ParroCha

low-level C# library that wraps libparrot

Autor: Juan Manuel Fernández Ribao
Tutor: Gregorio Robles Martínez

creative commons

Atribución 3.0 España

Tu eres libre de:

copiar, distribuir, comunicar y ejecutar públicamente la obra

hacer obras derivadas

Bajo las siguientes condiciones:

Atribución — Debes reconocer y citar la obra de la forma especificada por el autor o el licenciante.

With the understanding that:

Waiver — Any of the above conditions can be **waived** if you get permission from the copyright holder.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or **fair use** rights;
- The author's **moral** rights;
- Rights other persons may have either in the work itself or in how the work is used, such as **publicity** or privacy rights.

Notice — Al reutilizar o distribuir la obra, tienes que dejar bien claro los términos de la licencia de esta obra.

Esto es un resumen simple del texto legal (la licencia completa) disponible en los idiomas siguientes: Advertencia
Catalán Castellano Euskera Gallego

Índice de contenido

0 Agradecimientos....4
1 Introducción....5
2 Objetivos....17
3 Descripción de la solución informática....18
3.1 Entorno tecnológico....18
3.2 Descripción de la implementación....19
3.2.1 Mapeo de las APIs Embed y Extend....19
3.2.2 Acceso relativo a la biblioteca libparrot....20
3.3 Arquitectura de la implementación....21
3.3.1 Diagrama de clases....21
3.3.2 Diagrama de secuencia del constructor de Runtime....22
3.3.3 Diagrama de secuencia de cada método nativo....23
3.4 Hola Mundo Parrocha....23
4 Desarrollo en comunidad....26
4.1 Portales de desarrollo colaborativo....26
4.2 Licencia de la implementación....27
5 Trabajo futuro....28
5.1 Mapeo de macros y PMCs del núcleo de Parrot....28
5.2 Recubrimientos de los APIs de Embed de Mono y Hosting de .Net a través de PMCs nativos. 28
5.3 Implementación de Parrot sobre Mono....29
6 Conclusiones....30
Apéndice A) Bibliografía....32

0 Agradecimientos.

A mis padres por darme ánimos para realizar el máster y este proyecto en particular.

A mis compañeros de trabajo y a jefes del Centro de I+D de la Armada por facilitarme la difícil tarea de compatibilizar estudios y trabajo.

Al director de proyecto, por permitirme realizar el proyecto sobre una idea mía y ayudarme en la difícil tarea de definir el alcance de un proyecto a priori demasiado extenso para el tiempo en el que debía de realizarse.

1 Introducción.

Parrocha es envoltorio simple realizado para las plataformas de desarrollo basadas en el estándar CLI[8] de la biblioteca libparrot. Libparrot es una biblioteca de funciones implementada en C, que puede compilarse sobre varias plataformas y en la que se implementa la plataforma de desarrollo Parrot[43].

CLI[8] es el estándar sobre el que se basan .Net[1] y Mono[36]. Parrot[43] es una plataforma de desarrollo que nace con el objetivo de servir de base para la implementación oficial de PERL6[44]. El principal objetivo de este proyecto es implementar un mecanismo que permita utilizar programas escritos para Parrot[43] dentro de Mono[36] y .Net[1].

Parrocha permite utilizar programas de la plataforma Parrot[43] en plataformas CLI[8] a través de la aproximación más básica, utilizar una plataforma de desarrollo desde otra.

En el primer apartado de la introducción se exponen las características básicas de las plataformas de desarrollo. En los dos siguientes apartados se describen las plataformas CLI[8] y Parrot[43].

Los dos últimos apartados se centran en temas de implementación.

En el penúltimo apartado de la introducción se valora otra posible implementación para reutilizar programas de una plataforma de desarrollo en otra. Es aproximación es mucho más costosa, consiste en reimplementar los mecanismos más de bajo nivel de una en otra. En este apartado se verá el principal problema de implementar Parrot[43] sobre CLI[8], las continuaciones.

El último apartado es un breve estado del arte de los envoltorios para CLI[8]. Parrocha no es más que un envoltorio en C#[6] de una biblioteca escrita en C. Dentro de CLI[8] existe más de una opción para llamar a métodos de bibliotecas escritas en C, aquí veremos las distintas opciones.

1.1 Conceptos básicos de plataformas de desarrollo.

Una plataforma de desarrollo es un entorno común en el que cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encontraban relacionadas directamente a un sistema operativo. Actualmente es más común que se encuentren ligadas a una familia de lenguajes de programación y a una interfaz de programación de aplicaciones, API según sus siglas en inglés.

Con el objetivo de lograr que los programas desarrollados para una determinada plataforma de desarrollo sean portables entre varios dispositivos físicos y sistemas operativos nace el concepto de máquina virtual. Una máquina virtual es un programa nativo, es decir, ejecutable de una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, generado frecuentemente por un compilador de lenguaje de alto nivel.

Las máquinas virtuales, al ser en realidad programas, realizados para un sistema operativo y máquina real, pueden incorporar características difícilmente implementadas en *hardware*. Dentro de ese conjunto de características destaca la liberación automática de memoria, también conocida como recolección de basura.

La liberación automática de memoria no solamente permite al programador liberarse de esta tarea. También sirve para controlar el acceso a la memoria. Esto añade una nueva capa de seguridad al evitar los programas accedan a ciertas zonas de memoria, con mala intención del programador o simplemente por error.

Un hilo de procesamiento es un conjunto de instrucciones de bajo nivel que son ejecutadas en un determinado orden denominado flujo de ejecución. Los procesadores modernos incorporan mecanismos que permiten desordenar y ejecutar varias instrucciones máquina a la vez con el fin de mejorar rendimiento. Estas operaciones a nivel de procesador son transparentes para todo el software y no altera el resultado de la ejecución.

Los compiladores de lenguajes de alto nivel a código intermedio de máquina virtual traducen las sentencias de control de flujo del lenguaje de alto nivel a una o varias instrucciones de código intermedio.

Los entornos de desarrollo modernos también incorporan bibliotecas de funciones o métodos para trabajar con varios hilos de ejecución. En algunos casos, como Ada[2], la multitarea también puede expresarse a través del lenguaje.

PERL incorpora continuaciones como mecanismos de control de flujo. Una continuación es un mecanismo que permite interrumpir la ejecución de un método y volver a el dentro de un mismo hilo. Tanto la máquina virtual de Java[27] como .Net[1] CLR no soportan este mecanismo de control de flujo y MonoVM[36] lo implementa como una extensión sobre el estándar CLI[8].

Los diseñadores de .Net[1] CLR intentaron incorporar soporte para fibras en la segunda versión de la máquina virtual. El soporte para fibras

permitía implementar continuaciones en .Net[1]. Con la versión 2.0 final las fibras fueron descartadas, causaron numerosos fallos en el desarrollo.

Los diseñadores de máquinas virtuales, buscando la eficiencia, utilizan normalmente para guardar el contexto del flujo de ejecución el mecanismo de pila de llamadas del procesador. Máquinas virtuales que utilizan este mecanismo son .Net[1] CLR, MonoVM[36] y DaVinci[13] (Java 7.0[27]).

