

Introducción a WCF con GTK# y MonoDevelop

por Martín A. O Márquez <xomalli@gmail.com>

Windows Communication Foundation (WCF) es un framework que soporta aplicaciones orientadas a servicios con herramientas que facilitan la construcción y el consumo de servicios independientes de la plataforma, proporciona un modelo unificado de programación para aplicaciones distribuidas con tecnologías como: Web Services, Remoting, COM, DCOM, WSE, MSMQ, etc.

Este modelo está enfocado a desarrollar servicios orientados a procesos de negocio que los clientes pueden acceder y utilizar sin conocer los detalles de su implementación. Los beneficios de las aplicaciones orientadas a servicios son:

1. Los servicios independientes actúan como bloques de construcción que pueden reutilizarse para la construcción de nuevas aplicaciones o servicios.
2. Las aplicaciones en este contexto están totalmente desacopladas de los procesos de negocio y se convierten únicamente en interfaces de usuario (UI) que hacen uso de los servicios, además pueden o no estar construidas con las mismas herramientas de programación que el servicio.
3. El éxito depende más de los procesos de negocio que de la tecnología.
4. Los servicios son un grupo de métodos que comparten funcionalidades, esto permite que las aplicaciones respondan a los requerimientos cambiantes sin que se desarrollen desde cero esos requerimientos.

En este contexto WCF tiene dos niveles:

1. A nivel lenguaje de programación se ve como un sistema compuesto por objetos persistentes, reglas de negocio que pueden estar en objetos de .NET o en store procedures y una interfaz de negocio que expresa las operaciones del servicio en donde se revisan las precondiciones de cada operación, se ejecutan las actividades del negocio y se regresa un resultado para el consumidor del servicio.
2. A nivel servicio se tienen operaciones que pueden ser similares a las funciones a nivel lenguaje pero que son diseñadas para ser parte del servicio. Estas operaciones combinan varias funciones a nivel lenguaje y probablemente utilicen tipos de datos más acorde con ambientes distribuidos.

WCF proporciona funcionalidad a una amplia audiencia de clientes distribuidos que no comparten un mismo espacio de direcciones, estos clientes utilizan el servicio mediante una clase proxy. Los clientes y los servicios se comunican intercambiando mensajes que no están limitados a un conjunto particular de protocolos.

Las aplicaciones orientadas a servicios es un concepto relativo al estilo de la aplicación y a la granularidad de los servicios.

Toda la información necesaria para los clientes: qué es lo que el servicio hace, como debe ser accedido, en qué lugar está disponible, etc. WCF lo encapsula en un concepto llamado **EndPoint**, que es básicamente una combinación de address, binding y contract (lo que comúnmente se conoce como el ABC de WCF).

Un servicio WCF consta de los siguientes elementos:

1. Un service host que proporciona el runtime para activar el servicio Web, los hay en tres tipos: Internet Information Services (IIS), Windows Activation Services (WAS) and selfhost managed applications.
2. El contrato del servicio que es una interfaz a nivel lenguaje de programación que define las operaciones que serán expuestas a través del endpoint.
3. Clases ordinarias o componentes de .NET que implementan toda la lógica de negocios.

Hay tres componentes básicos para la creación de un servicio en WCF.

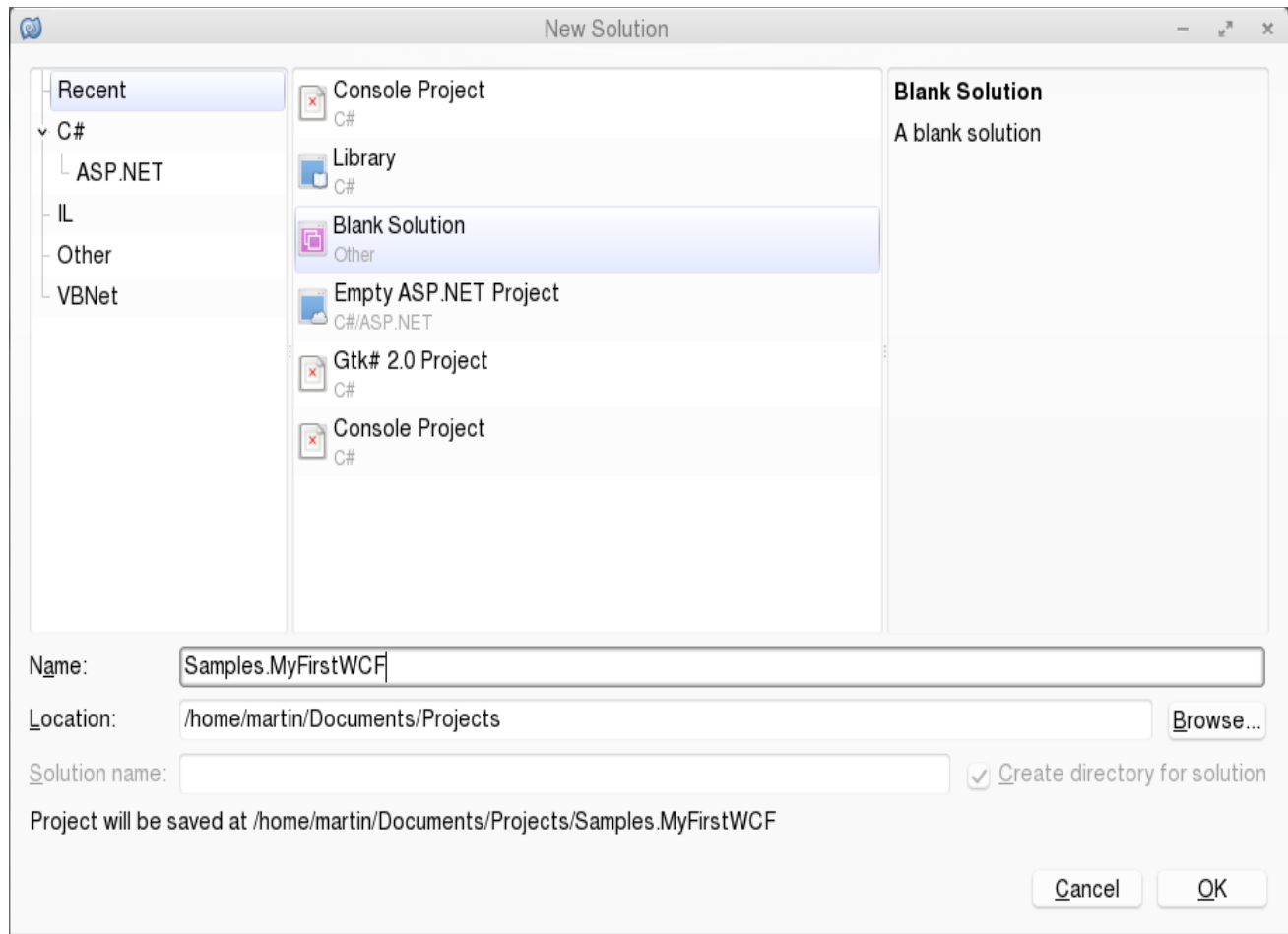
- (a) El servicio WCF
- (b) El service host
- (c) El cliente

Como ejemplo, para la creación de cada uno de estos componentes, voy a programar una aplicación GTK# que recibe un número entero, manda la petición al servicio y finalmente recibe su representación binaria como una cadena.

