# Framework para el diseño de servicios orientados a procesos

***Framework for the design of processes oriented services***

**Adonis Cesar Legón Campo, Jorge Landrián García**
Universidad de las Ciencias Informáticas (UCI)
{alegon, jlandrian}@uci.cu

**Resumen**

En el presente trabajo se muestra el diseño e implementación de un *framework* que describe el desarrollo de servicios orientados a unidades de procesos que los componen y que permite extender sus funcionalidades según los requerimientos definidos para cada caso. Surge de la necesidad de lograr mecanismos de integración e intercambio de información entre sistemas, además de otras aplicaciones como la notificación de mensajes a personas o entidad ante eventos o cambios de estados en los procesos que se desarrollan en determinado sistema. Estos servicios se proponen para la tecnología de servicios NT de Microsoft Windows, aunque el *framework* es independiente de la plataforma y solo describe la base fundamental para el desarrollo de servicios y sus procesos, permitiendo su uso o implementación sobre otras tecnologías de desarrollo y Sistemas Operativos.

**Palabras clave**: extensible, *framework*, independiente de la plataforma, proceso, servicio.

***Abstract***

*This work shows the design and implementation, of a framework that describes the development of services oriented to processes that composes them, and allows extending its functionalities as defined on the requirements for each one. It emerge due to the need to make integration and information interchange mechanisms between systems, and some other applications like message notification to persons or entities of some state changes or events that can occur on the processes defined in an specific system. These services are proposed for the Microsoft Windows NT service technology, although this framework is platform independent, and it only describe the fundamentals for the services development and its processes, allowing the implementation to be used in other development technology and Operative Systems.*

**Keywords*:*** extensible, framework, platform independent, process, service.

## Introducción

Con frecuencia muchas entidades u organizaciones necesitan de un soporte para obtener o intercambiar información con otras entidades como parte de sus procesos, además de informar y notificar a las personas, entidades o instituciones, de determinados eventos o cambios en el estado de determinada información, ocurridos en los procesos que se desarrollan, con el objetivo de brindar un servicio mas eficiente.

Partiendo de este análisis surge la implementación de un mecanismo que permita brindar soporte a estos requisitos, fácil de mantener y configurar, de gestión y control centralizados. La solución se basó en el desarrollo distintos

servicios que posibilitarían una mayor integración con entidades externas y la capacidad de ofrecer mayor información a las personas, además de mejorar los procesos internos de los sistemas para lograr mejores resultados en su funcionamiento.

Para la implementación de estos servicios, primeramente se partió del diseño de un *framework*, que permitiera la creación, mantenimiento y mejora de nuevos servicios, que de forma general podrían ser desarrollados para cumplir un objetivo específico dentro de los procesos que se definan en la entidad. El diseño de este *framework* debía cumplir un conjunto de requerimientos necesarios para soportar los objetivos por los que sería creado, los cuales se describen a continuación:

**Multiprocesamiento:** los servicios deben ser capaces de realizar varias operaciones por separado o con determinado nivel de dependencia, en un ambiente de multiprocesamiento paralelo, para lograr que cada uno contenga más cantidad de funcionalidades dentro de los procesos para los que serán definidos.

**Transporte de mensajes:** se debe soportar el transporte de mensajes a través de un mecanismo que permita hacer transferencias de datos e información por distintos medios como bases de datos, servidores de correo, de FTP (*File Transfer Protocol*) [1], de aplicaciones, entre otros que pudieran surgir como necesidades posteriores, para de esta forma poder aislar los mecanismos de transporte implícitos en estos procesos.

**Log de errores:** esta es una característica que deben tener todos los servicios debido a que, al ser aplicaciones corriendo en el fondo o "*background*", no utilizan generalmente mecanismos de notificación gráfica, por lo que debe ser capaz de persistir todos los errores que ocurran en los distintos procesos que se desarrollen en su funcionamiento, ya sea por archivos "logs" o por alertas del sistema.

**Extensible:** debe brindar la posibilidad de agregarles nuevos procesos o terminales de comunicación para transferencia de mensajes entre los distintos medios disponibles, siempre usando un mecanismo o metodología base que oriente el desarrollo para facilitar el proceso de implementación, integración y despliegue. Este *framework* debe contener las clases bases para el desarrollo y orquestación de nuevos servicios.

**Configurable:** los servicios deben ser totalmente configurables tanto de forma general por cada proceso como por cada componente de transferencia de mensajes, de esta forma permitirle a los administradores del sistema realizar cambios en cualquiera de los procesos que lo componen.