Parrot[43], como máquina diseñada especialmente para PERL6[44], utiliza continuaciones en lugar de la pila de llamadas del procesador.

Varios autores han intentado a través de distintas técnicas implementar continuaciones sobre las máquinas virtuales de Java[27] y .Net[1] CLR. El soporte para continuaciones ralentizaba la ejecución, obliga a llevar el contexto del programa fuera de la pila de ejecución.

El lenguaje Ruby[49], con el fin de permitir su implementación sobre .Net[1] y Java[27], ha eliminado las continuaciones de su sintaxis a partir de la versión 1.9.

1.2 **CLI.**

La infraestructura de lenguaje común, CLI[8], es un estándar de la organización ECMA para plataformas de desarrollo en el que se define una especificación de código binario, el entorno en cual se ejecuta y una biblioteca básica de clases que ha de proporcionar dicho entorno.

CLI[8], a diferencia de Java[27], fue diseñado desde un principio pensando en la utilización de compiladores en tiempo de ejecución, JIT, como base para la implementación de máquinas virtuales. La especificación CLI[8] ha sido pensada para aprovechar el rendimiento ofrecido por los compiladores JIT, por este motivo el contento de ejecución se basa en la pila de ejecución del procesador y la multitarea en el sistema de hilos del sistema operativo Windows[58] de Microsoft®, principal patrocinador de la plataforma.

CLI[8] es importante porque es el estándar que implementan .Net[1] y el proyecto Mono[36].

CLI[8] fue pensado como entorno de ejecución de lenguajes orientados a objetos estáticamente tipados. .Net[1] fue la primera implementación de CLI[8] y dentro de su SDK tiene únicamente compiladores para los lenguajes estáticamente tipados, C#[6] y Visual Basic.Net[55] son los más conocidos. Un lenguaje estáticamente tipado es aquel donde los tipos de los objetos y los métodos y campos de cada tipo se fijan en tiempo de compilación.

El lenguaje de programación más difundido en la plataforma CLI[8] es C#[6], fue específicamente desarrollado para CLI[8], también como estándar de ECMA. Mono[36] y .Net[1], con cada nueva versión de sus respectivos entornos, proporcionan un compilador de C#[6] con las últimas características añadidas al lenguaje.

La última característica que ha sido añadida a C#[6] es un soporte de tipado dinámico estático. Un objeto se tipa estáticamente como dinámico en tiempo de compilación. Este tipado dinámico se implementa a través del proyecto DLR[16].

El proyecto DLR[16] es un entorno de ejecución *opensource* desarrollado por Microsoft® para implementar lenguajes dinámicos sobre las máquinas virtuales .Net CLR[1] y MonoVM[36]. DLR[16] nace a partir del proyecto IronPython[23], cuyo objetivo era implementar Python[46] sobre .Net[1]. DLR[16] comenzó siendo parte de IronPython[23] y se disgregó de este cuando Microsoft® decide incorporar Ruby[49] a la lista de lenguajes dinámicos de .Net[1]. IronRuby[24] es el Ruby[49] de DLR[16].

El tipado dinámico con DLR[16] se soporta en un nivel superior al de la máquina virtual. No existen las continuaciones en DLR[16] porque este está implementado completamente sobre CLI[8], que carece de las mismas.

Como DLR[16] se implementa sobre CLI[8], que no incorpora continuaciones, tampoco incorpora soporte para este mecanismo. Este es el principal por el que DLR[16] no sirve para implementar PERL6[44].

IronScheme[25], un proyecto libre para implementar Scheme sobre DLR[16], solamente ha logrado implementar parcialmente las continuaciones de la especificación de Scheme.

CLI[8] ya antes de existir DLR[16] incorporaba el espacio de nombres System.Reflection.Emit. Este espacio de nombres permite generar código en tiempo de ejecución. Para mejorar el rendimiento de DLR[16] se añadió a este espacio de nombres la clase DynamicMethod en la versión 3.5 del .Net Framework[1]. DynamicMethod genera métodos en tiempo de ejecución sin la necesidad de tener que generar también un tipo nuevo para contenerlos.

MonoVM[36] como CLR implementan CLI[8]. Cuando sale DLR[16] Microsoft® informa a los desarrolladores de Mono[36] que deben de cambiar la forma en la que implementan los tipos genéricos que aparecen con la versión 2.0 del .Net Framework[1]. Microsoft® ayuda a la comunidad de Mono[36] para implementar la optimización interna para tipos genéricos de CLR en su entorno de ejecución. Sin esta mejora ya se podía ejecutar DLR[16] en Mono[36], aunque de forma mucho más lenta.

LSL.Net[34] es el lenguaje sobre el que se implementa el juego en red SecondLife. LSL.Net[34] se encuentra implementado para funcionar sobre plataformas CLI[8]. Debido a las necesidades propias de los servidores de SecondLife dicho lenguaje incorpora soporte explícito para continuaciones basado en la reescritura del lenguaje intermedio de la plataforma CLI[8].

Dentro de CLI[8] no hay ningún mecanismo estándar para utilizar bibliotecas binarias, ni tampoco para ser empotrado en aplicaciones nativas. Mono[36] implementa el mismo mecanismo que .Net[1] para acceder a bibliotecas nativas. Además con Mono[36] también hay una extensión para sistemas operativos no Microsoft®.

.Net CLR[1] y MonoVM[36] tienen APIs para permitir embeber dichas máquinas virtuales en aplicaciones nativas. Dichas APIs dependen de la máquina virtual.

.Net CLR[1] también incorpora mecanismos para acceder al modelo de objetos COM[10] del sistema operativo Windows[58]. MonoVM[36] está comenzando a soportar ese modelo del objetos tanto en Windows[58] como en otros sistemas operativos.

1.3 **Parrot.**

Parrot[43] es una máquina virtual diseñada especialmente para lenguajes interpretados, principalmente destinada a PERL6[44]. PERL6[44], como ya se ha comentado en los apartados anteriores de esta introducción incorpora continuaciones dentro de su especificación.

Las necesidad de portabilidad y la necesidad de soportar continuaciones y otras características de los lenguajes dinámicos son los dos principales motivos por el que los diseñadores de Parrot[43] han basado la máquina virtual en un intérprete.

Frente a las máquinas virtuales basadas en la pila de ejecución, Parrot[43] guarda el estado de ejecución en registros, como los procesadores reales. Dichos registros guardan el estado de la continuación al igual que la pila guarda el estado de ejecución en las máquinas virtuales basadas en pilas de ejecución.

Parrot[43] está presente en diversas plataformas porque está escrita en C y soportada para los compiladores GCC[18] y Visual C++[56]. El núcleo de Parrot[43] es una biblioteca dinámica que puede ser embebida en otras aplicaciones, como el servidor web Apache[4].

Parrot[43] ofrece tres APIs de C para interactuar con el intérprete:.