Tarea 1: Creación del servicio WCF

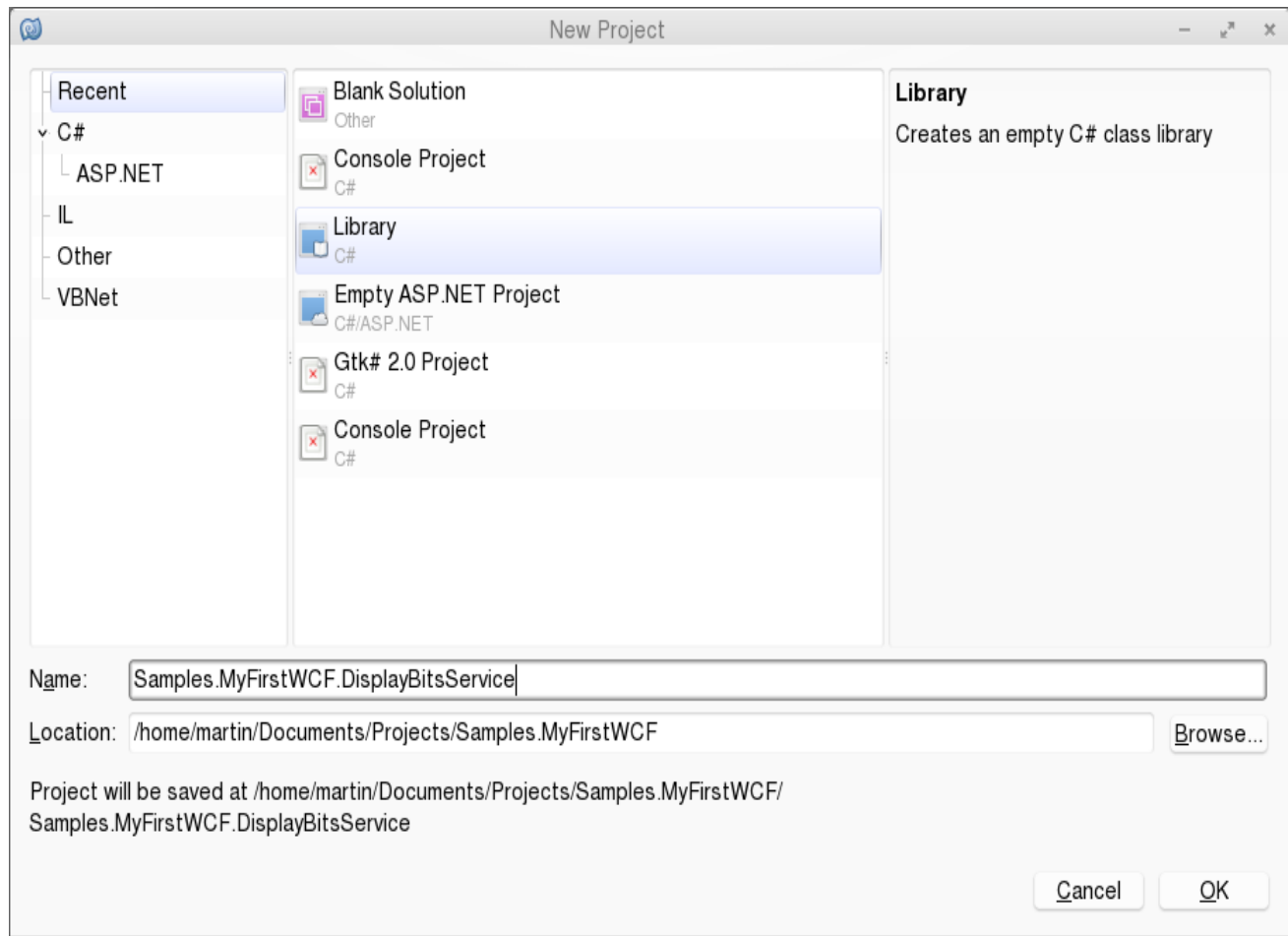
1-. Ejecuta MonoDevelop y crea una nueva solución del tipo “**Blank Solution**” con el nombre “**Samples.MyFirstWCF**”.

Fig 1 Seleccionando el tipo de proyecto



2-.A la solución “**Samples.MyFirstWCF**” agrega un proyecto del tipo “**Library**” con el nombre “**Samples.MyFirstWCF.DisplayBitsService**”.

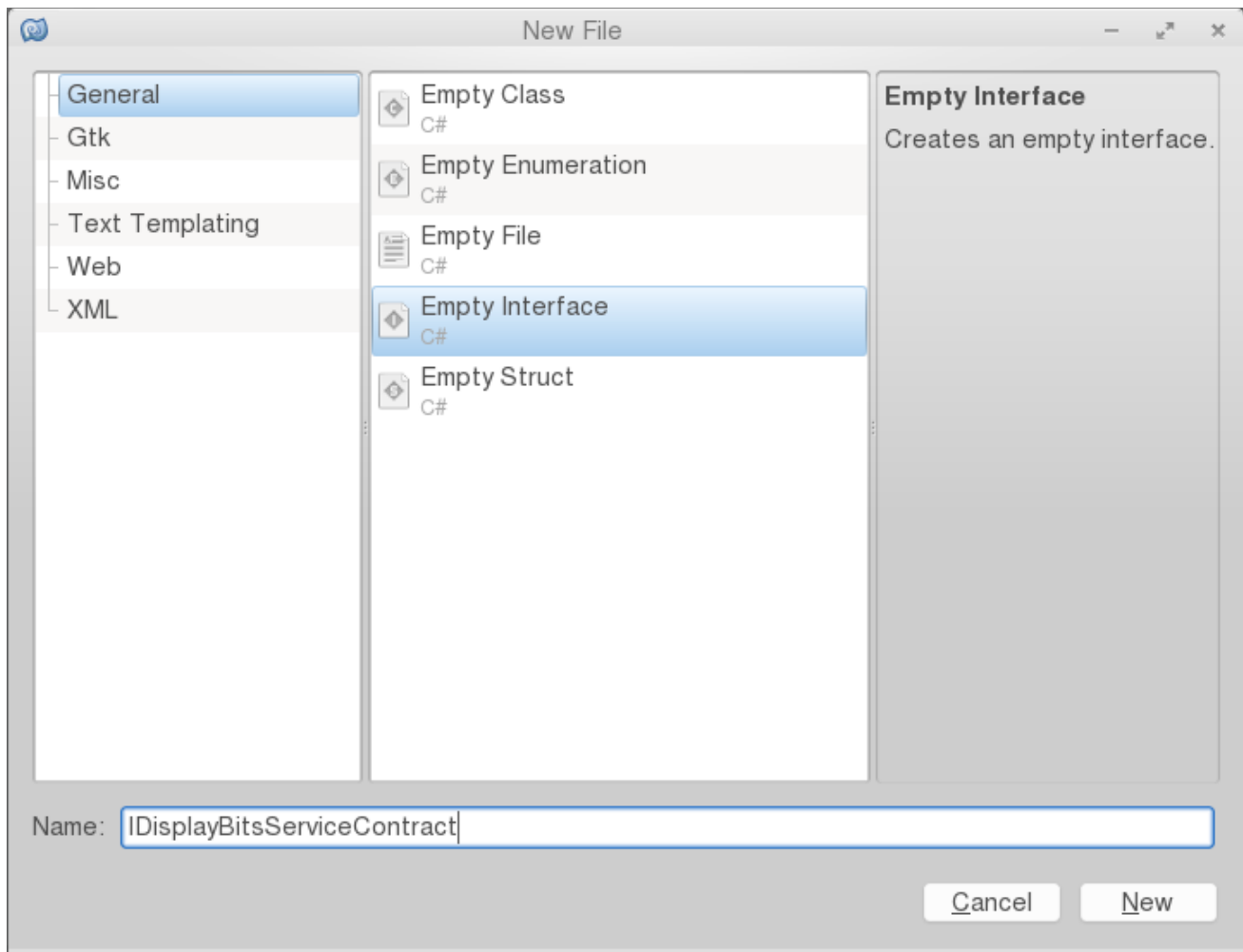
Fig 2 Agregar un proyecto de tipo library