**Descripción general de un servicio Windows NT**

Los servicios sobre la plataforma Windows o Windows Services, conocidos anteriormente como servicios de la tecnología Windows NT o NT Services, permiten crear aplicaciones que corran a tiempo completo en su propia sesión de Windows. Estos servicios exponen una interfaz a través del sistema operativo por la cual pueden ser

ejecutados automáticamente cuando el sistema inicia, o manualmente detenidos y reiniciados por el propio usuario administrador [2].

Con estas características, los servicios son la solución ideal sobre los Sistemas Operativos, para el desarrollo de aplicaciones que corran constantemente sin interferir con los usuarios que trabajan en la misma estación, ya que se ejecutan en el "fondo" o "*background*" y generalmente no utilizan interfaz gráfica con la que el usuario pueda interactuar.

El *framework* se denominó "**QuarkFramework"**, y está implementado usando la tecnología Microsoft.Net Framework v1.1.4, en el lenguaje c# y se usó el entorno de desarrollo Microsoft Visual Studio .Net 2003.

**Diseño**

En el desarrollo se definieron un conjunto de clases bases para facilitar la implementación de los distintos procesos que se realizan en un servicio y de esa forma usar funcionalidades comunes y definir ciertas métricas y metodologías para que sea mas sencillo detectar errores y probar cualquiera de sus componentes además de extender sus funcionalidades tanto en los casos específicos como en los comunes. El siguiente diagrama muestra las principales clases de los procesos definidos en el *framework*:

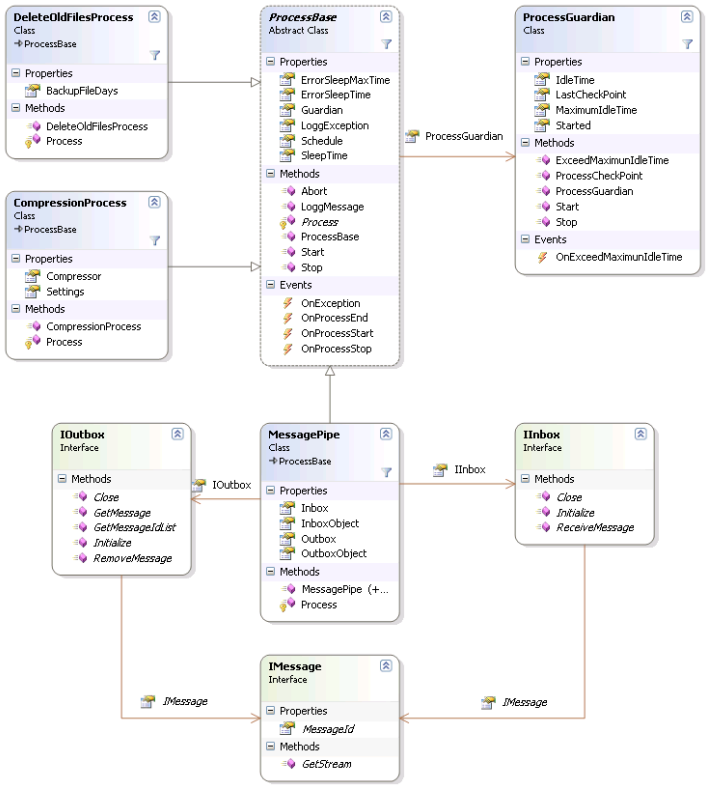


Fig. 2. Diagrama de clases del diseño.

**ProcessBase**

La clase **ProcessBase** es la unidad principal de procesamiento dentro de este *framework* de servicios. Es la encargada de gestionar le ejecución de un proceso, controlando aspectos como: la ejecución de procesamiento paralelo con el uso de la técnica de los "hilos", la gestión del proceso en ciclos con intervalos de espera e incluso con "*Schedule*" o programación de tiempo en operación, el registro de los errores ocurridos en archivos "log" y la notificación de estas excepciones internas y de los cambios de estado en el proceso. Las propiedades que definen este proceso base son heredadas en todos los tipos de procesos que se puedan crear, y se describen de la siguiente forma:

**SleepTime**: tiempo en milisegundos en el que está inactivo el proceso hasta volver a ejecutarse.

**ErrorSleepTime**: tiempo en milisegundos en que el proceso se "duerme" ante la ocurrencia de un error, este tiempo se va incrementando en la medida que sigan ocurriendo errores.

**ErrorSleepMaxTime**: tiempo máximo en milisegundos en que el proceso puede estar "durmiendo".

**Guardian**: define si este proceso va a usar un "guardián" para su funcionamiento.

**Schedule**: define una programación para la ejecución del proceso a través de un horario de inicio "**InitTime**" y un horario de fin "**EndTime**".