- El API Core permite interactuar con el intérprete a bajo nivel, está diseñado para ser utilizado por los programadores del núcleo del intérprete. A través de este API se puede acceder a los registros de estado internos de la máquina virtual.

- El API Extend sirve para implementar rutinas en C para ser utilizadas desde Parrot[43]. Incorpora métodos en C a través de los cuales podemos manipular crear y manipular los objetos y tipos básicos de Parrot[43].

- El API Embed, es el API que se ha de utilizar para empotrar la máquina virtual de Parrot[43] en otros programas. En los ejemplos de la

implementación se utiliza con los métodos del API Extend.

Como complemento a estas APIs Parrot[43] tiene un precompilador de C (pmc2c.pl) para generar tipos de objecto Parrot[43] o PMCs. Los PMCs pueden implementarse tanto en un lenguaje de alto nivel, implementado sobre la máquina virtual, como en C. Hay un grupo de PMCs que forman parte del núcleo de Parrot[43]. Todos los PMCs del núcleo están escrito en C por temas de eficiencia.

1.4 Interoperatividad entre máquinas virtuales.

La reutilización de código es uno de los principales temas de estudio de la ingeniería del software, especialmente importante en el caso concreto del software libre. La reutilización de código es uno de los temas que Eric Raymond, en su conocido ensayo La Catedral y el Bazar, destaca como uno de los principales pilares del éxito del sofware libre.

Existen varios mecanismos para lograr reutilizar funcionalidad escrita para una máquina virtual en otra:

- Mecanismos de llamadas a procedimientos remotos como CORBA[11] o servicios Web. Es el mecanismo de más alto nivel, solamente se requiere que las dos máquinas virtuales con las que se pretende trabajar sean capaces de comunicarse a través protocolos de red estándar. Sin duda este es el mecanismo más estándar con el protocolo SOAP[51] como máximo exponente.

- Comunicación a través de mecanismos IPC y del sistema de ficheros. S.O. Además del clásico mecanismo del tuberías de Unix, en el caso de Linux®[32], el proyecto FUSE[17] permite escribir sistemas de ficheros en espacio de usuario en máquinas virtuales. Una máquina virtual realice operaciones sobre un sistema de ficheros FUSE[17] puede estar comunicándose transparentemente con otra máquina virtual.

- Comunicación a través de bases de datos. Aunque no es aconsejable por

temas de rendimiento, bases de datos entre las que están Oracle®[41], SQL Server®[53] y PostgreSQL[45] permiten utilizar una máquinas virtuales de propósito general para la escritura de procedimientos almacenados. En los casos en los que desde una máquina virtual se haga una consulta a una base de datos que provoque la ejecución de uno de estos procedimientos almacenados ya tendríamos un caso de comunicación entre dos máquinas virtuales.

- Puentes entre máquinas virtuales, categoría a la pertenecería Parro*cha*. En esta categoría tenemos JNI4Net[29], que permite utilizar la máquina virtual de Java[27] desde .Net[1] a través del API JNI. En el S.O. Windows[58] el modelo de componentes COM[10] también permite este tipo de comunicación.

- Traductores de lenguajes de alto nivel que por ejemplo reescriban automáticamente código Java[27] en C#[6].

- Compiladores de lenguajes de alto nivel de una plataforma para otra, como Ja.Net SE[26], que compila código Java[27] para .Net[1].

- Extendiendo una máquina virtual para que sea capaz de entender el código de otra. El proyecto Parakeet[42] extiende la máquina virtual de Java[27] Jikes RVM[28] para que sea capaz de interpretar el código intermedio de Parrot[43].

- Traductores de lenguajes de bajo nivel, que transforman el código intermedio de una máquina virtual en el de otra. Parrot[43] incorpora el el compilador net2pbc que transforma código intermedio de CIL en binario de Parrot[43], este compilador no está muy completo, no es capaz de traducir todo la biblioteca básica de Mono[36]. El proyecto IKVM.Net[22] traduce *bytecode* de Java[27] en archivos de .Net[1].

- Máquinas virtuales escritas directamente sobre otra máquina virtual, como IKVM.Net[22]. IKVM.Net[22] como máquina virtual lo único que hace es utilizar su traductor de código intermedio en tiempo de ejecución.

LLVM[33], cuyas siglas en inglés significan máquina virtual de bajo nivel, a pesar de su nombre, tiene poco que ver con las máquinas virtuales tradicionales. LLVM[33] es en realidad una infraestructura para compiladores que máquinas virtuales como la de Mono[36] utilizan para mejorar su rendimiento.

LLVM[33] es un proyecto que ha adquirido bastante importancia por el patrocinio de Apple y Google. LLVM[33] es la base de código del compilador de C/C++ y Objetive-C Clang[7]. Clang[7] es el compilador que Apple tiene pensado utilizar en las futuras versiones de MacOSX®[35] y que ya usa como base del lenguaje para procesadores paralelos OpenCL[38]. Google también está utilizando LLVM[33] para optimizar Python[46] con su proyecto Unlanden Swallow[54], como soporte para Youtube.

Las máquinas virtuales se pueden clasificar en máquinas de pila, como Java[27] o .Net[1] y máquinas de registros, como Parrot[43] o Dalvik[12], la máquina virtual de Android[3]. Las máquinas virtuales de pila tienen un conjunto de instrucciones con 0 operandos para realizar operaciones sobre los valores en el tope de una pila. El conjunto de operaciones de una máquina de registros suele tener dos o tres operandos y aplica estas operaciones sobre múltiples registros de memoria.

Las operaciones de una máquina virtual de pila se traducen sin apenas merma de desempeño en una máquina de registros. Este es el caso de los JIT que traducen los bytecodes de Java[27] o el código intermedio CIL al código binario de los procesadores físicos como los de la familia X86. Un ejemplo de traducción entre código de una máquina virtual de pila en código de máquina virtual de registros lo tenemos en el compilador dx de Android[3] que transforma bytecode de Java[27] en código .dex, código de la máquina virtual Dalvik[12].

La traducción del código de una máquina virtual de registros, como Parrot[43] o Dalvik[12], en código de máquina virtual de registros acaba dando como resultado programas bastante lentos. El autor del proyecto Parakeet[42] explica en su tesis doctoral que prefirió extender la máquina virtual Jikes RVM[28] porque la transformación de código binario de Parrot[43] en *bytecodes* de Java[27] era mucho más complicada que utilizar el código intermedio interno de la máquina Jikes RVM[28], con operaciones sobre registros. Este problema también se lo encontraron los programadores del frontend de Java[27] del compilador GCC[18], que tampoco lograron realizar un compilador que a partir del lenguajes interno del GCC[18], basado en operaciones sobre registros, generara *bytecode* Java[27].

IKVM.Net[22] es probablemente el proyecto de software libre más conocido escrito con el fin de poder reutilizar software de una máquina virtual, en este caso Java[27], en otra, .Net[1] o Mono[36]. IKVM.Net[22] se distribuye con Mono[36] y es capaz de ejecutar proyectos tan complejos como Eclipse sobre Mono[36] y .Net[1]. Ejecutar un programa Java[27] sobre IKVM.Net[22] es mucho más lento que sobre una máquina virtual convencional. IKVM.Net[22] no solamente debe encargarse de traducir código, también se debe de encargar de tratar las diferencias entre los modelos de objetos, excepciones y multitarea de Java[27] y CLI[8].