3-. Al proyecto “**Samples.MyFirstWCF.DisplayBitsService**” agrega los siguientes elementos:

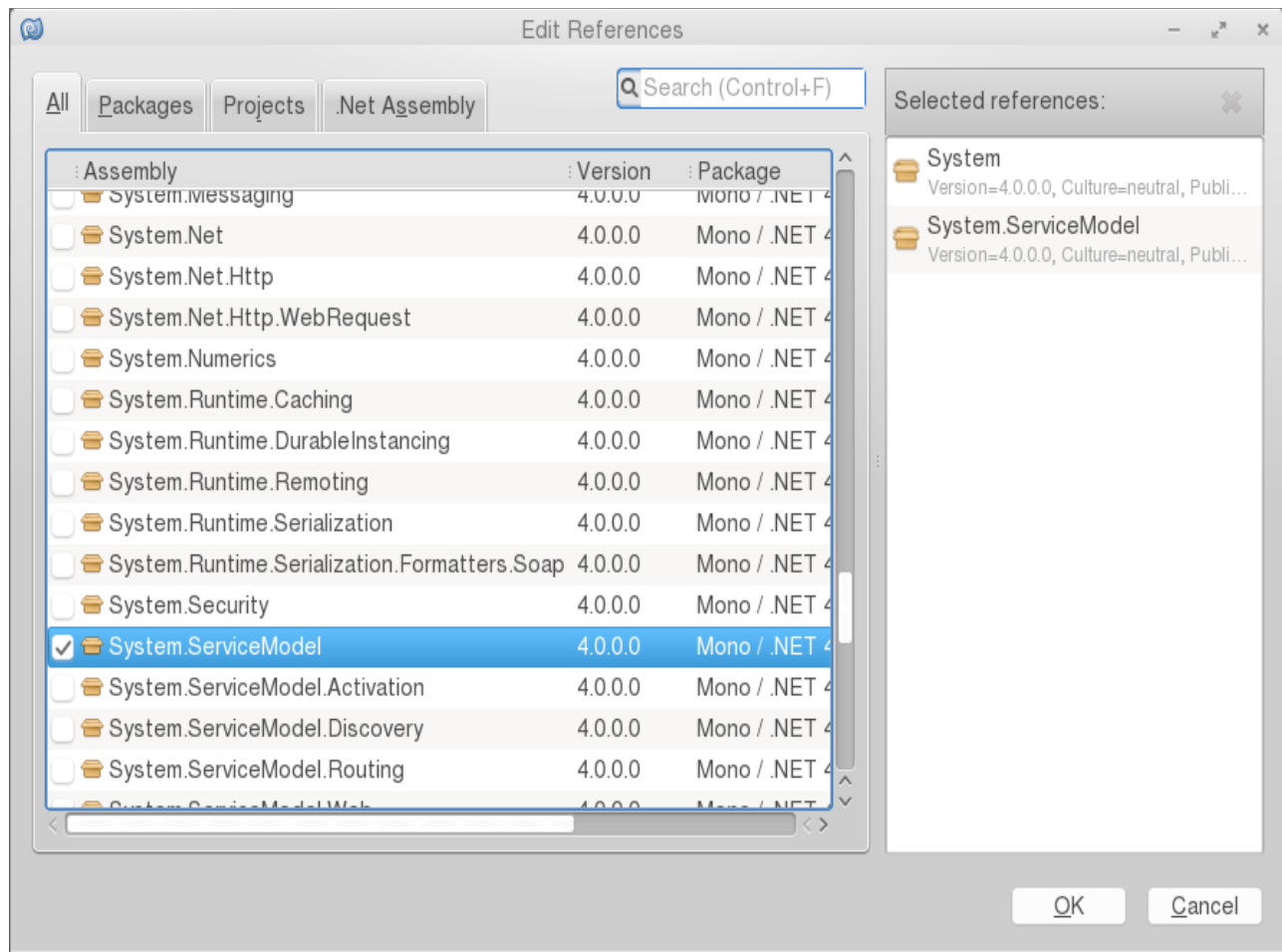
1. Una interface con nombre ***IDisplayBitsServiceContract***
2. Una clase con nombre ***DisplayBitsServiceImplementation***

Fig 3 Agregando la interfaz para el servicio



4-. Antes de escribir el código es importante agregar al proyecto la referencia al ensamblado **System.ServiceModel**

Fig 5 Agregando la referencia a los ensamblados para trabajar con WCF



5- Escribir el siguiente código para la interfaz ***IDisplayBitsServiceContract***:

```
using System;
using System.ServiceModel;

namespace Samples.MyFirstWCF.DisplayBitsService
{
    [ServiceContract]
    public interface IDisplayBitsServiceContract
    {
        [OperationContract]
        string DisplayToBits (int i);
    }
}
```

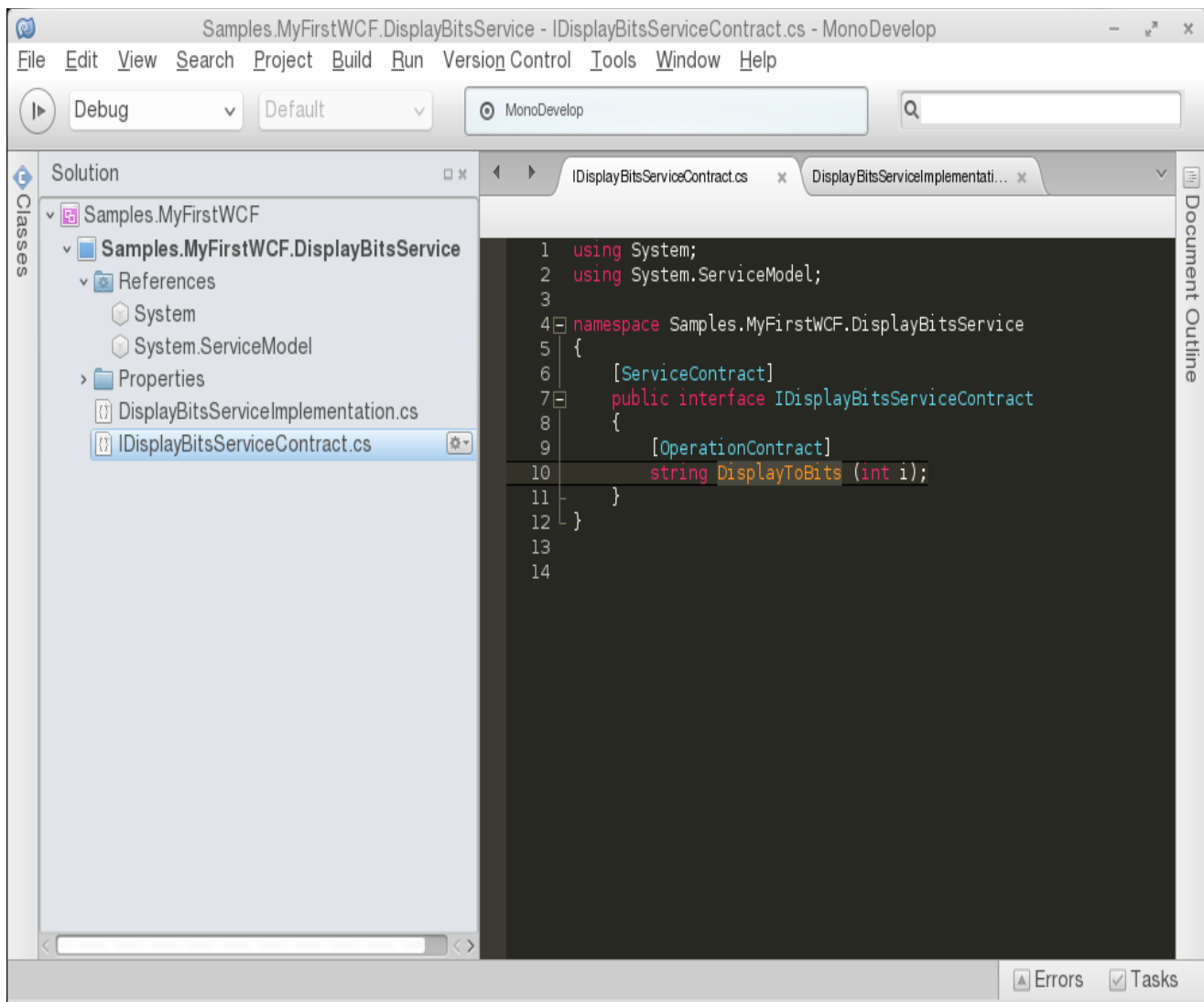
6-. Escribir el siguiente código para la clase ***DisplayBitsServiceImplementation***:

```
using System;
using System.Text;

namespace Samples.MyFirstWCF.DisplayBitsService
{
    public class DisplayBitsServiceImplementation : IDisplayBitsServiceContract
    {
        public string DisplayToBits (int i)
        {
            int mask = 1 << 31;
            StringBuilder buf = new StringBuilder (35);
            for (int bit = 1; bit <= 32; bit++)
            {
                buf.Append((i & mask) == 0 ? '0' : '1');
                i <<= 1;
                if(bit % 8 == 0)
                    buf.Append(' ');
            }
            return buf.ToString();
        }
    }
}
```

