Desarrollo de Software con Mono, una Implementación Libre de .NET, Multiplataforma e Independiente del Lenguaje

Luis I. Larrateguy, Milton D. Pividori y César M. Sandrigo

UTN Facultad Regional Santa Fe, Lavaise 610 - S3004EWB Santa Fe,
Página Web: http://www.frsf.utn.edu.ar/

Resumen Se presenta un desarrollo como prueba de concepto en la integración de tecnologías, utilizando Software Libre y multiplataforma. Se utilizó la plataforma de desarrollo Mono, implementación libre de .NET. Se probaron tecnologías como Remoting, Web Services, independencia del lenguaje de programación y *toolkits* gráficos libres. Además se utilizó el entorno de desarrollo integrado libre MonoDevelop. Se abordó la solución con técnicas orientadas a objetos y se aplicaron patrones de diseño. El marco de este trabajo fue la cátedra de Administración de Recursos de cuarto año de Ingeniería en Sistemas de Información, en donde la consigna del mismo era investigar sobre tecnologías actuales y nuevas que permitieran la integración de tecnologías. Uno de los objetivos propuestos por los autores, fue aprender a utilizar tecnologías libres y modernas utilizando Software Libre. En el caso de estudio se probó portabilidad entre Ubuntu GNU/Linux y Microsoft Windows.

1. Introducción

Durante el cursado de la cátedra Administración de Recursos, dictada en el 4^{to} año de la carrera de Ingeniería en Sistemas de Información, se realizó un trabajo práctico de investigación y desarrollo, con el objetivo de aprender a utilizar nuevas tecnologías para desarrollar software.

El Software Libre (SL) brinda varias ventajas para los clientes, usuarios y desarrolladores. Para los clientes la principal de ellas es la independencia que proporciona sobre las empresas proveedoras de software, habilitándolos a contratar a los profesionales que prefiera para realizar mantenimiento de su sistema, expansión, diseño de nuevas funcionalidades, entre otras. Para los usuarios, la seguridad de estar trabajando con estándares abiertos, ya que el SL hace un uso extensivo de estos, y la posibilidad de elegir entre varias alternativas. Los desarrolladores se benefician de los aportes que puedan realizar sus pares, de contar con una gran comunidad para el soporte, incluso de realizar trabajo diferencial sobre una pieza de software existente.

Una característica del Framework .NET de Microsoft interesante y que lo diferencia de otras plataformas, es su enfoque en la independencia del lenguaje de programación. El objetivo del trabajo es investigar esta característica, utilizando

varios lenguajes en el desarrollo, como C#, Java y Boo, y comunicándolos entre ellos a nivel de clase. Esto permite formar equipos de desarrollo donde el lenguaje común entre los individuos sea la *Orientación a Objetos*, y no un lenguaje de programación. De esta forma cada programador puede trabajar en el lenguaje que desee, sin preocuparse por la interoperabilidad con el código de los demás, habilitándolo a, por ejemplo, crear una nueva clase en Java que especialice una hecha en C#, y a su vez ésta especializar otra codificada en Boo o VB.NET.

Se investigó además características de integración con otros sistemas utilizando estándares abiertos con protocolos cerrados. Para esto se utilizó Remoting dentro de la plataforma .NET y Web Services para comunicarse con otras plataformas.

Mediante el planteo de un problema a resolver y la construcción de una prueba de concepto, se busca cumplir los objetivos propuestos.

2. Investigación

Al ser Mono una implementación del Framework .NET, es necesario investigar inicialmente sobre esta plataforma, ya que se trata básicamente de los mismos conceptos. Sin embargo se requiere conocer aquellas características propias del proyecto Mono, como la capacidad de correr en varias plataformas (de hardware y software) y las alternativas que propone para realizar algunas tareas.

En .NET las bibliotecas que se utilizan para crear una Interfaz Gráfica de Usuario (GUI, por sus siglas en inglés) se llaman Windows.Forms. Gtk# es otro conjunto de bibliotecas alternativa que presenta una serie de ventajas, como el hecho de ser multiplataforma.

También se investigó sobre tecnologías como Remoting y Web Services, que se han utilizado para escribir los componentes de comunicaciones, cada una con características apropiadas para interconectar sistemas homogéneos o heterogéneos.

2.1. .NET

No es objetivo del trabajo entrar en detalles sobre la plataforma .NET, sino mencionar los conceptos principales, necesarios para comprender las secciones siguientes, especialmente la de Mono.

Unas de las características del Framework .NET, es la compilación JIT (Just In Time), gestión de memoria (Garbage collector), *multithreading*, interoperabilidad, bibliotecas de clase base (BCL) disponible para todos los lenguajes, características de seguridad e independencia del lenguaje.

La independencia del lenguaje que ofrece la plataforma .NET es lograda compilando todo a un código intermedio, llamado CIL (Common Intermediate Language), como se puede apreciar en la Figura 1. Cada compilador transforma el código fuente de un determinado lenguaje a CIL. Luego el runtime (entorno de ejecución) se encarga de ejecutar el CIL, compilándolo a código máquina. Más información sobre el framework en [7].

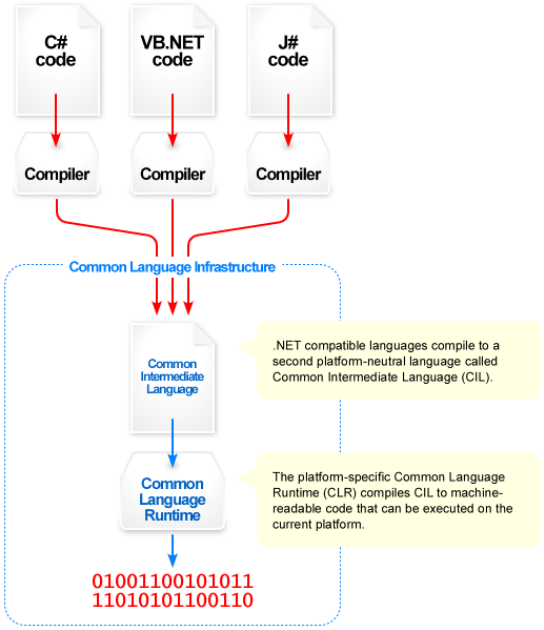

Figura 1. Independencia del lenguaje en .NET

2.2. Mono

Antes de explicar el resultado de la investigación acerca de características particulares y paquetes de Mono, se explicará brevemente de qué se trata el **Proyecto Mono**.