1.5 Envoltorios de bibliotecas nativas con CLI.

La implementación del proyecto Parro*cha* es un envoltorio escrito en C#[6] de la biblioteca donde se implementa la máquina virtual de Parrot[43].

.Net[1], a diferencia de Java[27], nace como una plataforma para la reutilización de código ya existente con mayor productividad. El objetivo principal de Microsoft® no era la portabilidad del código. El S.O. Windows[58] es el S.O. más extendido y a Microsoft® no le interesa que las aplicaciones desarrolladas sobre su plataforma puedan se portables a otras.

Microsoft® cuando desarrolla .Net[1] ya tenía desarrollados mucho código para Windows[58] que no podía tirar. .Net[1] tenia que ser capaz de reutilizar todo ese código y además mejorar la productividad del programador. Cuando Microsoft® apadrina los estándares ECMA-CLI e ISO-CLI no cubre en

ellos los aspectos de acceso al código nativo de plataforma. La manera en la que .Net[1] accede al código nativo es totalmente dependiente de Windows[58].

Los desarrolladores de Mono[36] copian el acceso a la plataforma de .Net[1] y lo extienden a través de un archivo de configuración. El archivo de configuración permite la portabilidad del código intermedio entre plataformas, todo lo que varié entre plataformas queda reflejado en dicho archivo.

A continuación vamos a ver una serie de envoltorios libres para .Net[1], y las técnicas utilizadas en cada uno.

1.5.1 **GTK#.**

GTK#[21] es un envoltorio escrito principalmente en C#[6] de varias bibliotecas del proyecto GNOME. La principal biblioteca que envuelve es GTK+[20], y de esta recibe su nombre.

Parte de GTK#[21] se genera automáticamente a partir de las herramientas de documentación de GNOME siguiendo el modelo de objetos de este.

1.5.2 **Qyoto y Kimono.**

Son envoltorios de las bibliotecas Qt[47] y KDE[30] respectivamente. Como las bibliotecas de KDE[30] y Qt[47] están escritos en C++ se hace necesario generar primero un envoltorio intermedio en C debido a que C++ no tiene un ABI estándar al que una implementación de CLI[8] pueda llamar directamente.

Se encuentran desarrollados con las herramienta de generación de envoltorios del proyecto KDE[30] SMOKE.

1.5.3 **OpenTK.**

Es un envoltorio de varios APIs C del grupo Khronos[31]. El API más famosa de las especificadas por este grupo es OpenGL[39]. API multiplataforma para el desarrollo de gráficos en 3D con aceleración gráfica.

Como Khronos[31] publica las especificaciones en XML, la mayor parte de la implementación se genera automáticamente a partir de dichas especificaciones.

Para acceder a las extensiones de OpenGL[39] es necesario el uso de punteros a función, parte del API utiliza un mecanismo distinto, basado en métodos delegados, al utilizado normalmente en .Net[1] y Mono[36].

Un delegado es un mecanismo de CLI[8] para representar punteros a función y sobre el que se desarrolla el modelo de eventos de la plataforma.

El mecanismo de carga de métodos a través de delegados es mucho más flexible que el normal, basado en anotaciones. Este mecanismo permite utilizar direcciones absolutas para la carga de bibliotecas, además de un mayor control de cuando accedidas estas.

La implementación final del envoltorio de Parrot[43] también se basarán en delegados para permitir obtener dinámicamente el acceso a la biblioteca.

1.5.4 **SlimDx.**

SlimDx[50] es un envoltorio del API para el desarrollo de aplicaciones 3D de Windows[58] DirectX[15]. Se encuentra desarrollado a mano en C++/CLI, una extensión estándar de C++ para trabajar con máquinas virtuales de CLI[8].

Debido a que la única implementación de CLI[8] sobre la que se puede utilizar los ensamblados mixtos generados desde C++/CLI es .Net CLR[1] este envoltorio solamente se puede utilizar con .Net[1].

2 **Objetivos**

El objetivo principal del este proyecto es la construcción de un envoltorio en C#[6] de las APIs Embed y Extend descritas en el apartado 1.3 de esta memoria que permita utilizar la máquina virtual de Parrot[43] desde Mono[36] o .Net[1] a partir del nombre del directorio donde se encuentra instalado esta.

Las API Embed es la que permite empotrar Parrot[43] en otros programas. La API Extend está desarrollada principalmente para desarrollar extensiones de la máquina virtual Parrot[43] en lenguajes compilados.

La API Embed se apoya en los métodos del API Extend para acceder al valor de las propiedades de los objetos de PMC y también para poder invocar métodos.

Un objetivo importante del proyecto es que el envoltorio sea capaz cargar la biblioteca que implementa Parrot[43] a partir de la dirección donde esta instalada la máquina virtual que se quiera utilizar. Deberá tenerse en cuenta que en un mismo sistema operativo puede haber varias máquinas Parrot[43] instaladas a la vez.

3 Descripción de la solución informática.

Parro*cha* es un envoltorio de dos APIs de C. Es por esto que a pesar de estar escrito en un lenguaje orientado a objetos, C#[6], tiene un modelo de objetos muy simple.

Lo verdaderamente importante del proyecto es la solución tecnológica adoptada para su desarrollo.

Este apartado se estructura en cuatro apartados:

1. Entorno tecnológico: tecnologías usadas en el proyecto.
2. Descripción de la implementación: explicación detallada y técnica de las principales decisiones tecnológicas
3. Arquitectura de la implementación: sencillo diagrama de clases de la implementación y principales diagramas de secuencia.
4. Hola Mundo Parrocha: simple implementación basada en el ejemplo Cotorra en C de la distribución oficial de Parrot[43].

3.1 ***Entorno tecnológico.***

El proyecto se ha desarrollado en el sistema operativo Windows Vista[58], en el lenguaje C#[6] y con el entorno de desarrollo Visual Studio 2008 Express Edition[57].

Se han elegido C#[6] y Visual Studio[57] por ser el lenguaje y la herramienta de desarrollo más estándar para el desarrollo en la plataforma .Net[1]. La versión 2008 Express Edition de Visual Studio[57] además es una herramienta gratuita.

Se ha comprobado también que los ejecutables de las pruebas internas del proyecto funcionasen también sobre la versión 2.6.3. de Mono[36] y que el proyecto pudiera ser abierto por el entorno de desarrollo MonoDevelop 2.2[37] sobre Windows Vista[58].

Inicialmente el proyecto fue desarrollado para la versión 2.2.0. de Parrot[43] para Windows[58], posteriormente se actualizó a la versión 2.3.0.

3.2 *Descripción de la implementación.*

En el proyecto Parro*cha* el estudio de las diferentes soluciones tecnológicas es la parte más importante y que más trabajo ha llevado.