La solución debe de verse como en la siguiente imagen:

Fig 6 La solución con las clases del servicio WCF



Tarea 2: Creación del programa Host

1-. Agrega un nuevo proyecto a la solución **Samples.MyFirstWCF** del tipo **Console Project** con el nombre de **Samples.MyFirstWCF.DisplayBitsSelfHost**.

Fig 7 El proyecto para la aplicación service host

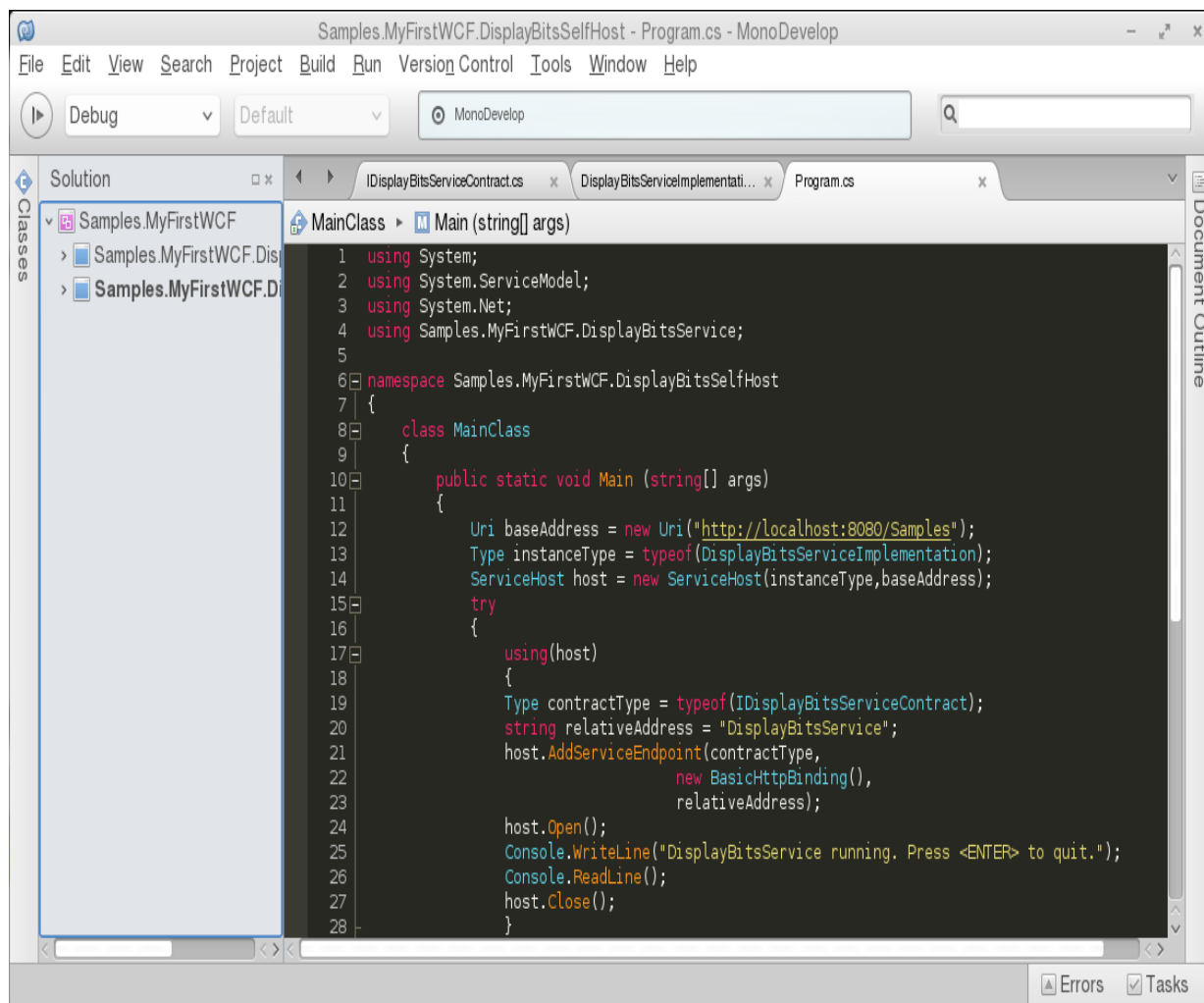

```

        new BasicHttpBinding(),
        relativeAddress);
    host.Open();
    Console.WriteLine("DisplayBitsService running. Press to quit.");
    Console.ReadLine();
    host.Close();
}
} catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
}
}

```

La solución debe de verse como en la siguiente imagen:

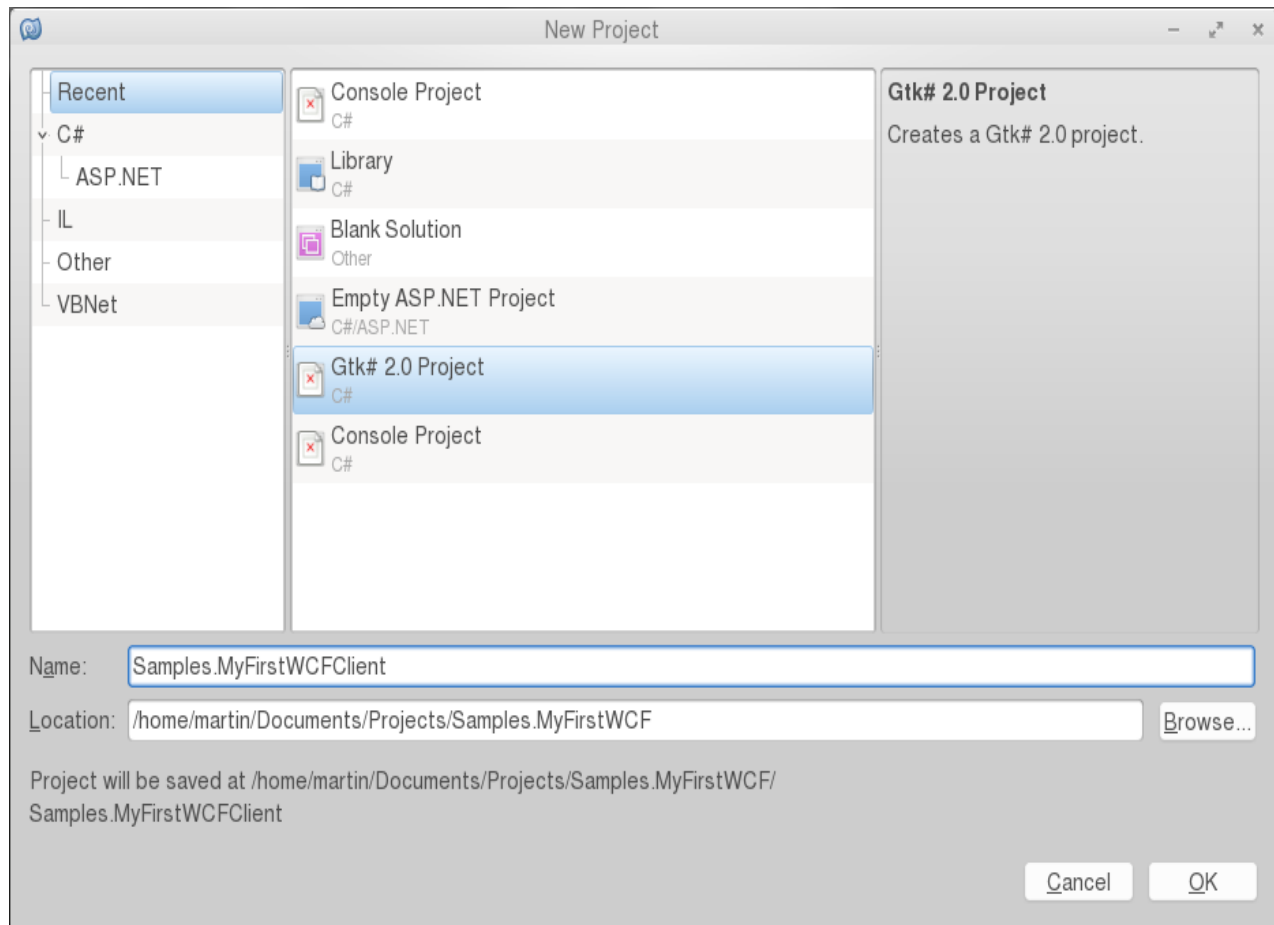
Fig 8 La solución completa para el service host



Paso 3: Creación del cliente GTK#

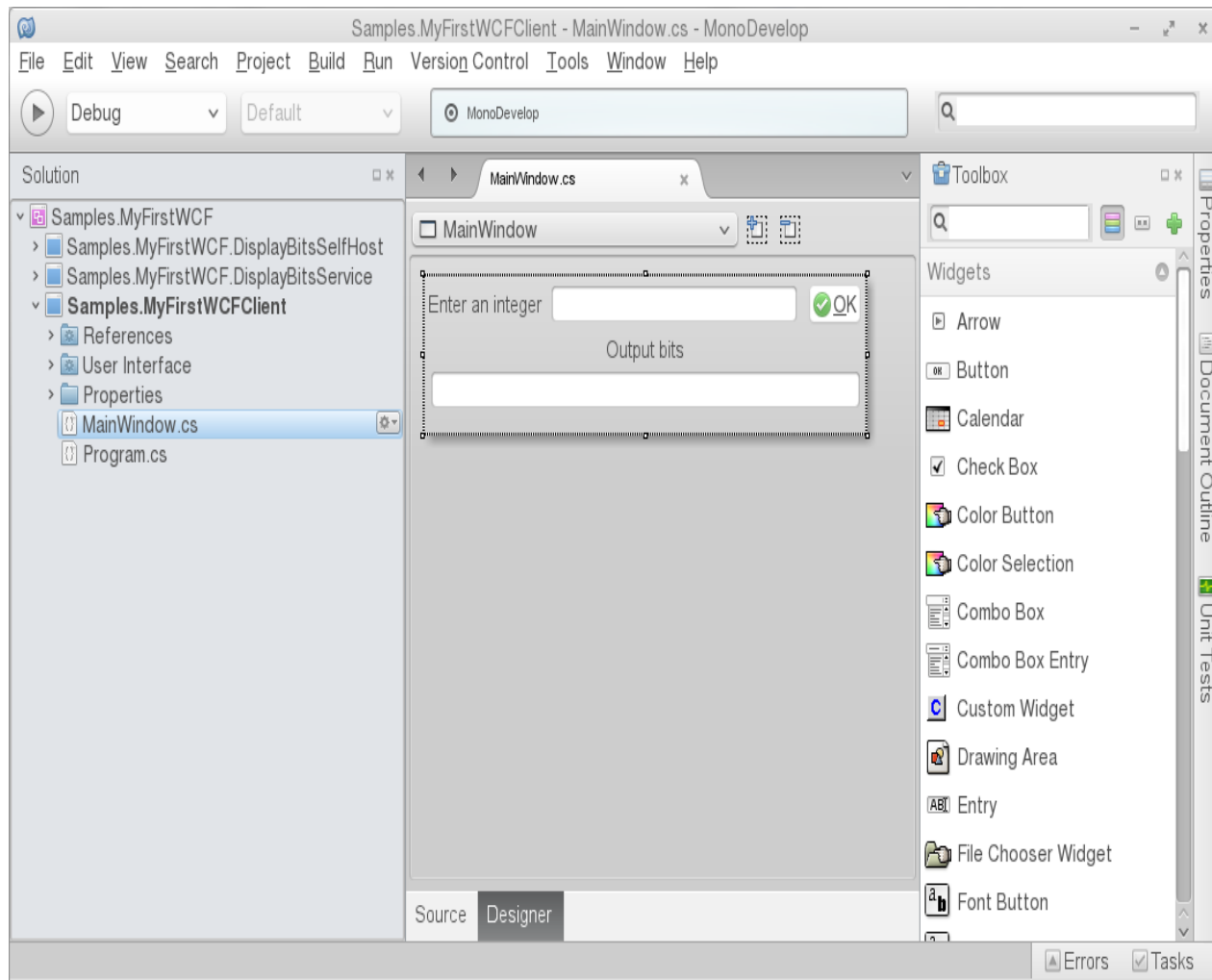
1-. Agrega un nuevo proyecto del tipo **GTK# 2.0 Project** con el nombre **Samples.MyFirstWCFClient**.