¿Qué es Mono?. En el sitio oficial del proyecto se lee: "Mono es una plataforma de software diseñada para permitir a los desarrolladores crear fácilmente aplicaciones multiplataforma. Es una implementación *open source* del Framework .NET de Microsoft basado en estándares ECMA para C# y el *Common Language Runtime*"[1]. Puede leerse más sobre los estándares ECMA en [2].

Según palabras de uno de los fundadores del proyecto, Miguel de Icaza, el objetivo de Mono originalmente era mejorar la plataforma de desarrollo en GNU/Linux. A medida que la comunidad crecía, se comenzó a soportar las APIs de Microsoft. Finalmente, cuando Mono se volvía cada vez más maduro, el objetivo era proveer de un runtime multiplataforma completo, y de permitir a los desarrolladores de Windows portar sus aplicaciones a GNU/Linux.

Activamente Desarrollado. El proyecto Mono es impulsado por Novell, y según datos del 2006, cuenta con 20 desarrolladores full-time, y más de 300 colaboradores.

Es Libre[3]. El compilador de C# y demás herramientas, están bajo la GNU General Public License (GPL). Las librerías de tiempo de ejecución bajo la GNU Library GPL 2.0 (LGPL), y las librerías de clase bajo la licencia MIT X11.

Es Realmente Multiplataforma[4]. A diferencia de la implementación cerrada de Microsoft, Mono es altamente portable: está disponible para casi toda la familia Unix, como GNU/Linux, Mac OS X, Solaris y BSDs, además de funcionar en Windows, a partir de la versión 2000. También está disponible en la plataforma de desarrollo libre para dispositivos móviles Maemo. Además corre en distintas arquitecturas de CPU: x86, PowerPC, AMD64, Sparc, s390, IA64, ARM.

En la Figura 2 se puede ver un gráfico que permite visualizar la independencia del lenguaje, y la capacidad de correr en varias plataformas.

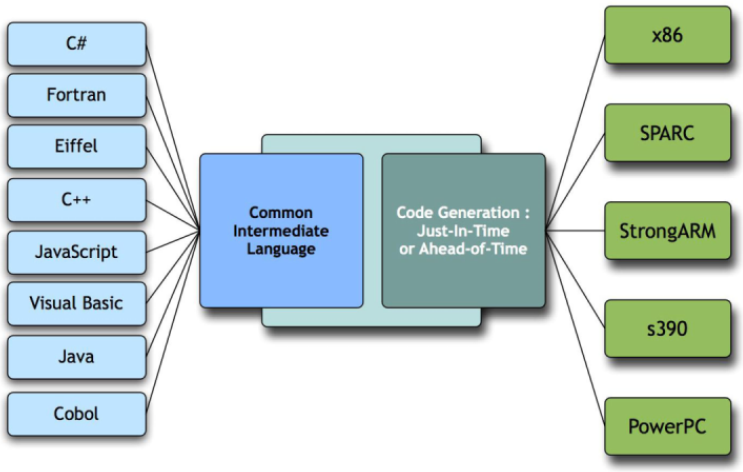

Figura 2. Mono: independiente del lenguaje y multiplataforma

¿Quién usa Mono?. Algunos usuarios de esta plataforma de desarrollo libre son[5]:

- **Novell** (http://www.novell.com/), tanto para aplicaciones cliente como servidor. Algunas de ellas son: iFolder, Beagle, F-Spot, Banshee, etc.
- **Second Life** (http://secondlife.com/): en el futuro, se utilizará Mono como motor de scripting, por la performance que ofrece en la ejecución de scripts[8].
- **Wikipedia** (http://wikipedia.org) usa Mono para sus recursos de búsqueda. El indexado y la búsqueda es realizada por aplicaciones basadas en Mono.
- **Cogmation** (http://www.cogmation.com/) usa Mono en su software de robótica para manejar el scripting en su plataforma[6].

- **Otee** (http://www.otee.dk/) desarrolla Unity, herramienta para el modelado 3D de juegos, que utiliza Mono en algunos componentes.
- **Interopix** (http://www.interopix.com/): Corren Mono en un servidor FreeBSD con alta carga.
- **Coversant** (http://www.coversant.net/): Desarrolladores de SoapBox, un servidor Jabber.
- **Zing** (http://www.zing.net): su reproductor MP3/Wifi usa Mono para manejar su stack de aplicaciones.
- **VistaDB** (http://www.vistadb.com) tiene en el mercado su base de datos embebida VistaDB en Mono.

2.3. Gtk#

Como se mencionó anteriormente, éste es el toolkit gráfico que se ha utilizado para el software desarrollado. Representa una alternativa para el desarrollo de GUIs, con una serie de ventajas que se verán a continuación. Se pretende en esta sección dar una breve introducción al respecto.

¿Qué es Gtk+?. "Gtk+ es un toolkit altamente utilizable y completo para crear interfaces gráficas de usuario, y se jacta de su capacidad multiplataforma y de una API fácil de usar."[9].

Si bien Gtk+ está escrito en C, existen *bindings* para otros lenguajes como C++, Python y Java. Además está bajo la licencia GNU LGPL 2.1, lo que permite utilizarla para desarrollos libres o propietarios, sin cargos o que implique el pago de regalías.