La implementación del proyecto costa de dos partes, el mapeo de las funciones C a C#[6] y el mecanismo de llamada desde el mapeo en C#[6] a la biblioteca que implementa Parrot[43].

En los dos siguientes apartados se describen los motivos que ha llevado a la solución tecnológica seleccionada en cada uno de ellos.

3.2.1 Mapeo de las APIs Embed y Extend.

Según se puede leer en el apartado 1.3. la biblioteca libparrot exporta tres APIs.

La API Core es el API de más bajo nivel, permite acceder a los registros internos de la máquina virtual a través de métodos que devuelven punteros a estructuras. Aunque C#[6] permite mapear a través de tipos por valor, que son las estructuras propias del lenguajes, las estructuras de C se ha optado por no realizar el mapeo de este API. El mapeo de estructuras de C en C#[6] es complicado y el resultado no resultaría útil para el objetivo del proyecto. La utilización del Parrot[43] desde una máquina virtual de CLI[8].

La API Embed es la diseñada por los creadores de Parrot[43] para empotrar la máquina virtual en aplicaciones. El API Embed sustituye los punteros a estructuras de C por punteros del tipo void*. Los punteros del tipo void* en C#[6] se mapean al tipo de datos IntPtr. La mapeo del API Embed, por este motivo, es muy sencillo.

El API Extend es el que deben de utilizar los programadores de

extensiones de la máquina virtual en C. Se utiliza también en conjunto del API Embed para permitir leer y escribir el estado de los objetos de Parrot[43].

El ancho del tipo de dato IntPtr depende de la máquina real sobre la está escrita la virtual y coincide con el tamaño de los punteros de la misma. Este tamaño es de 32 bits en arquitecturas de 32 bits y de 64 en arquitecturas de 64 bits.

3.2.2 Acceso relativo a la biblioteca libparrot.

En el apartado 1.5.3. se menciona el método que utiliza la biblioteca OpenTK[40] para cargar los métodos de las extensiones del API OpenGL[39].

Las extensiones de OpenGL[39] no se encuentran implementadas en todos los drivers. Por este motivo los autores de OpenTK[40] han utilizado un método basado en las bibliotecas de enlace dinámico de cada sistema operativo y delegados.

Un delegado de C#[6] es un puntero a una función. El valor de ese puntero en el caso de OpenTK[40] se obtiene a través de las diferentes bibliotecas de enlace dinámico de casa sistema operativo.

A través del API reflection de CLI[8], OpenTK[40], consulta el nombre de cada delegado y con ello lee los métodos de la biblioteca que contiene al driver de OpenGL[39].

Cada sistema operativo tiene su API para leer dinámicamente métodos de una biblioteca. Este método de mapeo por tanto debe de identificar primero el sistema operativo y luego utilizar al API correspondiente.

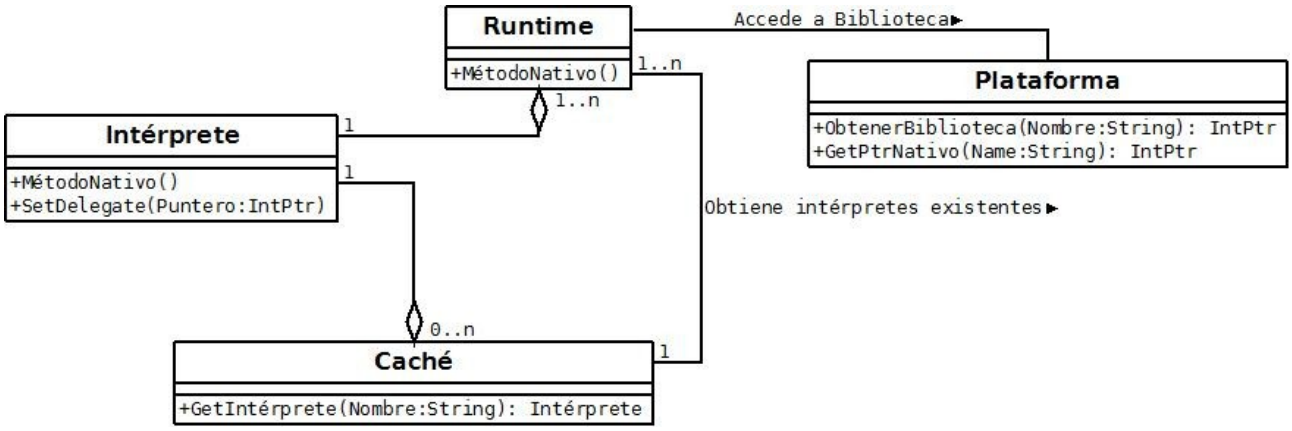
Los métodos de la biblioteca libparrot no varían de un sistema operativo a otro. Lo que si puede variar es la localización de la biblioteca en cada instalación de Parrot[43].

La lectura dinámica de bibliotecas y métodos que utiliza OpenTk[40] también sirve para leer la biblioteca cuando esta no se encuentra en el camino de lectura prefijado del sistema operativo.

Parro*cha* utiliza el mismo método que OpenTK[40] para acceder a la biblioteca nativa. Este método si permite cumplir con el objetivo de que el usuario pueda instalar la máquina virtual Parrot[43] en la localización que considere conveniente.