Fig 9 El proyecto cliente en GTK#



2-. Utilizando el diseñador de MonoDevelop creamos una interfaz gráfica como se muestra en la siguiente imagen:

Fig 10 El proyecto cliente en GTK#

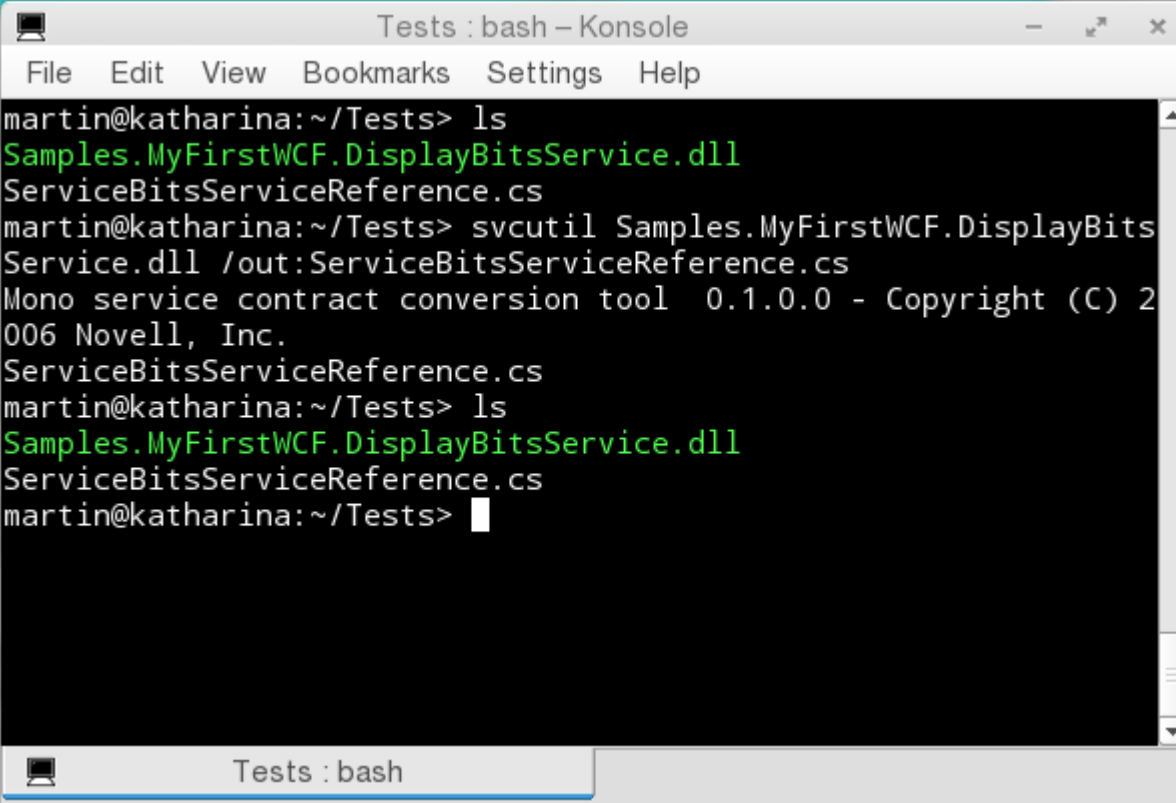


3-. Una aplicación cliente de WCF puede comunicarse con un servicio WCF utilizando una clase proxy. Para generar el código de esta clase, se utiliza la herramienta **svcutil (ServiceModel Metadata Utility Tool)**.

4-. Abrimos una terminal y tecleamos el comando **svcutil** utilizando como argumento el ensamblado del servicio y la opción **/out** para ponerle nombre al archivo y no utilizar el predeterminado.

```
$ svcutil Samples.MyFirstWCF.DisplayBitsService.dll  
/out:ServiceBitsServiceReference.cs
```

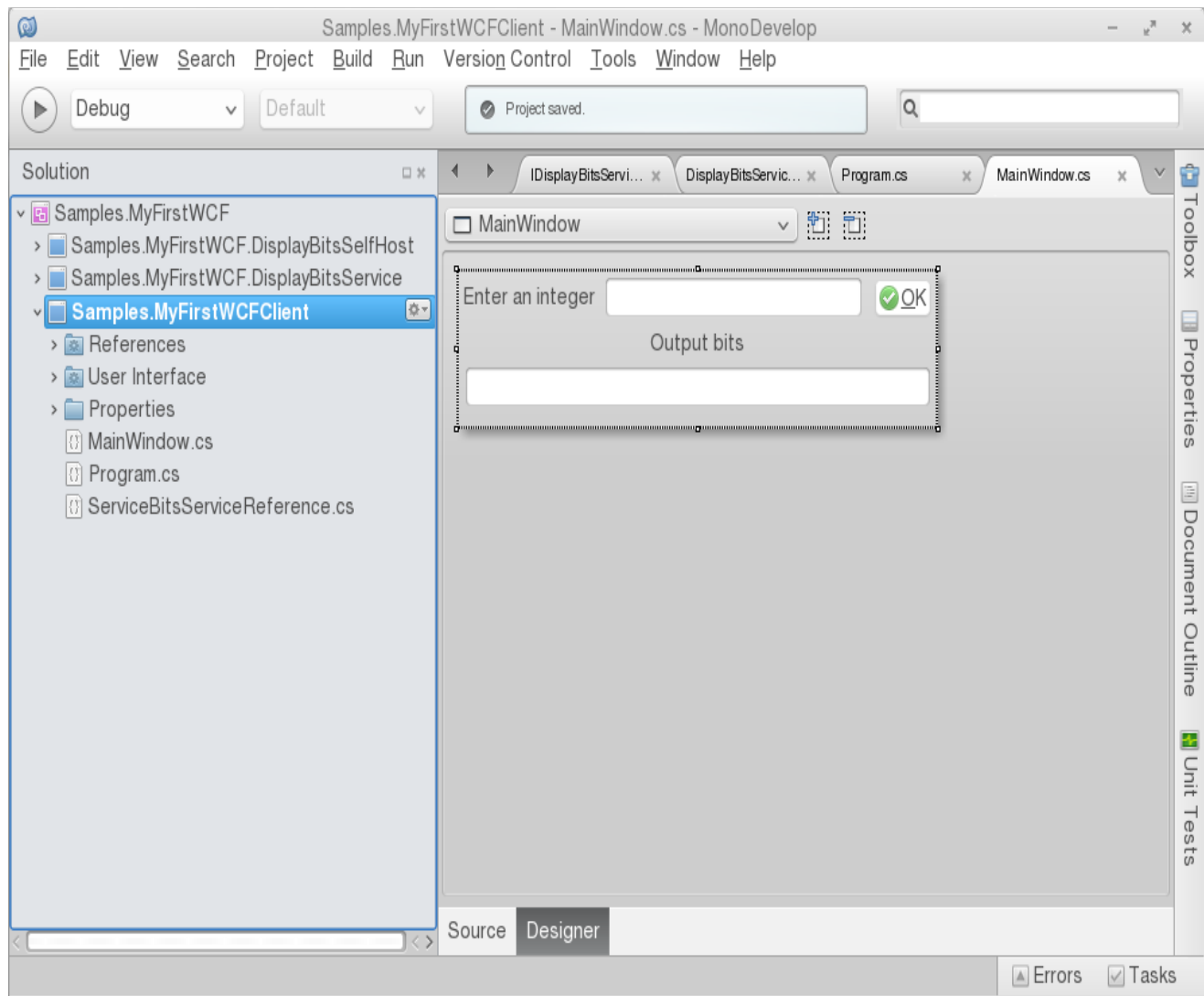
Fig 11 Creando la clase proxy



```
Tests : bash - Konsole
File Edit View Bookmarks Settings Help
martin@katharina:~/Tests> ls
Samples.MyFirstWCF.DisplayBitsService.dll
ServiceBitsServiceReference.cs
martin@katharina:~/Tests> svcutil Samples.MyFirstWCF.DisplayBits
Service.dll /out:ServiceBitsServiceReference.cs
Mono service contract conversion tool 0.1.0.0 - Copyright (C) 2
006 Novell, Inc.
ServiceBitsServiceReference.cs
martin@katharina:~/Tests> ls
Samples.MyFirstWCF.DisplayBitsService.dll
ServiceBitsServiceReference.cs
martin@katharina:~/Tests> 
```

5-. Una vez generada la clase proxy, la agregamos a la solución **GTK#** para que la aplicación pueda invocar los métodos del servicio.