¿Quién lo Utiliza?. Algunos grandes usuarios de este toolkit gráfico son el proyecto GNOME y la plataforma Maemo. Gtk+ es soportado por una amplia comunidad de desarrolladores, y algunos de los principales son de compañías como Red Hat, Novell, Imendio y Opened Hand[11].

Multiplataforma. Gtk+ corre en GNU/Linux, Unix, Windows y Mac OS X.

¿Qué es Gtk#?. Gtk# es el nombre de los *bindings* correspondientes de Gtk+ para la plataforma Mono/.NET. Es posible utilizar este toolkit gráfico desde cualquier lenguaje soportado por la plataforma, no sólo C#.

2.4. Remoting

Se llama *Remoting* a un conjunto de servicios que permite la comunicación entre procesos. Estas aplicaciones pueden residir en la misma computadora, o en diferentes estando conectadas por una LAN o Internet. Permite la comunicación entre objetos de diferentes dominios de aplicación o procesos. Es posible utilizar diferentes protocolos de transporte, optar por varias formas de serialización, configurar el esquema de tiempo de vida de los objetos y elegir entre varios modos de creación de los mismos. Se puede ver un gráfico de la arquitectura de Remoting en la Figura 3.

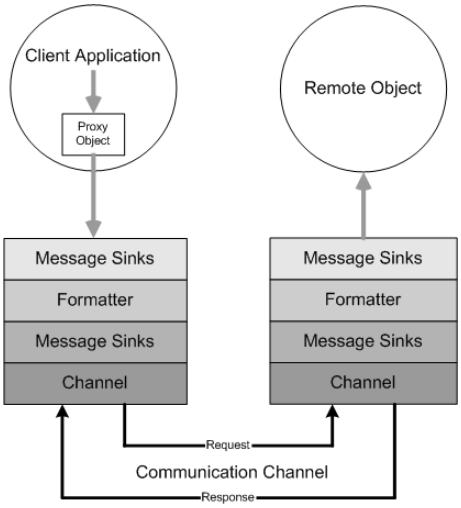

Figura 3. Arquitectura de Remoting

Transporte de Objetos. Los objetos pueden transferirse entre procesos de dos formas: serialización por copia, o serialización por referencia[13].

Serialización por Copia. Los objetos pueden residir en el cliente, mediante serialización, pasando el objeto a una cadena XML, o algún otro tipo de representación. En inglés, esta técnica recibe el nombre de *marshaling by value*.

Cuando un proceso recibe un objeto por copia, al ejecutar sus métodos, lo está haciendo localmente. Es decir que se ejecutan en el mismo proceso donde se encuentra el objeto. Lo que podemos realizar con este modo de transporte, es pasar información de un lugar a otro.

Serialización por Referencia. Esta posibilidad permite ejecutar métodos en objetos remotos, y es la que se utilizó en la prueba de concepto. En esta modalidad, se serializa una referencia a un objeto, el cual permanece en el servidor. El cliente recibe un *proxy* que atiende sus pedidos y se comunica con el objeto remoto, ejecutándose los métodos del lado servidor. De esta forma, en el cliente pareciera que se está utilizando un objeto local.

Modos de Activación. Hay distintas formas en las que se llevan acabo las operaciones anteriormente explicadas. Los objetos se pueden activar en el servidor o en el cliente[14].

Activación en el Servidor. En este modo, el servidor publica un objeto de una clase remota a través de una URL. Es el mismo servidor el que controla cómo el objeto se crea y su tiempo de vida.

Uno de los modos es *Singleton*, en el que un solo objeto es creado para todos los clientes, por lo tanto todos ellos comparten su estado. En este caso la instancia se crea la primera vez que el cliente llama a uno de los métodos.

Otra modalidad es llamada *SingleCall*. A diferencia del anterior, cada llamada a un método remoto crea una nueva instancia. Este nuevo objeto se destruye cuando la llamada finaliza.

Un tercer modo es publicar directamente un objeto. Se utiliza cuando el mismo se debe crear utilizando algún constructor parametrizado, y no el por defecto, que se utiliza en el primer caso (*Singleton*).

Activación en el Cliente. En este modo de activación hay más control por parte del cliente. El objeto remoto sigue residiendo en el servidor, pero hay una relación uno a uno con el cliente. Es decir, el cliente puede estar seguro que su instancia mantendrá su estado interno (en realidad, esto no siempre es así, pero esta excepción va más allá de los límites de este trabajo). Otro aspecto del objeto que controla el cliente es su tiempo de vida.

2.5. Web Services

Los *web services* o *servicios web*, son interfaces accesibles a la funcionalidad de una aplicación, construidas sobre estándares de tecnologías que utilizan Internet [15]. En otras palabras, cualquier aplicación que pueda ser accedida a través de una red usando protocolos como HTTP, SMTP, XMPP, en definitiva es un *servicios web*. Estos no son nada nuevo, sino que representan una evolución en los principios que sostienen el uso de Internet desde el comienzo. Sin embargo se basan en nuevos estándares que definen la comunicación entre el proveedor, consumidor y publicador de servicios. En la Figura 4 se puede apreciar la arquitectura de la solución de *web services*[16].

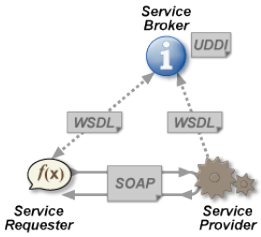

Figura 4. Arquitectura de web services

Los *servicios web* sirven como una capa de abstracción entre la aplicación cliente y el código de la aplicación. Gracias a esta abstracción, los clientes pueden ser implementados en distintos lenguajes, consumiendo esos servicios para realizar las tareas. Cabe aclarar que no sólo debe pensarse en la parte cliente de

un modelo cliente/servidor, sino que una aplicación que se ejecuta en un servidor puede estar haciendo uso de *servicios web* para resolver los requerimientos.

