AlterNative: An executable to executable Cross-platform system for translating high-level to native applications

A. Albalá, J. López.

***Abstract*—Cross-platform applications are increasing their market due to the growing number of heterogeneous devices in the market. Developers are required to design and implement the software application in all the different targets that the product will be available. In this paper it is presented AlterNative, a cross-platform that tries to solve the problem of the fast development in multi-platform (cross-platform) applications.**

I. Introduction

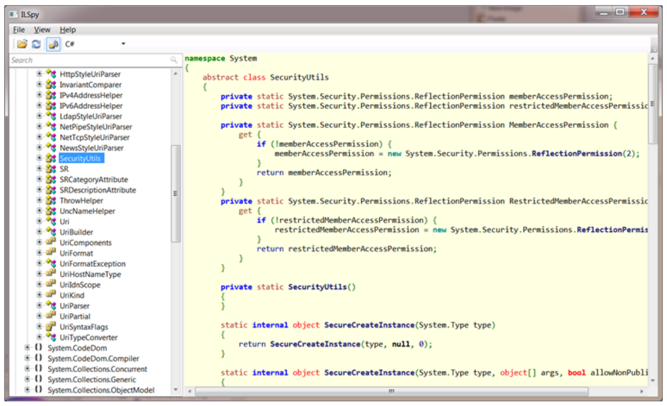
Nowadays all the service sectors like industrial, energy, healthcare, security among others are including new technological devices connected in the network. The number of interconnected devices is increasing exponentially in the last years. Therefore there are appearing huge variety of devices, with different operating systems and different programming languages. In other words the future situation can be described as an heterogeneous distributed system. The future of the devices tends to converge in a multiplatform environment composed by different devices and elements but with similar capabilites and possibilities, not only smartphones, smart TVs or tablets but also all kind of elements. In this work we present the AlterNative software as a cross-platform code translation tool focused on the performance of applications maintaining the maximum number of functionalities. AlterNative tries to solve the problem of multiplatform software avoiding the use of elements which can be perjudicial to the performance like virtual machines and aims to increase the efficiency of the code by using native languages.

II. State Of The Art

The rising market of Internet of Things devices promotes the development of software and tools for maintaining the interoperability between devices it is necessary to compile applications written in the specific language that accepts the processor. One option is to deploy Virtual Machines in all the devices benefiting of the portability of the code, but the performance it is not ideal. The State of the Art can be dividide in three main groups: Code translators, Decompilers and Research projects.

A. Code Translators

The translator is a tool for converting the code compiled for one system to the same application but compiled for other system. The translation can be done in the source code level (change text strings) or in other levels like byte code level



```
namespace System
{
    abstract class SecurityUtils
    {
        private static System.Security.Permissions.ReflectionPermission memberAccessPermission;
        private static System.Security.Permissions.ReflectionPermission restrictedMemberAccessPermissic

        private static System.Security.Permissions.ReflectionPermission MemberAccessPermission {
            get {
                if (!memberAccessPermission) {
                    memberAccessPermission = new System.Security.Permissions.ReflectionPermission(2);
                }
                return memberAccessPermission;
            }
        }
        private static System.Security.Permissions.ReflectionPermission RestrictedMemberAccessPermissic
            get {
                if (!restrictedMemberAccessPermission) {
                    restrictedMemberAccessPermission = new System.Security.Permissions.ReflectionPermis
                }
                return restrictedMemberAccessPermission;
            }
        }

        private static SecurityUtils()
        {
        }

        static internal object SecureCreateInstance(System.Type type)
        {
            return SecureCreateInstance(type, null, 0);
        }

        static internal object SecureCreateInstance(System.Type type, object[] args, bool allowNonPubli
        {
```

Figure 1. ILSpy decompiler

(change instructions). The translation provides the user with a tool which for developing the application in a comfortable language (high-level language) benefiting of all the high-level features and gaining time in the programming stage. When the code is finished, the translator is able to port this application developed in a high-level language to a low-level language programmed application, benefitting of the speed and performance of these languages. In this section the more relevant considered translators are explained.

B. Decompilers

For this work it is important to study different tools for extract code from applications or assemblies. This feature allows extracting information of the code and program structure from a file which is previously compiled. In this section will be studied different decompilers and assembly browsers. All of them have approximately the same features, the only difference is the languages supported (input and output languages).

ILSpy [1] is a .NET assembly browser and decompiler. It is open-source and its development started after the announcement that the free version of NET Reflector would cease to exist by end of February 2011.