Fig 12 La clase proxy dentro de la solución

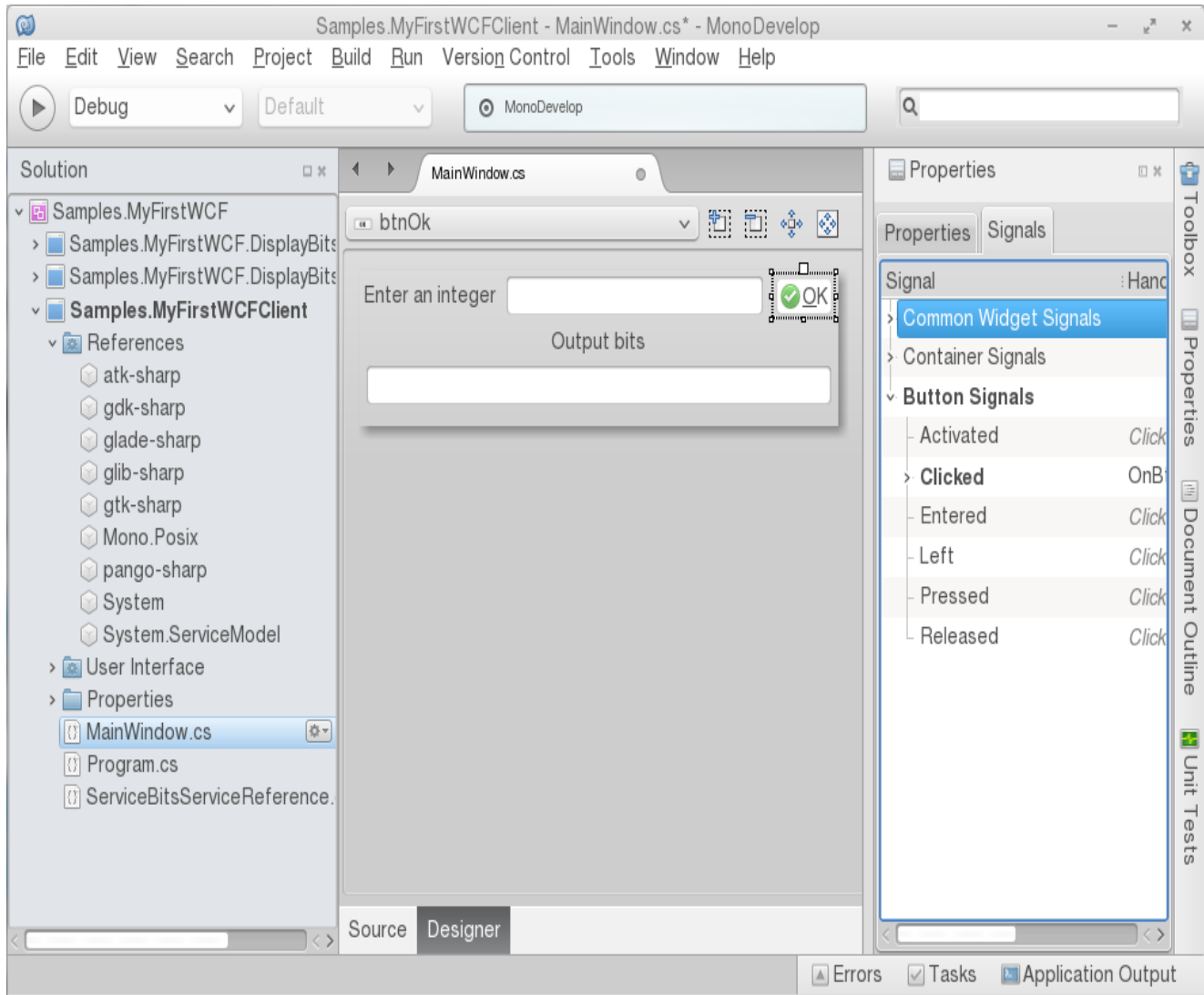


La clase proxy implementa un channel stack del lado del cliente. Todas las respuestas recibidas desde el servicio pasan a través de este stack, por lo que para comunicarse el cliente y el servicio deben utilizar un stack y una configuración equivalente.

6-. Antes de compilar es importante agregar una referencia al ensamblado **System.ServiceModel**.

7-. Ahora vamos a generar el código para utilizar el proxy, en la ventana **Properties**, busca dentro de la categoría **Button Signals** un evento llamado **clicked**, haz doble-click o pulsa enter para que MonoDevelop genere la plantilla para el evento.

Fig 13 Generando el evento del boton para invocar el servicio



8-. Agrega el siguiente código para completar el método.

```
protected void OnBtnOkClicked (object sender, EventArgs e)
{
    var proxy = GetProxy ();
    int intToDisplay = 0;
    if (!string.IsNullOrEmpty (txtInteger.Text))
    {
        intToDisplay = Convert.ToInt32 (txtInteger.Text);
        txtOutput.Text = proxy.DisplayToBits (intToDisplay);
    }
}
```

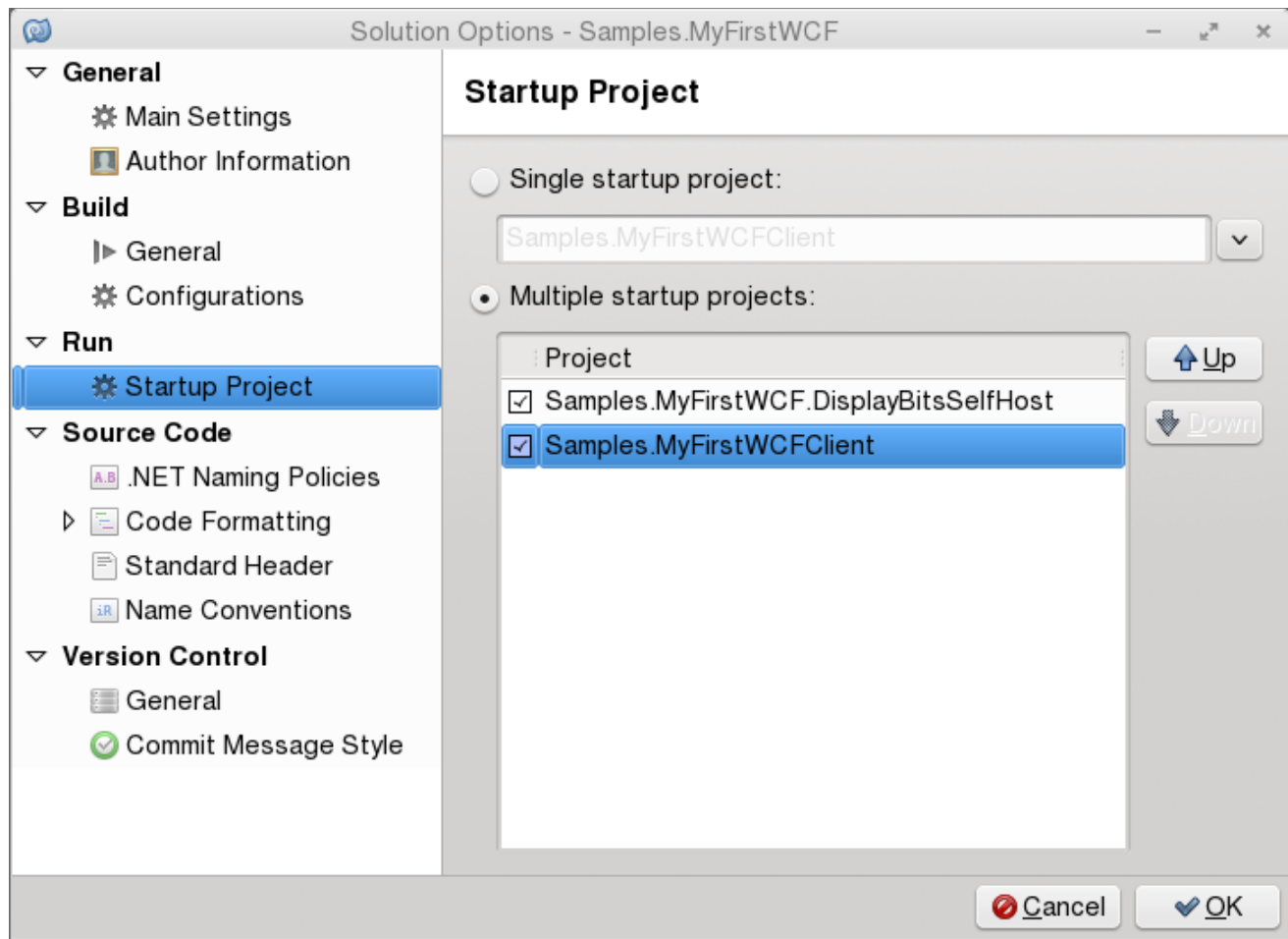
```
IDisplayBitsServiceContract GetProxy(){
    EndpointAddress address =
    new EndpointAddress ("http://localhost:8080/Samples/DisplayBitsService");
    BasicHttpBinding binding = new BasicHttpBinding ();
    IDisplayBitsServiceContract proxy =
    ChannelFactory.CreateChannel (binding,address);
}
```

```
return proxy;
}
```

