaspberryPi. 12

File Descriptor file descriptor (FD) is an abstract indicator for accessing a file on POSIX systems. 13

IRQ Interrupt Request. 12, 13

PWM Pulse-width modulation. 9

SPI Serial Peripheral Interface. 9, 12, 18

UART Universal Asynchronous Receiver/Transmitter. 9

REFERENCES

[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: MicroFramwork a Linux

TITULACIÓ: Grau en Telemàtica

AUTOR: Gerard Solé i Castellví

DIRECTOR: Juan López Rúbio

DATA: December 13, 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