ILSpy loads the assembly with the Mono.CECIL library (explained after in this chapter). This library inspects programs and libraries in CIL format. With the CECIL library the ILSPy extracts all the types definitions, methods, fields etc. Through an AST Builder, the program generates an Abstract Syntax Tree representing the source code in an abstract way.

An abstract syntax tree (AST) is a tree representation of the abstract syntactic structure of source code written in a

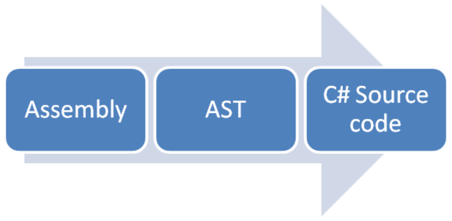


Figure 2. ILSPy process

programming language [2], [3]. Each node of the tree denotes a construct occurring in the source code. The syntax is 'abstract' in not representing every detail appearing in the real syntax. The final stage is to convert this Abstract Syntax Tree in text (source code). This program defines different templates, each one for each language. The supported output languages are:

- IL
- C#
- VB

There are several decompilers in the community that aim to the same objective as ILSpy does, the difference between them are the languages that they accept or the languages that they provide at the output.

In this work the ILSpy decompiler is used as the basis for the decompilation process of the assembly. ILSpy is in charge of get the assembly, extract the AST tree and get the source code from this tree. This process will allow this work to get an abstract representation of the source code.

C. *Multi-platform and cross-platform mobile tools*

There are different multi-platform tools that aim to make easy the development of mobile applications targeting the different operating systems of the market. Fore more information of the most popular cross-platform tools see [4]. A brief summary of the framework is provided.

1) PhoneGap [5]: PhoneGap 2.7.0 is an open-source mobile development tool developed by Adobe System Inc. under Apache 2.0 license. PhoneGap allows developers and companies to build free, commercial and open-source applications, and give them also the possibility to use any licenses combination. The development environment is cross-platform and permits the creation of applications for Android, Bada, BlackBerry, iOS, Symbian, webOS and Windows Phone OS's.

This framework is composed by a java library file and html files for the application views. PhoneGap provides an API for control the core of the mobile (sensors, camera, media ...) and uses standard JavaScript, HTML5 and CSS3 for bringing the views to the user. The use of all these standards allow PhoneGap to provide to the developers with a framework compatible with most of the operating systems in the mobile market. PhoneGap is compatible with: iOS, Android, Blacberry OS, WebOS, Windows Phone, Symbian and Bada.

2) Rhodes [6]: Rhodes 3.3.3 is a cross-platform mobile application tool developed by Motorola Solutions Inc. under Massachusetts Institute of Technology (MIT). It is developed to rapidly build native applications for all major mobile OS's (iOS, Android, BlackBerry, Windows Mobile/Phone and Symbian). The main goal of Rhodes is to provide a high level of productivity and portability in programming. It is an open source Ruby-based mobile development environment. Thanks to this environment, developers can create and maintain enterprise applications and data based on single source code across different mobile OS's. RhoMobile suite provides an IDE called RhoStudio which is an innovative solution dedicated to users that want to develop applications through a hosted IDE. This solution can be used across Linux, Mac, and Microsoft Windows OS's. Alternatively, RhoMobile offers the possibility to write applications with any other editor or IDE which supports HTML, HTML5, CSS, JavaScript and Ruby. Rhodes aims to bring agility [7] to the mobile development world.

3) DragonRad [8]: DragonRad 5.0 is a cross-platform mobile application development platform by Seregon Solutions Inc. and distributed Fig. 3. Complete schema of PhoneGap architecture and interfacing among components [5]. under a commercial license. It allows developers to design, manage and deploy mobile applications once and use it across iOS, Android, BlackBerry and Windows Mobile [3]. The tool focuses on database driven mobile enterprise applications with easy and wide range of databases support. It provides the D&D environment which help developers to save programming time and to create logics. DragonRad provides their own built IDE, that can be configured for different simulators like iOS, Android, BlackBerry, Windows Mobile etc.