9- En el panel ***Solution explorer***, haz click derecho en la solución ***Samples.MyFirstWCF***, y entonces en el menú ***Options***.

10- Configura la solución para que los proyectos ***Samples.MyFirstWCF.DisplayBitsSelfHost*** y ***Samples.MyFirstWCFClient*** empiecen cuando se ejecute la solución.

Fig 14 Estableciendo la ejecución simultanea de ambos proyectos.



11-. Compila y ejecuta la solución, si todo está bien se verán las siguientes imágenes.

Fig 15 Una primera ejecución del cliente GTK#

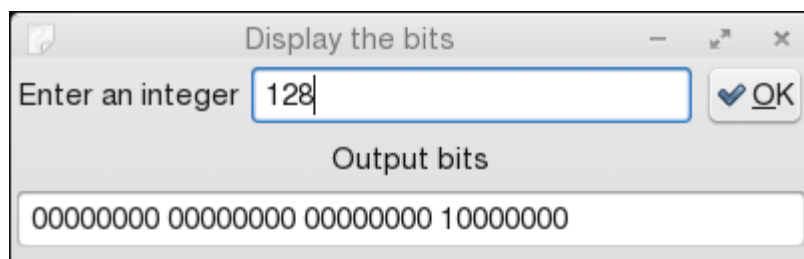
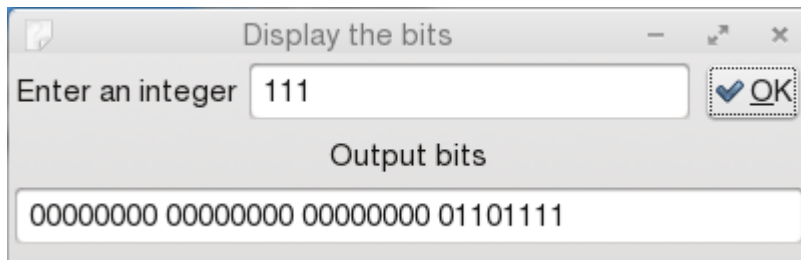
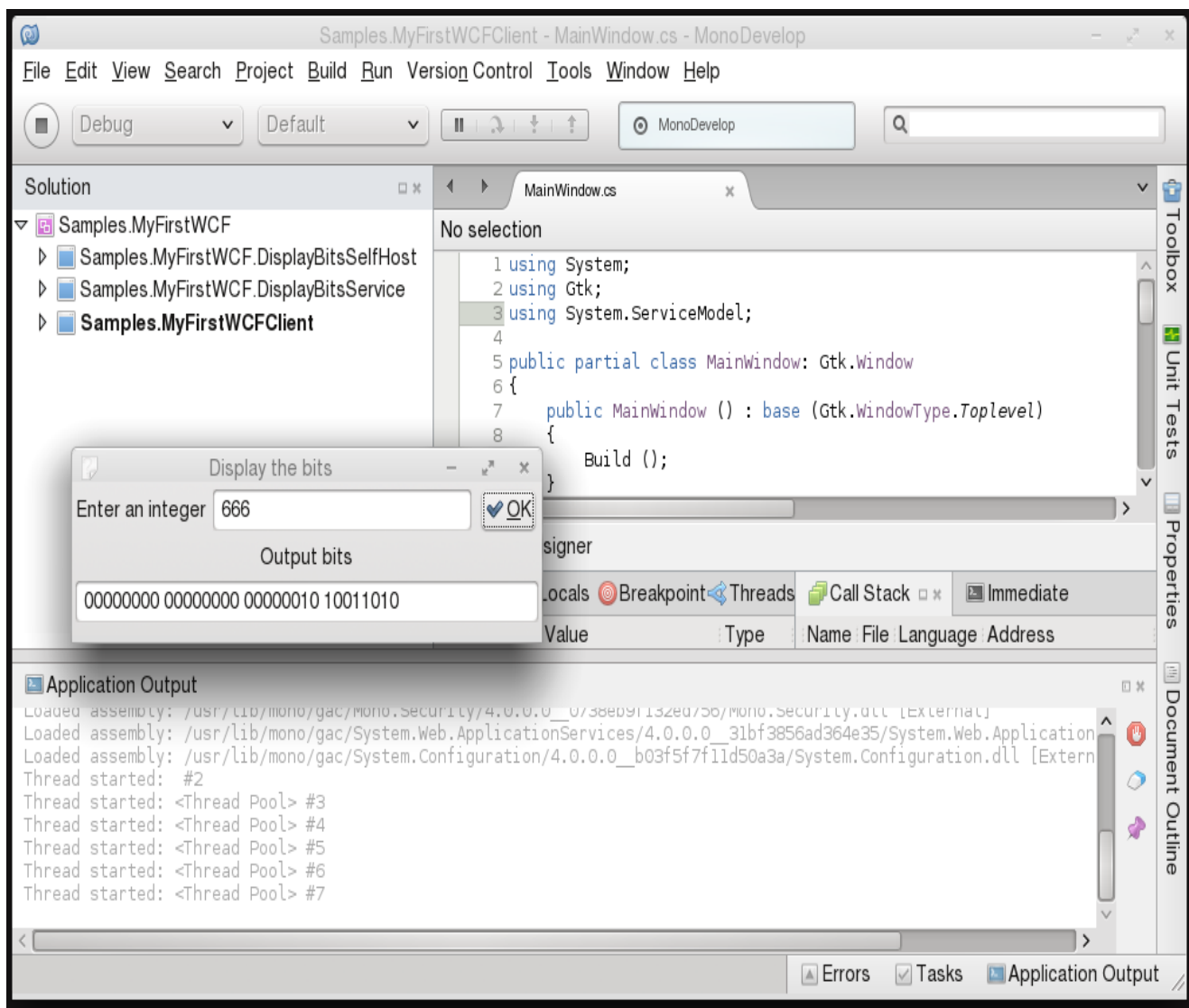


Fig 16 Probando el cliente con otro valor



Al presionar el botón el cliente envía la petición al programa host y este le regresa el resultado correcto.

Fig 17 Probando la petición del cliente y la respuesta del servicio.





Download el código fuente para Xamarin Studio o Visual Studio

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»