Un ejemplo del uso de *servicios web* es una aplicación web, donde el cliente selecciona el idioma en el que desea ver determinado contenido y envía su REQUEST al servidor. Éste recibe la petición, y mediante un algún *servicios web* de traducción, traduce sus contenidos para devolvérselos al cliente. Otro ejemplo es una aplicación web que tenga implementado con Javascript un cliente, que consulta los últimos valores de cotización de determinada moneda sin intervención por parte del servidor de la aplicación. Es decir, el cliente se comunicó directamente con otro servidor para consumir ese servicio. Sin embargo los *servicios web* no solo se limitan a aplicaciones web, sino que pueden ser utilizados por cualquier aplicación con conexión a una red interna o Internet.

Los *servicios web* además permiten la integración de tecnologías dispares que antes su interoperabilidad se veía comprometida o complicada [15]. Se pueden ver como una evolución de las llamadas a procedimientos remotos (en inglés RPC), donde no importa la plataforma ni el lenguaje de implementación del consumidor ni del proveedor de los servicios. Para esto se utilizan estándares como SOAP para la comunicación, WSDL para la definición y UDDI para el descubrimiento, siendo XML la base de estos lenguajes [15].

Se investigó además acerca de la existencia de muchas soluciones alrededor de los *servicios web*, como ser descubrimiento, publicación, seguridad, entre otros, aunque no fueron abordados porque escapaban del alcance del trabajo.

3. Planteo de un Problema a Resolver

Para poder realizar una prueba de concepto con el fin de cumplir los objetivos, se planteó un escenario con un problema a resolver. Como se expuso anteriormente se quiere mostrar la integración de diversas tecnologías, así también como la integración de lenguajes preparados para correr sobre la plataforma Mono con otros que no lo estaban (como ser Java). Por supuesto que todo esto utilizando Software Libre. El problema que se eligió resolver fue el de enviar "mensajes instantáneos" a través de una red. Las condiciones del caso de uso son las siguientes:

- Clientes que se pudieran comunicar enviando mensajes de texto plano.
- Ver contactos conectados.
- Poder enviar y recibir mensajes mediante un sistema web.

Para resolver este problema se plantearon las siguientes restricciones:

- Los clientes usarían Remoting para enviarse los mensajes.
- Programar en distintos lenguajes de programación.
- El cliente web debía funcionar utilizando Web Services.
- Utilizar exclusivamente Software Libre.

4. Arquitectura

Se planteó una arquitectura cliente - servidor para la comunicación entre los clientes desarrollados que correrán sobre la plataforma Mono (o .NET). Para los clientes que utilizarán Web Services, la arquitectura planteada difiere en que cada una de las acciones son expuestas en formas de servicios (conectarse, enviar mensajes, recibir mensajes, ver lista de contactos, desconectarse) los cuales son ejecutados por "clientes virtuales", que representan al cliente web en su ausencia. Esta decisión se tomó debido a cómo funciona un cliente web, que no conserva un estado o sincronismo con el servidor, y la naturaleza *pull* del protocolo HTTP[15]. Si bien se evaluó utilizar alguna técnica *push* desde el servidor al cliente, escapa de nuestro objetivo de estudio.

La Figura 5 muestra la arquitectura propuesta para resolver el problema de interoperabilidad y poder hacer una prueba de integración, no sólo de clientes que corran sobre la plataforma .NET, sino con clientes implementados en otra plataforma (como PHP) utilizando Web Services.

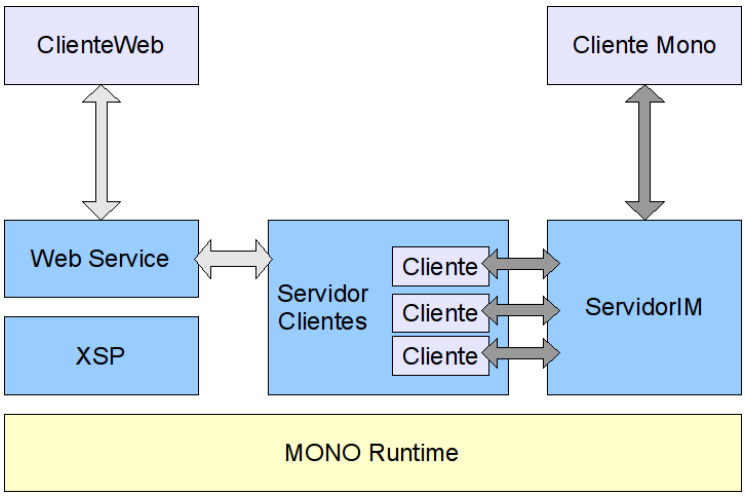

Figura 5. Mensajero: Arquitectura propuesta

El *ServidorIM* es el servidor que realiza la comunicación entre los clientes, y les informa de los eventos ocurridos. Para esto se optó por la modalidad *Singleton* explicada anteriormente en la sección de Remoting, donde todos los clientes comparten el mismo objeto, en este caso el *servidor*. El *Servidor Clientes*, provee una interfaz para crear clientes que representen a un *ClienteWeb* (el que trabaja en un modo de poleo como fue explicado) para no perder los eventos enviados por el servidor. Estos van capturando los eventos y modificando su estado interno para cuando los clientes web soliciten el estado, le envíe la sumatoria de esos

eventos ocurridos. De esta forma, se aprovecha toda la estructura diseñada con Remoting para brindar los Web Services correspondientes.

5. Implementación y Despliegue

En la Figura 6, se puede observar como fue implementada esa arquitectura, utilizando Remoting. Especializando `MarshalByRefObject`, se obtienen clases que pueden generar instancias únicas tanto del servidor como de los clientes. El Servidor mediante un stub de `ClienteRemoto` puede informar de los eventos a cada cliente, y éstos pueden enviar un mensaje al objeto remoto `ControladorConexiones` mediante un stub creado por Remoting. En la Figura 7, se muestran los paquetes en los cuales se implementó por separado, primero fijando las interfases necesarias para poder continuar el desarrollo en forma conjunta y distribuida.