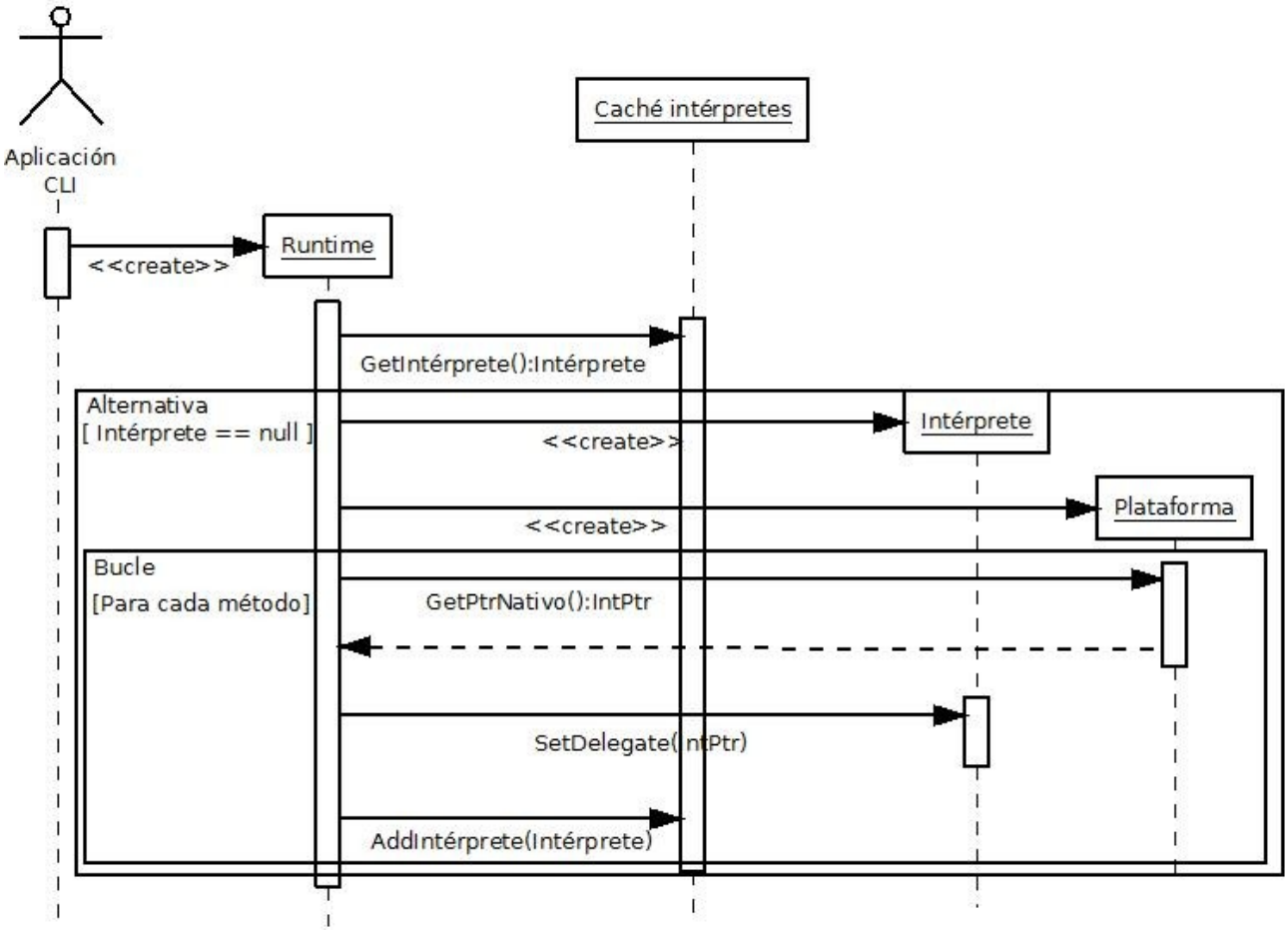
3.3 ***Arquitectura de la implementación.***

3.3.1 Diagrama de clases.



- Runtime: Clase principal de la aplicación, carga los métodos nativos en un objeto intérprete y hace de fachada del mismo.
- Intérprete: Clase que contiene los punteros a función nativos en campos de tipo delegado.
- Caché: Tabla *hash* con intérpretes previamente cargados para su reutilización en nuevos Runtimes.
- Plataforma: Clase que contiene los métodos nativos de cada sistema operativos soportado para acceder a la bibliotecas de la plataforma.

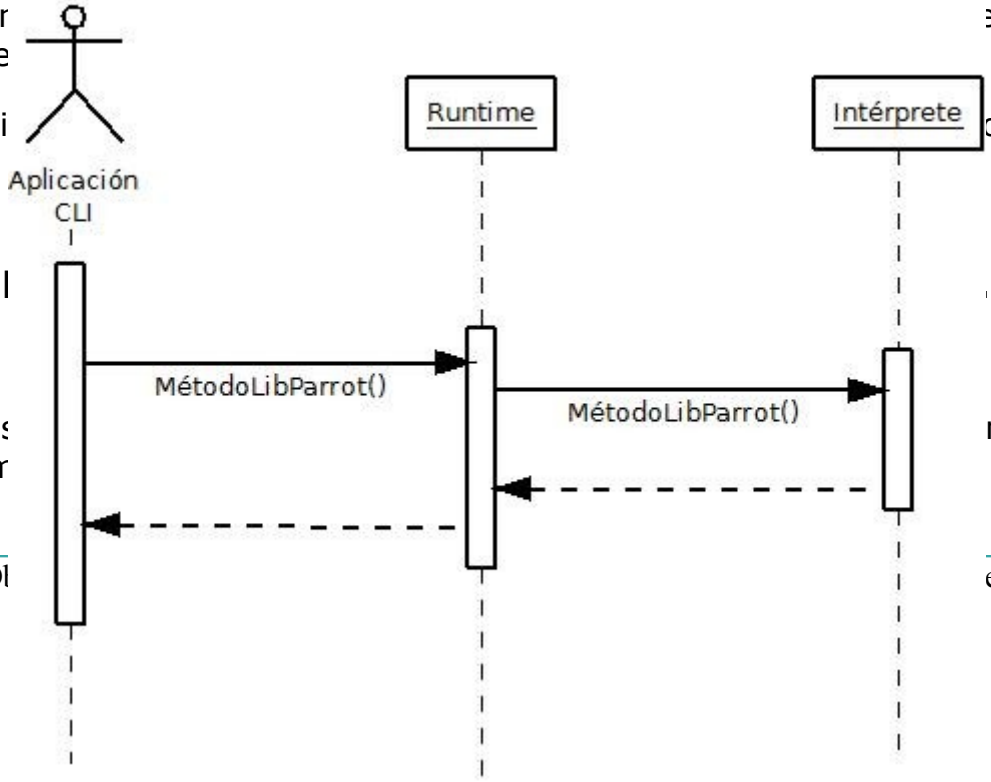
3.3.2 Diagrama de secuencia del constructor de Runtime.



Este es el método más importante de la aplicación. En él a partir del nombre de una biblioteca se comprueba si ha sido previamente utilizada a través de ur ete con el que mape

Graci ot.

3.3.3 I



Todos ndo un runtime com

Se ha contemplado el caso de que un método nativo pueda no existir en una determinada plataforma. Cuando esto ocurre, en lugar de guardar un puntero nulo en el delegate, se guarda otro método que únicamente lanza una excepción con el detalle del error.

3.4 Hola Mundo Parrocha.

Sin duda implementar "Hola Mundo" es el ejemplo más simple para ver la funcionalidad básica de Parrocha.

"Hola Mundo Parrocha" es un programa de consola escrito en C#[6] que utiliza la biblioteca Parro*cha* para imprimir una cadena por pantalla ejecutando un programa con Parrot[43].

PIR es un lenguaje de medio bajo nivel para la máquina virtual de Parrot[43]. PIR está desarrollado por el mismo equipo de Parrot[43] y se interpreta con el mismo ejecutable que viene con Parrot[43] para ejecutar los archivos binarios de la plataforma.

Para comenzar la implementación hay que comenzar por escribir el "hola mundo" en PIR. El código es el siguiente.

```
.sub 'hola mundo Parrocha'
        say "Hola mundo Parrocha."
.end
```

Salvamos el código anterior como holamundo.pir. Este código ya se puede ejecutar con la máquina virtual de Parrot[43]:

```
C:\Parrot-2.3.0\bin>parrot holamundo.pir
Hola mundo Parrocha.
```

Para ejecutar el programa con Parro*cha* es necesario compilar holamundo.pir a un binario de Parrot[43]. El comando para realizar esta compilación es el siguiente:

```
C:\Parrot-2.3.0\bin>parrot -o holamundo.pbc holamundo.pir
```

Este binario también se puede ejecutar desde la máquina virtual de Parrot[43]:

```
C:\Parrot-2.3.0\bin>parrot holamundo.pbc
Hola mundo Parrocha.
```

En estos momentos estamos en disposición de codificar el archivo en C#[6] que ejecuta el hola mundo de Parrot[43].