4) MoSync [9]: MoSync 4.0 is an open source solution developed by a Swedish company targeted to mobile market. MoSync has fully fledged SDK which helps developers to build and package all type of mobile applications, such as simple, advanced and complex application that share the same code base. MoSync SDK is proving to be very powerful tool with many components tightly coupled together like Libraries, Runtimes, Device Profile Database and Compilers, and so on. It provides the full fledge Eclipse-based IDE and the use of standard C/C++. It also added the support with web-based language like HTML, HTML5, CSS and JavaScript. It provides well documented API's both in C/C++ and web-based. The idea involved to support multiple mobile OS's is different from other tools and also in very isolated way from other mobile operating code. Applications in MoSync are built to target a device profile by using GNU Compiler Collection (GCC) and pipetool. After writing the application, pipe-tool is used to compile the resources present in the application. Then GCC backend is called and path to target device profile passed to it. GCC uses it to produce MoSync intermediate language, which then fed in to pipe-tool. Then, pipe-tool behaves as the bridge between MoSync applications to target device profile. The profile database helps the application in ensuring that it has adapted correctly to the device. Runtimes are libraries which are bound to provide support related to all like regarding graphics, audio, communications, input, uniform interface to low level system API's and other device features.

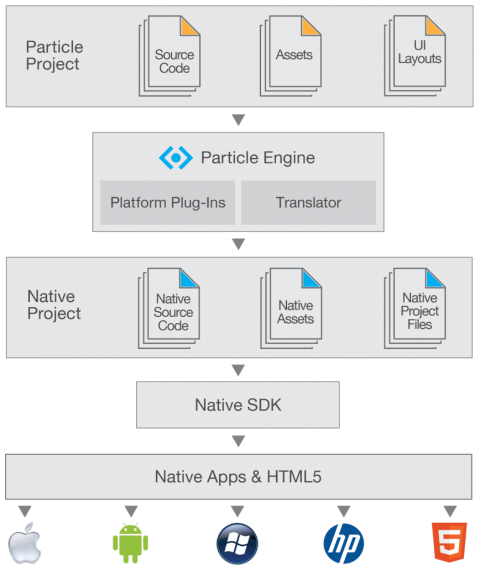


Figure 3. Particle Code engine

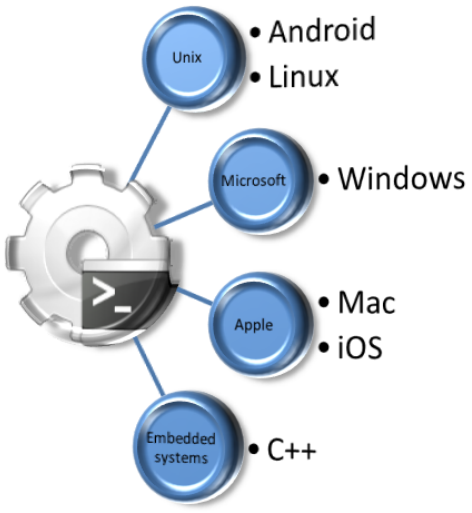


Figure 4. AlterNative concept

D. Particle code: end to end solution

The Particle Platform [10] is an open and extensible cross-platform end-to-end solution that enables developers to create native applications for smartphone platforms as well as HTML5 web applications. It allows developers to code in modern object-oriented languages like Java and ActionScript3, while working in the popular Eclipse IDE with a robust WYSIWYG front-end to manage User Interface design on various devices.

E. AlterNative contributions

In this chapter have been studied all the existing solutions targeting the same concept of AlterNative aims to. The most advanced research line is the Decompilers line, but these decompilers always target high-level languages and often do not output a legible code with a high-performance. AlterNative will take benefit of all the work done in this line and will use ILSpy decompiler for obtain the source code of an assemblied file in order to translate it to a low level language.

Nowadays, as the mobile and embedded market is increasing, the multiplatform tools for these markets are growing. However, these tools are focused on unify the application graphical user interface by means of standard web languages (i.e. HTML and CSS). The frameworks that aim to unify also the logical part are based on non-native code libraries. The performance is good enough but the can not target to other devices rather than modern mobile devices.

Concluding the AlterNative objective, this project aims to unify the best of all the multi-platform worlds for create and innovative tool able to translate specific high-level apps to multi-platform native applications reducing the development time and increasing the portability and performance of the application. A key point that makes AlterNative become a novel project is the possibility of not only run in graphics powered operating systems but also terminal applications with compatibility even in low power embedded systems.

III. Concept

The concept of AlterNative is to maximize the idea of Internet of Things by providing a tool for easy port applications from hmirigh-level languages (such as .NET) to native languages (such as C++). Most of the actual systems are C++ compatible, thus if the application is ported to this language, it can be executed in several platforms (i.e. smartphones, tablets, embedded systems, computers with different operating systems).

The philosophy of AlterNative is to provide the user with a system for taking the advantage of fast developing of high-level languages and also take the advantage of performance of low-level languages, and also exploit the possibility of run the native code in several systems, in other words, this philosophy is similar to the WORA (Write Once, Run Anywhere) slogan created by Sun Microsystems to illustrate the cross-platform benets of the Java Virtual Machine. The difference is focused on the final performance since AlterNative outputs in native language and does not depend on any virtual machine.