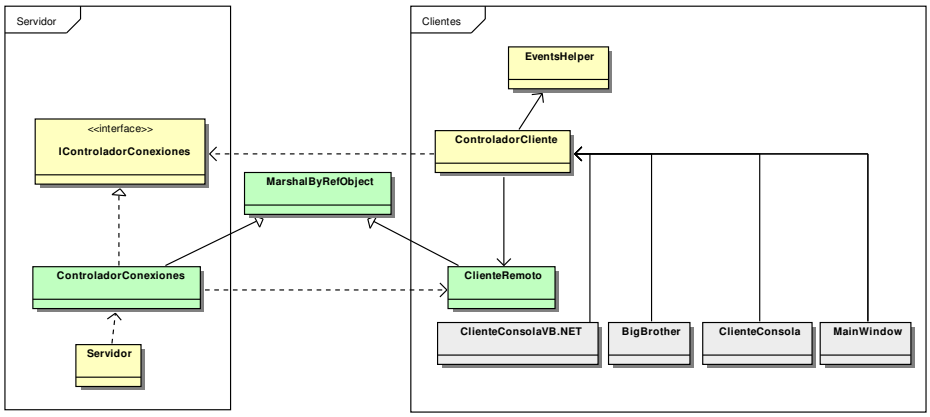

Figura 6. Diagrama de clases correspondientes al cliente y al servidor

5.1. Implementación en Distintos Lenguajes

Cada paquete está programado en un lenguaje diferente. Se utilizó C#, Boo, Java (mediante IKVM.NET) y VB.NET. Con otros lenguajes también se hicieron pruebas pero no estaban lo suficientemente maduros.

Al compilar cada uno de estos paquetes se obtiene código CIL. Sin embargo, se tuvo en cuenta algunas consideraciones al programar en distintos lenguajes, algunos diseñados específicamente para trabajar sobre la CLI, como C#, Boo y VB.NET, y otros donde se utilizó un compilador que generaba CIL pero de un lenguaje existente como Java.

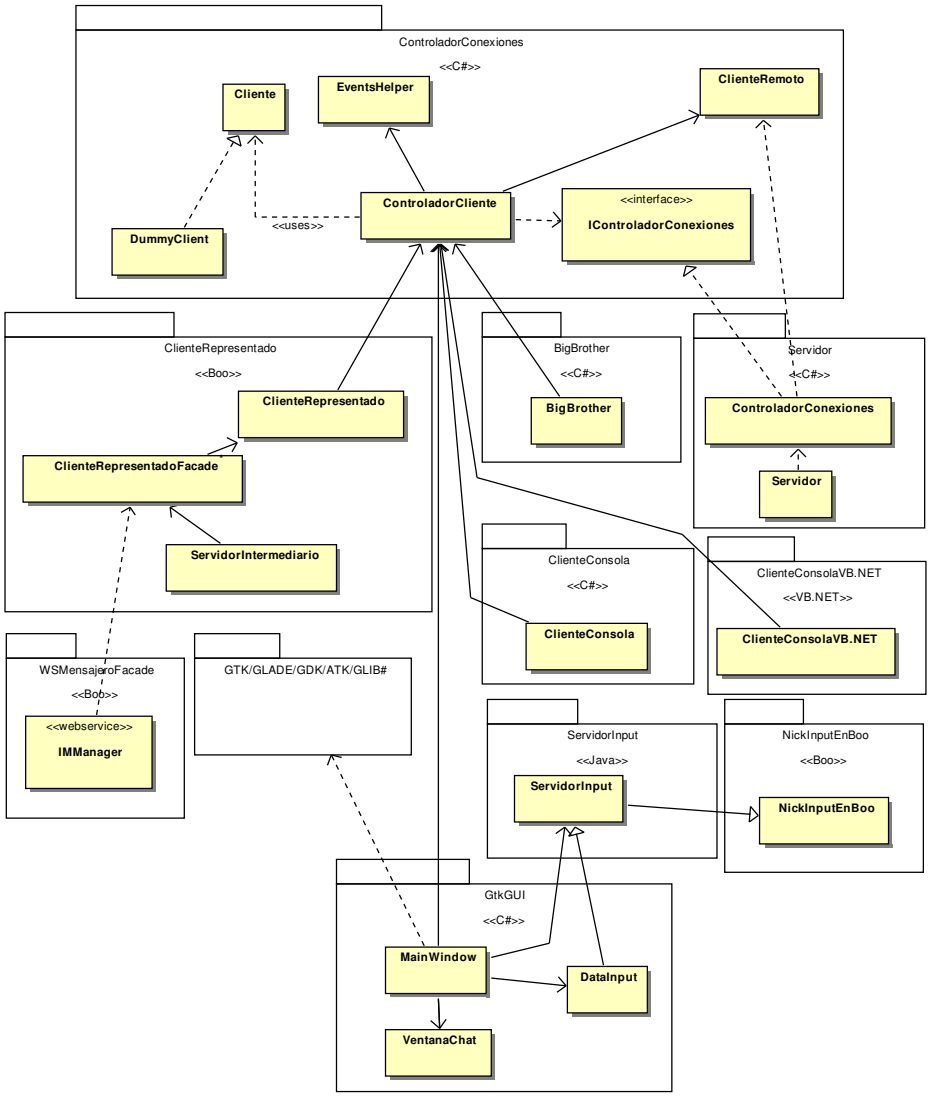

Figura 7. Diagrama con todas las clases del sistema

C# posee una característica que se llama *Property*. Lo que permiten las *Propiedades* es acceder de una forma más cómoda a las características de un objeto, como si se estuviera modificando un atributo con el operador asignación pero permitiendo hacer validaciones de todos modos. Java carece de dicha característica, así que las *Propiedades* dentro de Java se veían como *get_Property* y *set_Property*.