Desde MonoDevelop[37] o Visual Studio[57] creamos un ejecutable de consola y añadimos Parrocha.dll como referencia.

El código del método principal del ejecutable utiliza Parro*cha* para acceder al API Embed de Parrot[43]. Gracias a Parro*cha* el programador, una vez cargada la biblioteca, puede traducir los ejemplos de C a C#[6] fácilmente. Solamente hay que tener en cuenta que el nombre de la función C es el método del objecto runtime adaptado al código de estilo de C#[6].

```
using System;
using Parrocha;

namespace HolaMundoParrocha
{
    class Program
    {
        static void Main(string[] args)
        {
            // Cargar la biblioteca de Parrot.
            Parrocha.ParrotRuntime runtime =
                new ParrotRuntime(@"c:\Parrot-2.3.0\bin\libparrot.dll");

            // Leer el archivo con el hola mundo de Parrot.
            IntPtr read =
                runtime.ParrotPbcRead(@"c:\Parrot-2.3.0\bin\holamundo.pbc", 0);

            // Si no ha sido posible cargar escribo el error.
            if (read == IntPtr.Zero)
            {
                Console.Error.WriteLine("No se ha podido cargar binario de parrot");
                Environment.Exit(-1);
            }

            // Cargo el archivo leido para ser ejecutado.
            runtime.ParrotPbcLoad(read);

            // Ejecuto el hola mundo.
            runtime.ParrotRuncode(0, null);
        }
    }
}
```

El resultado de la ejecución en consola del programa es el mismo que cuando se utiliza parrot.exe:

```
C:\Users\jmribao\Documents\Visual Studio 2008\Projects\Parrocha\HolaMundoParroch
a\bin\Release>HolaMundoParrocha.exe
Hola mundo Parrocha.
```

4 Desarrollo en comunidad

4.1 *Portales de desarrollo colaborativo.*

Existen numerosos portales de desarrollo colaborativo gratuitos en Internet para albergar proyectos libres.

Un portal de desarrollo colaborativo implementa servicios necesarios para facilitar el desarrollo en comunidad. Entre estos servicios podemos destacar la página web, foros, seguimiento de errores, servidor de ficheros para las publicar versiones y sistema de control de versiones.

Se puede publicar un proyecto es más de un portal de desarrollo colaborativo. En lo referente a la página web y a la publicación de versiones no plantearia ningún problema. Lo que no resulta recomendable es tener dos sistemas de control de versiones y dos servidores de seguimiento de errores.

Con el fin de no tener duplicados los servicios de versionado y seguimiento de errores se hace necesario escoger únicamente uno.

SourceForge[52] es el portal colaborativo más famoso y el que más proyectos libres contiene. La única desventaja de subir el proyecto a SourceForge[52] es que se trata de un repositorio demasiado generalista.

Parro*cha* es un proyecto desarrollado en C#[6], destinado sobre todo a desarrolladores de .Net[1]. Codeplex[9] es un portal colaborativo lanzado por Microsoft en Mayo de 2006 para promocionar el desarrollo de proyectos *opensource* para su plataforma de desarrollo .Net[1]. Codeplex[9] es propiedad de Microsoft y está basado en su propia tecnología de desarrollo colaborativo Team Foundation System.

Aunque Codeplex[9] no esté basado en software libre, resulta la mejor alternativa para la promoción del proyecto porque en el se encuentran

proyectos de temática muy similar a Parro*cha*. En Codeplex[9] se encuentran publicados los proyectos *opensource* de Microsoft® de implementación de lenguajes dinámicos DLR[16], IronPython[23] e IronRuby[24].

Con el fin de permitir en un futuro Parro*cha* pueda ser parte oficial de Parrot[43] se ha escogido la misma licencia libre bajo el que se publica este como se detalla en el siguiente apartado.

4.2 Licencia de la implementación.

Parro*cha* se publicará bajo la licencia Artistic License ver 2.0[5]. La elección de licencia es muy importante en los desarrollos de software libre.

Aunque nos podríamos haber decantado por una licencia más restrictiva, como la GPL[19], la licencia de Parro*cha* es la misma que Parrot[43] por dos motivos:

- Promocionar la biblioteca dentro de la comunidad de usuarios de PERL utilizando la licencia más difundida en esta comunidad.
- Cumplir con el acuerdo de licencia de los contribuidores del proyecto Parrot[43] facilitando la futura incorporación de Parro*cha* a dicho proyecto.

5 Trabajo futuro.

El trabajo en Parro*cha* debe de completarse con:

- Actualización a la última versión de Parrot[43].
- Finalización de la implementación en Linux®[32] y MacOSX®[35].
- Escritura de las pruebas unitarias para los métodos mapeados en C#[6].

A los largo de la introducción se han mencionado otras formas de interoperatividad entre máquinas virtuales. Tomando como base Parro*cha e* ideas de otros proyectos de software libre, como Parakeet[42] o IKVM.Net[22], la interoperatividad entre Parrot[43] y CLI[8] podría completarse con:

5.1 ***Mapeo de macros y PMCs del núcleo de Parrot.***

En los archivos de cabecera de Parrot[43] se encuentran numerosas macros de C. Las macros de C no son funciones que sean exportadas con la biblioteca, simplemente son atajos que trata el precompilador de C.

Por motivos de tiempo se han eliminado del proyecto el mapeo de macros y de los PMCs del núcleo de la máquina virtual Parrot[43].

En C#[6] no hay macros, el compilador JIT de la máquina virtual a través de una serie de reglas expande en determinados casos los métodos de una clase. Las macros de C en C#[6] deberían de implementarse como métodos. El único motivo por el que no se ha realizado este proceso es el corto periodo de tiempo disponible para la realización del proyecto.

En el API C expuesto por libparrot también se incluyen funciones que nos permiten tratar con objetos PMC del núcleo de la máquina virtual. Estos objetos, como vimos en el apartado 1.3. están escritos en C con la ayuda del precompilador pmc2c.pl.

Para mejorar el rendimiento en el acceso a las propiedades de los PMC del núcleo resultaría útil el desarrollo de otro precompilador que generase automáticamente el envoltorio de las funciones escritas por pmc2c.pl.

5.2 ***Recubrimientos de los APIs de Embed de Mono y Hosting de .Net a través de PMCs nativos.***

En el apartado 1.3. se vio que los tipos de los objetos de Parrot[43] se

denominan PMCs. Un PMC es el equivalente a una clase en Java[27] o en CLI[8]. Los PMCs pueden ser implementados como binarios de Parrot[43] o en C con la ayuda del precompilador pmc2c.pl.