IV. Process

A. Overall progress

The process of the AlterNative software is divided in three parts: Decompilation, translation and recompilation.

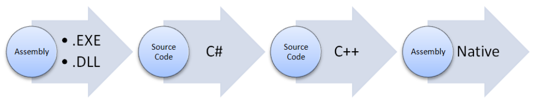


Figure 5. AlterNative process

First of all the assembly received is passed through a decompiler in order to extract the source code. In this case the code extracted is C#. The code is not extracted in text format, but in an AST (Abstract Syntax Tree) that is an abstract representation with nodes and hierarchies of the code.

Algorithm 1 ForEach example input C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections.Generic;

namespace ForEach {
    class Program
    {
        static void Main(string[] args)
        {
            List<float> myList = new List<float>();
            myList.Add(5.6f);
            myList.Add(5.7f);
            myList.Add(5.2f);
            myList.Add(5.9f);
            myList.Add(3.6f);
            myList.Add(52.6f);
            myList.Add(523.6f);

            foreach (float f in myList)
            {
                Console.WriteLine(f);
            }
        }
    }
}
```

This representation is organized in a tree from the top-level (Assembly) until de low-level (instructions, types and constants). The step of translation consists on apply different conversions to the AST in order to obtain a second AST representing the source code but in other language (in this case C++). The final step consists on compile the C++ code into a new assembly. This final assembly maintains the same funcionalities of the first assembly but takes benet of the native code performance. The effort of this software is not focused on the code translation, but it is focused in maintaining all the features and functionalities of the original code. The problem is that these features and functionalities often are directly related to the language, for instance a garbage collector, specic expressions or even language syntax. AlterNative is able to provide most of the features of the original code to the final code using external open-source libraries and propietary libraries in order to be fully compatible with high-level languages.

B. Example

In this section is explained an example of how a feature of a high level language is ported to a low level language. The language feature used is the *foreach* keyword. The source code languages are C# for the input and C++ for the output.

The algorithm 1 shows the input source code. Actually the input is a compiled executable file containing the described code. The example used is a ver easy program that creates a float list and initializes some values to a random number.

The resulting C++ code is listed in the algorithms 2 and 3 (Header file and Code file respectively). Notice that the low level output code is completelly legible from the developer point of view instead of the traditional low level conventions and namings. AlterNative always tries to maintain a high level syntaxis in order to facilitate the developers work.

For the success of this example it is necessary the following external work:

- **External C++ library List_T<T>:** A List library with the same syntaxis and class hierarchy of System.Collections.Generic.List<T> of .NET. In this case the List<T> .NET implementation extends from IEnumerable (an extended class of a classic iterator). In the C++ library this implementation has to follow the same hierarchy in order to take benefit of the IEnumerable and iterators methods, like the foreach feature.
- **External C++ library Console::WriteLine:** A Console library with the same syntaxis and class hierarchy of System.Console.WriteLine(float f). This library is in charge of mapping this method to the C++ standard output (std::cout).
- **External macro for FOREACH:** Supposing that the List_T<T> class declared is well templated and inherits from IEnumerable (also implemented in the library), the macro FOREACH call the methods that provide the iterators for managing the collection.

Algorithm 2 ForEach example C++ header output

```
#pragma once #include "System/System.h"
#include "System/Collections/Generic/List.h"
#include "System/Console.h"

using namespace System::Collections::Generic;
using namespace System;
namespace ForEach {
        class Program : public virtual Object  {
                public:
                        static void Main(String* args[]);
        };
}
```

Algorithm 3 ForEach example C++ code implementation output