El paquete *ControladorConexiones* contiene las clases necesarias para la intercomunicación entre los clientes. Existe una interfaz `IControladorConexiones` la cual es implementada por `ControladorConexiones`. Esta interfaz define los servicios básicos que deberá proveer una clase que sea auxiliar de un proceso servidor. Como se puede observar en la Figura 7, en el paquete *Servidor*, se encuentran dichas clases. Todas éstas implementadas en C#.

`ControladorCliente` es la clase que brinda servicios a los distintos clientes que desean comunicarse con el servidor. Cabe destacar que tanto `ControladorConexiones` como `ControladorCliente` son los que brindan los servicios básicos de conexión, desconexión y envío de mensajes, a modo de **Façade**[17].

En los paquetes *ClienteRepresentado*, *ClienteConsola*, *GtkGUI*, *ClienteConsolaVB.NET*, y *BigBrother* se implementaron diversos clientes capaces de conectarse al servidor y comunicarse con el resto mediante mensajes de texto. En otras palabras un servicio simple de mensajería instantánea. Dejando de lado `ClienteRepresentado` que se abordará luego, el resto de los clientes especializan la clase `Cliente` del paquete *ControladorConexiones*. Estos clientes serán notificados mediante la ayuda de `EventHelper` de los eventos ocurridos, como ser conexión o desconexión de otro cliente, recepción de un mensaje.

`ClienteConsola` fue desarrollado en C#, manteniendo una funcionalidad simple, controlada mediante un ciclo con condición de salida que permitía elegir entre conectarse, desconectarse, ver contactos conectados, enviar un mensaje. En la Figura 8 se pueden ver algunas capturas del cliente funcionando. `ClienteConsolaVB.NET` fue desarrollado en VB.NET, con una funcionalidad idéntica al de C#, sólo con el propósito de probar el compilador VB.NET de Mono.

`GtkGUI` implementa una interfaz simple pero completa de mensajería instantánea. La interfaz gráfica en sí, está diseñada utilizando Glade. Glade permite diseñar las interfaces con un diseñador, pero el resultado es guardado en un archivo XML, lo que permite que el mismo diseño de interfaz se utilice en otra plataforma o lenguaje de programación, siempre y cuando existan enlaces a bibliotecas GTK/Glade. Se pueden ver capturas de dos clientes GTK corriendo en Ubuntu GNU/Linux, utilizando *Mono* y en Microsoft Windows Vista con el runtime de *.NET* en las Figuras 9 y 10.

Para conectarse cada cliente debe elegir un nickname, indicar la IP del servidor y el puerto donde éste está escuchando, como se puede apreciar en la Figura 11.

5.2. Implementación del Cliente Web Mediante Web Services

Como se mencionó anteriormente, uno de los objetivos del trabajo era probar además de la independencia del lenguaje, la interoperabilidad con otras plataformas. Para esto se hizo uso de los Web Services.

La clase `ClienteRepresentado` programada en Boo, implementa un cliente que será almacenado por `ServidorIntermediario`. Este servidor, corre a la par del servidor de mensajes original, comunicándose con éste. Su función es albergar

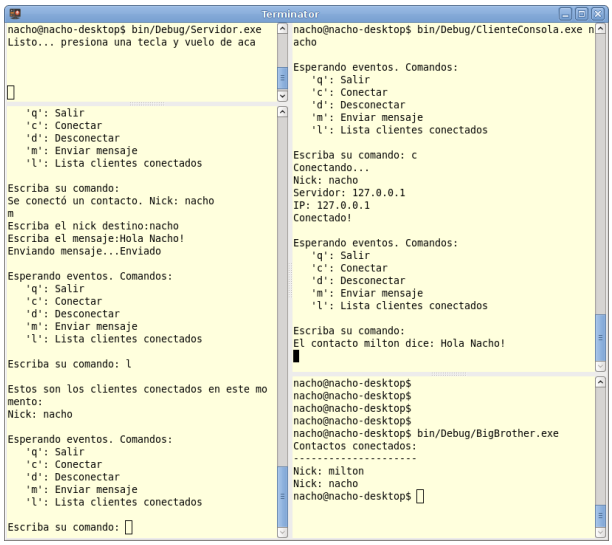

Figura 8. Captura de pantalla con 4 terminales. Servidor, ClienteConsola C#, ClienteConsolaVB.NET, y BigBrother mostrando los clientes conectados

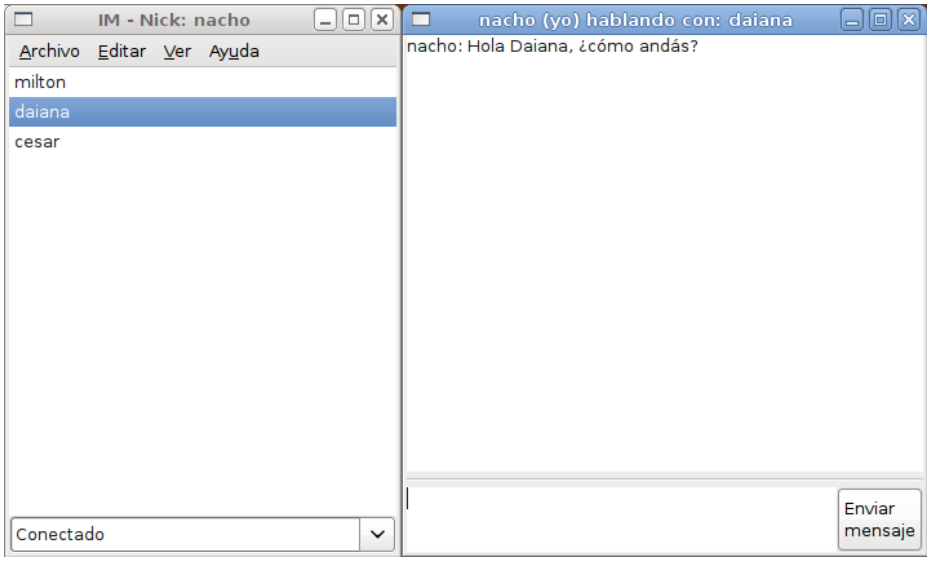

Figura 9. Captura de pantalla de la lista de contactos y una ventana de chat corriendo en Ubuntu