La máquina virtual de Mono[36], al igual que Parrot[43], tiene un API C que permite empotrarla en programas nativos. El equivalente al API Embed de Mono[36] en .Net[1] es el API Hosting. Desde cualquiera de estos dos APIs es posible cargar ensamblados e invocar a métodos en código intermedio en la máquina virtual correspondiente.

Parro*cha* es capaz de cargar la máquina virtual de Parrot[43] en Mono[36] y .Net[1]. El desarrollo del PMCs en código C recubriendo el API Mono[36] Embed y Hosting de .Net[1] permitirían la interoperatividad de Parrot[43] y CLI[8] en sentido contrario.

5.3 ***Implementación de Parrot sobre Mono.***

A lo largo de los apartados 1.2 y 1.4 se tratan las limitaciones de CLI[8] para lenguajes dinámicos y los problemas de implementar una máquina virtual sobre otra.

El esfuerzo necesario para el diseño y codificación del envoltorio de libparrot es muy inferior al requerido para el desarrollo de una máquina virtual completa de Parrot[43] para máquinas CLI[8]. Si bien, la realización del envoltorio si permite hacerse una idea del funcionamiento interno de Parrot[43] y de los principales inconvenientes de desarrollo de la misma.

6 **Conclusiones.**

Al finalizar el proyecto fin de máster Parro*cha* contiene el mapeo de las APIs Embed y Extend de la versión 2.3.0 de Parrot[43] y es capaz de cargar el intérprete en Windows[58] a partir de la dirección donde se ha instalado la máquina virtual.

En el momento de presentar este trabajo la última versión de Parrot[43] es la 2.4.0. El equipo de Parrot[43] libera versiones de la máquina virtual cada 2 meses. Durante el desarrollo de Parro*cha* ya fue necesario actualizar de la versión 2.2.0. a la versión 2.3.0.

Con la versión 2.4.0. el equipo de Parrot[43] ha escrito test unitarios para las funciones de las APIs nativas de la plataforma. Parro*cha* al no contar con estos test no ha podido ser probado concienzudamente.

Con Parrot[43] he podido aprender más sobre máquinas virtuales. Parrot[43] es una máquina virtual muy diferente a .Net[1], Mono[36] o Java[27]. La arquitectura de Parrot[43] es consecuencia del modelo de desarrollo del software libre, y solamente a partir del software libre se explica el revolucionario modelo basado en continuaciones y registros en el que se basa.

El modelo de desarrollo de software libre también tiene sus contrapartidas. En el caso concreto de Parrot[43] este modelo recibió muchas críticas hasta la publicación de la primera versión, llegó a parecer software abandonado.

Parrot[43] ha sido siempre software libre y el modelo de desarrollo del software libre, según menciona Raymond en "La catedral y del bazar" no es el más indicado para los comienzos de un proyecto. El software libre necesita una versión parcialmente funcional para comenzar a desarrollarse con la que los miembros de la comunidad puedan jugar.

A partir de esa primera versión es necesario sacar versiones cada poco tiempo. Resulta muy interesante observar la rápida evolución de un proyecto donde se liberan versiones cada poco, en Parrot[43] cada dos meses.

El ciclo de vida del proyecto da idea de la actividad que hay en la comunidad, aunque para mi desarrollo ha supuesto tres importantes problemas:

- Imposibilidad de tener preparada una versión del envoltorio con la última

versión de Parrot[43] para la presentación del proyecto.

- Falta de actualización de la documentación. El código de las cabeceras no se correspondía completamente con la documentación del proyecto.
- Solamente he podido probar Parrot[43] en Windows[58]. El único binario que se encontraba actualizado con la última versión era el de Windows[58], recientemente también se ha publicado un paquete con la versión 2.3.0 de Debian Linux®[14].

Apéndice A) **Bibliografía.**

Nº	Nombre	Dirección
1	.Net	http://www.microsoft.com/net/
2	Ada	http://www.adahome.com/
3	Android	http://www.android.com/
4	Apache Web Server	http://httpd.apache.org/
5	Artistic License V. 2.0	http://www.opensource.org/licenses/artistic-license-2.0.php
6	C#	http://www.ecma-international.org/publications/standards/Ecma-334.htm
7	Clang	http://clang.llvm.org/
8	CLI	http://www.ecma-international.org/publications/standards/Ecma-335.htm
9	Codeplex	http://www.codeplex.com/
10	COM	http://www.microsoft.com/com/default.mspx
11	CORBA	http://www.corba.org/
12	Dalvik	http://www.dalvikvm.com/
13	DaVinci	http://openjdk.java.net/projects/mlvm/
14	Debian Linux®	http://www.debian.org/
15	DirectX®	http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx
16	DLR	http://dlr.codeplex.com/
17	FUSE	http://fuse.sourceforge.net/
18	GCC	http://gcc.gnu.org/
19	GPL	http://www.gnu.org/licenses/licenses.html
20	GTK+	http://www.gtk.org/
21	GTK#	http://www.mono-project.com/GtkSharp
22	IKVM.Net	http://www.ikvm.net/
23	IronPython	http://ironpython.net/
24	IronRuby	http://ironruby.net/
25	IronScheme	http://ironscheme.codeplex.com/
26	Ja.Net SE	http://www.janetdev.org/
27	Java	http://www.java.com/
28	Jikes RVM	http://jikesrvm.org/

Nº	Nombre	Dirección
29	JNI4Net	http://jni4net.sourceforge.net/
30	KDE	http://www.kde.org/
31	Khronos	http://www.khronos.org/
32	Linux®	http://www.kernel.org/
33	LLVM	http://llvm.org/
34	LSL.Net	http://wiki.secondlife.com/wiki/LSL_Portal
35	MacOSX®	http://www.apple.com/macosx/
36	Mono	http://www.mono-project.com/Main_Page
37	MonoDevelop	http://monodevelop.com/
38	OpenCL	http://www.khronos.org/opencl/
39	OpenGL	http://www.khronos.org/opengl/
40	OpenTK	http://www.opentk.com/
41	Oracle®	http://www.oracle.com/us/products/database/index.html
42	Parakeet	http://parakeet.sourceforge.net/
43	Parrot	http://parrot.org/
44	PERL6	http://dev.perl.org/perl6/
45	PostgreSQL	http://www.postgresql.org/
46	Python	http://www.python.org
47	Qt	http://qt.nokia.com/
48	Qyoto/Kimono	http://www.ohloh.net/p/qyoto
49	Ruby	http://www.ruby-lang.org/
50	SlimDx	http://slimdx.org/
51	SOAP	http://www.w3.org/TR/soap/
52	SourceForge.Net	http://sourceforge.net/
53	SQL Server®	http://www.microsoft.com/sqlserver/2008/en/us/default.aspx
54	Unlanden Swallow	http://code.google.com/p/unladen-swallow/
55	Visual Basic.Net	http://msdn.microsoft.com/vbasic/default.aspx
56	Visual C++	http://msdn.microsoft.com/visualc/default.aspx
57	Visual Studio	http://msdn.microsoft.com/vstudio/default.aspx
58	Windows(tm)	http://www.microsoft.com/windows/default.aspx