```
#include "Program.h"
namespace ForEach {
        void Program::Main(String* args[]){
                List_T<float>* myList = new List_T<float>();
                myList->Add(5.6f);
                myList->Add(5.7f);
                myList->Add(5.2f);
                myList->Add(5.9f);
                myList->Add(3.6f);
                myList->Add(52.6f);
                myList->Add(523.6f);
                FOREACH(_F, myList){
                        float f = *_F;
                        Console::WriteLine(f);
                }
        }
}
```

V. Use cases

AlterNative can target to several use cases. In this section will be explained two of the more significative use cases: The performance improvement, and the cross-platform code generation tool.

A. Performance

In general the major advantage of this platform can be obtained when an application or library is very complex in a computational way. The more complex is the target, the more the benefit can be obtained. Suppose the following scenario:

You have an application for Android written in Java. The Android system is composed by a Java application running over a DalvikVM (Java Virtual Machine) which is running over a unix distribution. This application takes several images from the camera and processes each frame in order to change from YUV format (Android standard) to RGB format. After trying your application in Java environment you obtain a

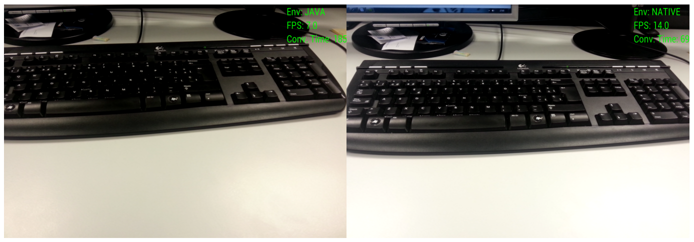


Figure 6. Java vs C++ performance. Device: Samsumg Galaxy S III Running Android 4.0

framerate of 7 frames per second. After this test you want to upgrade your application improving the performance. AlterNative platform allows you to take the Java code layer in charge of processing the image and convert it to C++ code (and also provides you the code with JNI naming ready to use in Android or Java environments) which allows you to run code below the DalvikVM. After that you repeat the test but with the processing layer running in C++ and you obtain 14 frames per second (two times faster than the first one).

B. Cross-Platform embedded systems

Other advantage of the standard low level languajes (like C++) is that a high percentaje of device porcessors can execute this code. We will explain this concept with an example.

Suppose we have a .NET MicroFramework board like NetDuino Mini familiy boards [11]. We use SPI and GPIO ports of this board for control some sensors and actuators. We want to port the application in a more powerful board like a Raspberry pi (wich is arm-linux architecture). A way to execute C# .NET code in linux environment is to use the Mono framework [12], but the actual mono implementation does not support MicroFramework applications. AlterNative allows us to translate the original C# MicroFramework code to C++ code, therefore, the output code can be compiled to an arm-linux target in order to execute the original code (with no further development) in other platform.

VI. Tests results

A. Tests suite

AlterNative project is provided by a testing suite following the model of regression test. Every test is in charge of check if a specific language feature is translated and compiled in the correct way. The test outputs information about the AlterNative process, the cross-compilation process, the compilation in the target platform process, the time performance difference between the original code and the translated code, the line differences between the original and the translated code and the comparison of the original and final program outputs.

For the evaluation of the system a CPU stress performance test will be executed. The tests selected are the fannkuch-redux benchmark [13] and the n-bodies benchmark [14]. The source codes used are listed below.

Algorithm 4 Fannkuch-redux test C# code

```
using System;
class FannkuchRedux
{
   public static int[] fannkuch(int n) {
      int[] p = new int[n], q = new int[n], s = new int[n];
       int sign = 1, maxflips = 0, sum = 0, m = n-1;
      for(int i=0; i<n; i++){ p[i] = i; q[i] = i; s[i] = i; }
            do {
         // Copy and flip.
         var q0 = p[0];// Cache 0th element.
         if (q0 != 0){
            for(int i=1; i<n; i++) q[i] = p[i];// Work on a copy.
            var flips = 1;
            do {
                var qq = q[q0];
               if (qq == 0){// ... until 0th element is 0.
                  sum += sign*flips;
                  if (flips > maxflips) maxflips = flips;// New maximum?
                  break;
               }
               q[q0] = q0;
               if (q0 >= 3){
                  int i = 1, j = q0 - 1, t;
                  do { t = q[i]; q[i] = q[j]; q[j] = t; i++; j--; } while (i < j);
               }
               q0 = qq; flips++;
            } while (true);
          }
         // Permute.
         if (sign == 1){
                          // Rotate 0<-1.
            var t = p[1]; p[1] = p[0]; p[0] = t; sign = -1;
         } else {
                          // Rotate 0<-1 and 0<-1<-2.
            var t = p[1]; p[1] = p[2]; p[2] = t; sign = 1;
            for(int i=2; i<n; i++){
               var sx = s[i];
               if (sx != 0){ s[i] = sx-1; break; }
               if (i == m) return new int[]{sum,maxflips};  // Out of permutations.
               s[i] = i;
               // Rotate 0<-...<-i+1.
               t = p[0]; for(int j=0; j<=i; j++){ p[j] = p[j+1]; } p[i+1] = t;
            }
         }
      } while (true);
   }