instancias de `ClienteRepresentado`. Cada una de estas instancias representan a

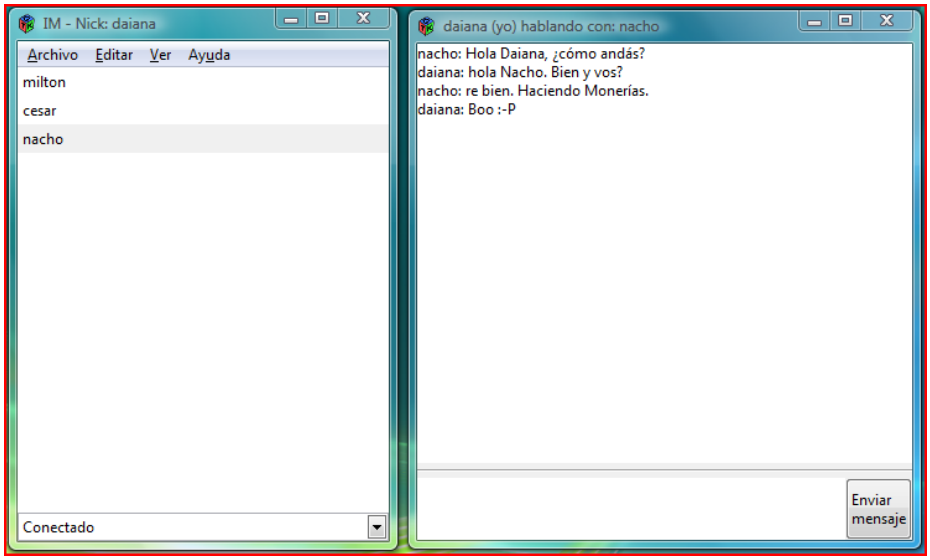

Figura 10. Captura de pantalla de la lista de contactos y una ventana de chat corriendo en Windows Vista, con el Framework .NET y GTK#

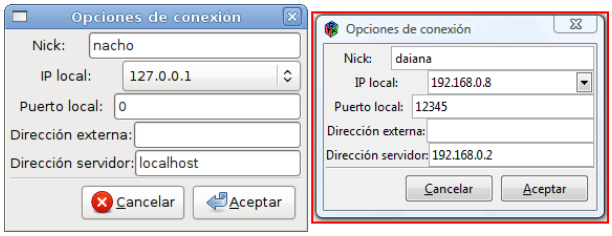

Figura 11. Captura de pantalla de la interfaz de conexión en Ubuntu y Vista

un cliente que está accediendo mediante la Web. Con esto se logra que un cliente que reside en un medio "sin estado" [18], al cuál no se le pueden "empujar" los datos (salvo que se utilice alguna técnica PUSH [20]) sino que éste los solicita mediante un REQUEST [19], pueda pedir por todos los mensajes recibidos hasta el momento por su representante.

`ClienteRepresentadoFacade` expone las funcionalidades de `ClienteRepresentado` para que los otros objetos Remoting puedan comunicarse con éste. `IMManager` es la clase que implementa el Web Service, ofreciendo los servicios en Internet. El fragmento de código en la Figura 12 muestra gracias al diseño utilizando patrones, y a la abstracción propuesta en .NET que resulta sencillo incorporar un cliente web a la arquitectura.

```
/* licencia GPL e imports quitados        */
/* codigo simplificado a modo de ejemplo */
[WebService (Description:"", Namespace:"http://localhost/
    webservices/examples/representante")]
class IMManager():

  [WebMethod(Description:"Autentica y crea un objeto remoto")]
  public def Conectar(nick as string) as bool:
    c  as  ClienteRepresentadoFacade = \
    Activator.GetObject(typeof(ClienteRepresentadoFacade),
    "tcp://127.0.0.1:8086/ClienteCreator")
    b1 = c.createClienteRepresentado(nick)
    if b1:
      b1 = c.conectar(nick)
    return b1

  [WebMethod(Description:"Desconecta del representante")]
  public def Desconectar(key as string) as bool:
    c  as  ClienteRepresentadoFacade = \
    Activator.GetObject(typeof(ClienteRepresentadoFacade),
      "tcp://127.0.0.1:8086/ClienteCreator")
    b1 = c.desconectar(key)
    if b1:
      c.destroyClienteRepresentado(key)
    return b1

  [WebMethod(Description:"Devuelve la lista de contactos")]
  public def GetListaContactos(key as string):
    c  as  ClienteRepresentadoFacade = \
    Activator.GetObject(typeof(ClienteRepresentadoFacade),
      "tcp://127.0.0.1:8086/ClienteCreator")
    contactos = c.getContactosConectados(key)
    return contactos

  [WebMethod(Description:"Devuelve la lista de contactos")]
  public def GetUltimosMensajesRecibidos(key as string):
    c  as  ClienteRepresentadoFacade = \
    Activator.GetObject(typeof(ClienteRepresentadoFacade),
      "tcp://127.0.0.1:8086/ClienteCreator")
    msjs = c.getUltimosMensajesRecibidos(key)
    return msjs
```

Figura 12. Código del servicio web implementado en Boo

El cliente web de la Figura 13 consiste de dos partes. Una parte "cliente" que es la que utiliza el usuario final, y otra "servidor" que es la que se comunica con el Web Service y consume estos servicios. La parte del servidor fue implementada

con PHP, utilizando la biblioteca NuSOAP[21] que es una API para acceder a los servicios brindado por un Web Service. La parte del cliente fue implementada, utilizando llamadas asíncronas con Javascript de fondo, con el fin de chequear por eventos en el servidor. Estas llamadas se repetían en un modo de "polling" cada un intervalo de tiempo determinado.