   static void Main(string[] args){
      int n = (args.Length > 0) ? Int32.Parse(args[0]) : 7;
      var pf = fannkuch(n);
      Console.Write("{0}\nPfannkuchen({1})␣=␣{2}\n", pf[0], n, pf[1]);
   }
}
```

B. Fannkuch-redux benchmark

We have compared three languages with three compilation targets. The languages are C# (original), C++ original (manually translated) and C++ alternative (obtained automatically). The three compilation targets are Windows 32 bits, Linux and Android, all of them with Release optimizations. The figure 7 shows the comparison of the languages in Linux and Android targets. In the Windows platform, C# .NET performs several optimizations that allows the executable to run with a reasonable performance. Even sometimes these optimizations makes the executable overcome to the C++ exectuable version.

The C++ original code obtains the better benchmark as expected. The AlterNative C++ generated code is near the original code. This behaviour is also expected because AlterNative does not implement stack or heap optimizations in the variables, now all the variables are allocated dynamically. The access to these variables are computationally expensive than in stack variables.

C. N-Bodies benchmark

The platforms and languages tested in this benchmark are the same that the tested in the Fannkuch-redux benchmark [14].

The results of the execution of this test are illustrated in the figure 7.

The behaviour of this test is similar to the Fannkuch-redux benchmark test. The difference between the original C++ code

Algorithm 5 N-Bodies algorithm

```
using System;
class NBody {
        public static void Main(String[] args) {
                int n = 50000000;
                NBodySystem bodies = new NBodySystem();
                Console.WriteLine("{0:f9}",bodies.Energy());
                for (int i=0; i<n; i++)
                        bodies.Advance(0.01);
                Console.WriteLine("{0:f9}",bodies.Energy());
                }
        }
}
class NBodySystem {
        private Body[] bodies;
        public NBodySystem() {
                bodies = new Body[]{
                        Body.Sun(),
                        Body.Jupiter(),
                        Body.Saturn(),
                        Body.Uranus(),
                        Body.Neptune()
                };
                double px = 0.0;
                double py = 0.0;
                double pz = 0.0;
                foreach (Body body in bodies) {
                        px += body.vx * body.mass;
                        py += body.vy * body.mass;
                        pz += body.vz * body.mass;
                }
                bodies[0].OffsetMomentum(px,py,pz);
        }
        public void Advance(double dt) {
                double dx, dy, dz, distance, mag;
                for (int i=0; i < bodies.Length; i++) {
                        for (int j=i+1; j < bodies.Length; j++) {
                                dx = bodies[i].x - bodies[j].x;
                                dy = bodies[i].y - bodies[j].y;
                                dz = bodies[i].z - bodies[j].z;
                                distance = Math.Sqrt(dx*dx + dy*dy + dz*dz);
                                mag = dt / (distance * distance * distance);
                                bodies[i].vx -= dx * bodies[j].mass * mag;
                                bodies[i].vy -= dy * bodies[j].mass * mag;
                                bodies[i].vz -= dz * bodies[j].mass * mag;
                                bodies[j].vx += dx * bodies[i].mass * mag;
                                bodies[j].vy += dy * bodies[i].mass * mag;
                                bodies[j].vz += dz * bodies[i].mass * mag;
                        }
                }
                foreach (Body body in bodies) {
                        body.x += dt * body.vx;
                        body.y += dt * body.vy;
                        body.z += dt * body.vz;
                }
        }
        public double Energy() {
                double dx, dy, dz, distance;
                double e = 0.0;
                for (int i=0; i < bodies.Length; i++) {
                        e += 0.5 * bodies[i].mass *( bodies[i].vx * bodies[i].vx
                        + bodies[i].vy * bodies[i].vy
                        + bodies[i].vz * bodies[i].vz );
                        for (int j=i+1; j < bodies.Length; j++) {
                                dx = bodies[i].x - bodies[j].x;
                                dy = bodies[i].y - bodies[j].y;
                                dz = bodies[i].z - bodies[j].z;
                                distance = Math.Sqrt(dx*dx + dy*dy + dz*dz);
                                e -= (bodies[i].mass * bodies[j].mass) / distance;
                        }
                }
                return e;
        }
}
class Body {
        const double PI = 3.141592653589793;
        const double SOLAR_MASS = 4 * PI * PI;
        const double DAYS_PER_YEAR = 365.24;
        public double x, y, z, vx, vy, vz, mass;
        public Body(){}

        internal static Body Jupiter() {
                Body p = new Body();
                p.x = 4.84143144246472090e+00;   p.y = -1.16032004402742839e+00;
                p.z = -1.03622044471123109e-01;
                p.vx = 1.66007664274403694e-03 * DAYS_PER_YEAR;
                p.vy = 7.69901118419740425e-03 * DAYS_PER_YEAR;
                p.vz = -6.90460016972063023e-05 * DAYS_PER_YEAR;
                p.mass = 9.54791938424326609e-04 * SOLAR_MASS;
                return p;
        }
        internal static Body Saturn() {
                //INITIALIZATION CODE ...
                return p;
        }
        internal static Body Uranus() {
                //INITIALIZATION CODE ...
                return p;
        }
        internal static Body Neptune() {
                //INITIALIZATION CODE ...
                return p;
        }
        internal static Body Sun() {
                //INITIALIZATION CODE ...
                return p;
        }
        internal Body OffsetMomentum(double px, double py, double pz) {
                vx = -px / SOLAR_MASS;   vy = -py / SOLAR_MASS;
                vz = -pz / SOLAR_MASS;
                return this;
        }
}
```

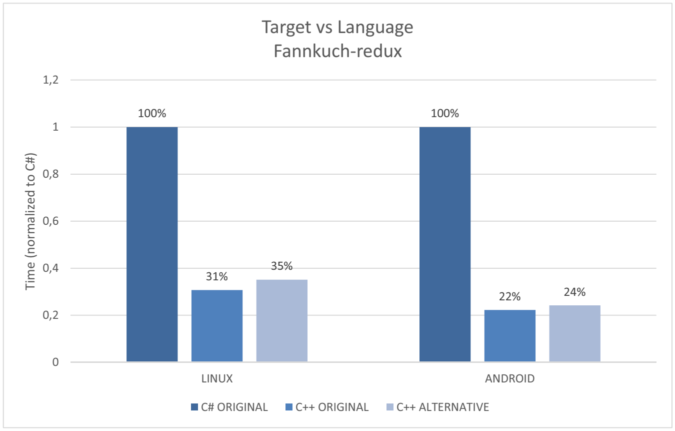


Figure 7. Time in function of the target platform for the Fannkuch-redux test

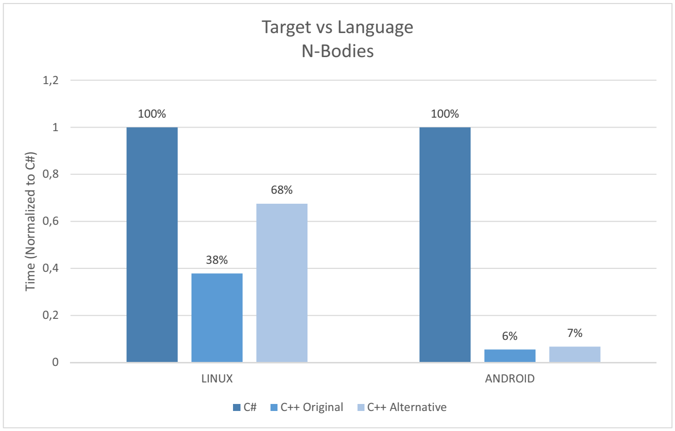


Figure 8. Time in function of the target platform for the N-Body test

and the AlterNative C++ code in this case is more accused than in the previous case. This result is due to this example access the memory a huge number of times. In stacked objects this is much faster than in dynamically allocated objects.

VII. Conclusions

Most of the cross-platform tools in the community are focused only on fast development for different devices (for instance games for smartphones). AlterNative a part of maintaining the objective of facilitate the easy and fast development, also is focused on the final performance of the application. Multiplatform systems based on virtual machines, or in HTML + CSS frameworks are useful for visual applications which do not require a high computational power, the main advantage of AlterNative software is that it is able to run code with native performance while the user is programming its code without taking into account the final application performance.

AlterNative platform also allows cross-platform development adapted to computers, mobiles and also embedded systems. This work is a novel tool that allows the developers to save time and develop powerful and cross-platform applications able to run in the major part of the devices

VIII. Future Work

The results obtained in this work are good in a cpu performance point of view. However, other important point in the high-level languages is the memory management. The memory management of high-level languages often depends in an external service called garbage collector. AlterNative implements a garbage collector in C++ using the popular Boehm gc library [15]. This implementation consists on track all the new objects and all the “destroyed” objects in order to manage all the memory allocations of the program. One future research line for this work is to automatize the garbage collection process in order to deallocate unused memory in the program. In the actual implementation, the system is prepared but never deallocates unused memory. A Binary Tree test [16] has been made in order to check that the memory management is not working properly. The figure 9 shows the results.