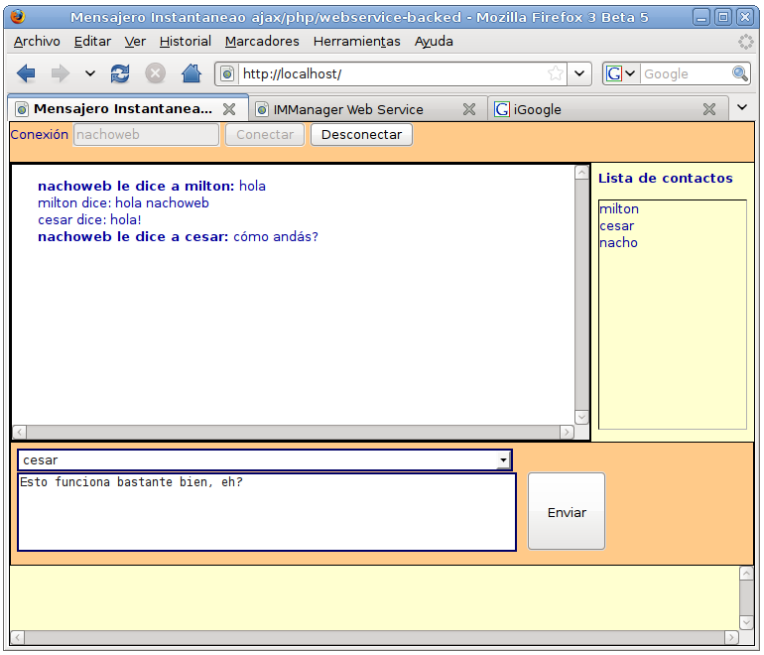

Figura 13. Captura de pantalla del cliente Web escrito en PHP, corriendo sobre un servidor Apache

El usuario del cliente web, ve una sola ventana de chat, en donde él se puede comunicar individualmente con cada persona, pero recibe los mensajes en una misma pantalla. Ésto es sólo una simplificación de implementación.

6. Conclusión

Las tecnologías abiertas y particularmente el Software Libre han ido avanzando y ganando terreno en la comunidad y el mundo empresarial. Cada día se ve más necesidad de personal capacitado en este área. El desarrollo del presente trabajo permitió a los autores conocer una nueva plataforma que se integra con otras existentes, privativas y libres. El involucrarse con estas tecnologías y seguir de cerca su evolución permite ir evaluando la madurez de los productos y la aceptación que va teniendo en la industria.

La arquitectura resultante de la *prueba de concepto* permitió probar funcionalidades de integración de tecnologías heterogéneas y ganar experiencia para futuros desarrollos utilizando Software Libre.

Referencias

1. Proyecto Mono - Sitio oficial: What is Mono. (acceso al 05/2008) http://www.mono-project.com/What_is_Mono
2. Proyecto Mono - Sitio oficial: ECMA. (acceso al 05/2008) http://www.mono-project.com/ECMA
3. Proyecto Mono - Sitio oficial: FAQ: Licensing. (acceso al 05/2008) http://www.mono-project.com/FAQ:_Licensing
4. Proyecto Mono - Sitio oficial: Supported Platforms. (acceso al 05/2008) http://www.mono-project.com/Supported_Platforms
5. Proyecto Mono - Sitio oficial: Companies Using Mono. (acceso al 05/2008) http://www.mono-project.com/Companies_Using_Mono
6. Cogmation Robotics: .Net scripting provided by The Mono Project. ©2007 Novell, Inc.(acceso al 05/2008) http://www.cogmation.com/mono-net-scripting.php
7. Wikipedia: .NET Framework. (acceso al 05/2008) http://en.wikipedia.org/wiki/.NET_Framework
8. SecondLife: Mono - Second Life Wiki.(acceso al 05/2008) https://wiki.secondlife.com/wiki/Mono
9. The Gtk+ Project: What is Gtk+? (acceso al 05/2008) http://www.gtk.org/
10. The Gtk+ Project: Commerce. (acceso al 05/2008) http://www.gtk.org/commerce.html
11. The Gtk+ Project: Features. (acceso al 05/2008) http://www.gtk.org/features.html
12. Blog de Milton Pividori "il libero". Gtk# - Aplicaciones Sensibles. (acceso al 05/2008) http://www.miltonpividori.com.ar/2007/12/06/gtk-aplicaciones-sensibles
13. MSDN - Manual del programador de .NET Framework - Arquitectura de .NET Framework Remoting.
14. La Cara Oculta de C# - .NET Remoting - Ian Marteens.
15. Programming Web Services with SOAP. Tidwell D., Snell J., Kulchenko P. O'Reilly. 2001.
16. Web service - Wikipedia, the free encyclopedia (acceso al 05/2008) http://en.wikipedia.org/wiki/Web_service
17. Design Patterns: Elements of Reusable Object-Oriented Software. Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Addison-Wesley. ISBN 0-201-63361-2.
18. Stateless server - Wikipedia, the free encyclopedia (acceso al 05/2008) http://en.wikipedia.org/wiki/Stateless_server
19. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1 (acceso al 05/2008) http://www.faqs.org/rfcs/rfc2616.html
20. W3C Workshop on Push Technology. September 8-9, 1997. Boston - USA (acceso al 05/2008) http://www.w3.org/Architecture/9709_Workshop/

21. NuSOAP - SOAP Toolkit for PHP (acceso al 05/2008)
http://sourceforge.net/projects/nusoap/