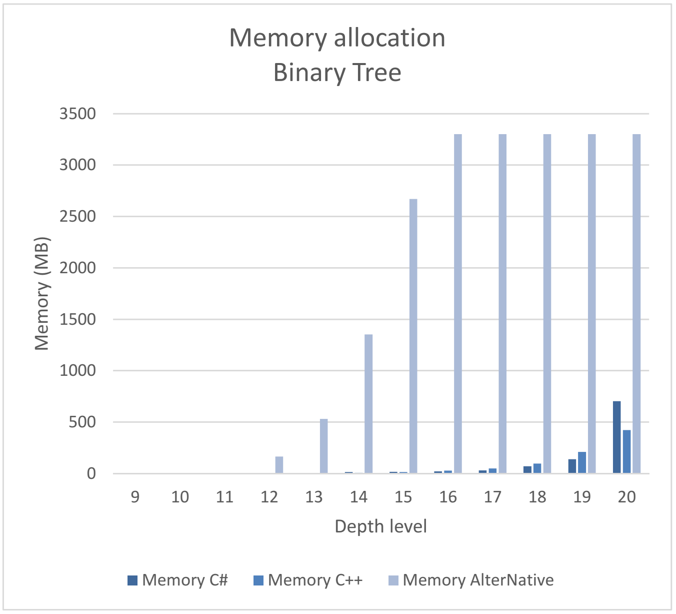


Figure 9. Memory allocation in Binary-tree test

Other future line in the project is to complete and expand the actual support library in order to cover all the basic C# language features.

Regarding the testing suite, there are several techniques for optimizing and reducing the cost of the regression test. An example of these optimizations can be found in [17], [18], [19].

References

[1] ILSpy homepage, http://ilspy.net/, May 2013
[2] A. V. Aho, R. Sethi, J. D. Ullman, “Compilers. Principles, Techniques, and Tools”, Bell Telephone Laboratories, United States of America, 1986
[3] D. Grune, H. E. Bal, C. J.H. Jacobs, K. G. Langendoen “Modern Compiler Design”, Vrije Universiteit, Amsterdam and Delf University, John Wiley & Sons, Ltd, England, 2000
[4] M. Palmieri, I. Singh, A. Cicchetti, “Comparison of Cross-Platform Mobile Development Tools”, Innovation, Design and Engineering Mälardalen University, Sweden, 2012
[5] B. Curtis. PhoneGap and the Enterprise. http://www.slideshare.net/drbac/phonegap-day-ibm-phonegap-andthe-enterprise, July 2011
[6] Motorola Solutions. http://www.motorola.com/Business/USEN/ Business+Product+and+Services/Software+and+Applications/RhoMobile+Suite, 2012
[7] Leckylao. Rhodes framework: Agile mobile web development. http://leckylao.com/2010/06/12/rhodes-framework-agile-mobileweb- development, June 2010
[8] DragonRad. http://dragonrad.com, 2012
[9] M. Kindborg. The wormhole javascript library. http://www.mosync.com/content/html5-javascript-wormhole, February 2012
[10] Particlecode, “Overview | Particle Code”, http://www.particlecode.com/features/overview/, May 2013
[11] Netduino mini specificatinos, http://www.netduino.com/netduinomini/specs.htm, May 2013
[12] Xamarin, “Mono, a technical whitepaper” http://www.mono-project.com/Mono,_a_technical_whitepaper May 2013
[13] Benchmarks game, “fannkuch-redux”, http://shootout.alioth.debian.org/u32/ performance.php?test=fannkuchredux, May 2013
[14] Benchmarks game, “nbody”, http://benchmarksgame.alioth.debian.org/u32q/ program.php?test=nbody, May 2013
[15] HPLabs, “A garbage collector for C and C++”, http://www.hpl.hp.com/personal/Hans_Boehm/gc/, May 2013
[16] Benchmarks game, “Binary trees”, http://benchmarksgame.alioth.debian.org/u32/ benchmark.php?test=binarytrees, May 2013
[17] P. Nagahawatte and H. Do, “The Effectiveness of Tegression Testing Techniques in Reducing the Occurrence of Residual Defects”, Department of Computer Science, North Dakota State University, Fargo, 2010
[18] Q. Gu, B.Tang, D. Chen, “Optimal Regression Testing based on Selective Coverage of Test Requirements”, Nanjing University, University of Aizu, 2010
[19] A. Kumar, S. Tiwari, K. K. Mishra and A.K. Misra, “Generation of Efficient Test Data using Path Selection Strategy with Elitist GA in Regression Testing”, Computer Science & Engineering Department, MNNIT, allahabad INDIA, 2010