**LogException**: si se desea hacer "log" de los errores que ocurren durante el proceso.

Para crear un nuevo tipo de proceso solo es necesario heredar de esta clase e implementar el método "**Process**", el cual es ejecutado cuando este proceso comience a correr en el servicio, e implementar dentro las operaciones que debe realizar según su definición cuando fue concebido. En la biblioteca de clases contenidas en este *framework*, se han creando algunos tipos de proceso que realizan operaciones específicas, implementando el método anteriormente descrito:

**CompressionProcess:** es un proceso capaz de comprimir todos los archivos que se encuentren en una ubicación determinada hacia una ubicación de destino donde se copiarán los archivos ya comprimidos luego de, al ser empaquetados por grupo, sobrepasen determinado tamaño también definido por configuración.

**DeleteOldFilesProcess:** este proceso se encarga de eliminar los archivos temporales de algún otro proceso partiendo de la fecha de la creación del fichero y teniendo en cuanta la cantidad de días que pueden durar según se defina por configuración.

**MessagePipe**

Es un tipo de proceso muy utilizado en los servicios ya desarrollados. Como su nombre lo indica es una "tubería" de mensajes que permite la transferencia de datos empaquetados en mensajes entre dos terminales lógicos: el "**Inbox**" o bandeja de entrada y el "**OutBox**" o bandeja de salida. Esta tubería realiza un proceso descrito de la siguiente forma:

- El proceso ordena pedir la lista de identificadores de los mensajes, a la bandeja de salida o "**OutBox**", que estén listos para enviar, a través del "**GetMessageIdList**".
- Luego por cada "**Id**" o identificador de mensaje obtenido se le pide el mensaje a la bandeja de salida con el método "**GetMessage**", el cual debe contener toda la información y datos que serán procesados por la bandeja de entrada.
- Este mensaje se le pasa a la bandeja de entrada o "**InBox**" usando el método "**RecieveMessage**" para que esta haga el procesamiento necesario con el mensaje recibido.
- Por último el mensaje es eliminado de la bandeja de salida con el método "**RemoveMessage**".

En caso de ocurrir algún error en el proceso de envío de los mensajes existe un mecanismo para lograr que, o bien se termine el proceso de envío o solo se salte el mensaje que dio error y se puede continuar con el siguiente. Esto se puede lograr usando un tipo de excepción llamada "**ProcessException**", la cual será lanzada en cualquier de lo métodos descritos anteriormente de las bandejas de entrada o de salida, teniendo en cuenta al implementarlos que: si se quiere que el proceso de envío de mensajes no sea abortado por completo, se captura la "**Excepción**" ocurrida, y se lanza una instancia de la clase "**ProcessException**", con la "**Excepción**" capturada, un mensaje describiendo el error ocurrido y estableciendo en "**true**" la propiedad "**Handled**" de este objeto. De esta forma se podrá loguear el problema y continuar al siguiente mensaje, de lo contrario si se quiere que el proceso termine de enviar los mensajes, solo se lanza la excepción de tipo "**ProcessException**" o de cualquier tipo, con el mensaje del error. El siguiente esquema muestra el flujo de llamadas entre las bandejas de entrada y salida para el procesamiento de un mensaje:

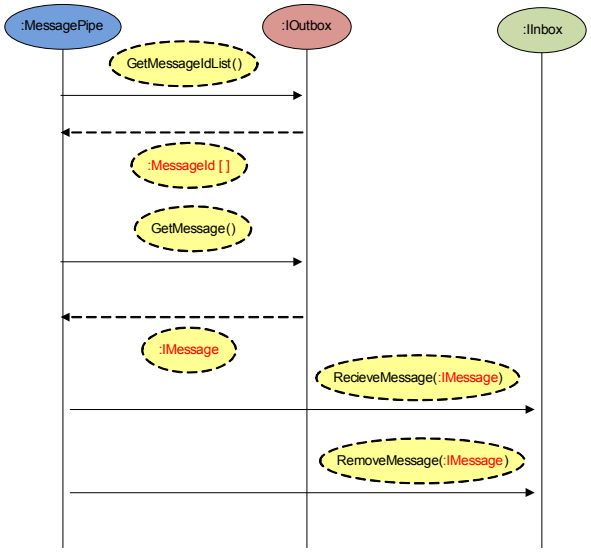


Fig. 1. Flujo de mensajes en las bandejas del MessagePipe.

**OutBox**

La bandeja de salida se implementa con la interfaz "**IOutBox**", y puede usarse para la comunicación con distintos medios de información como Bases de Datos, a través de la implementación de los métodos anteriores. Algunas bandejas de salida ya están disponibles en la biblioteca de clases.

**DirectoryOutBox**: esta bandeja de salida se encarga de devolver como mensajes, todos los archivos que se encuentra en una carpeta definida por configuración a través de la propiedad "**OutboxRoot**", permitiendo obtener los que se encuentran en subcarpetas mediante la propiedad "**IncludeSubDirectories**", además esta bandeja contiene y ejecuta un proceso de tipo "**DeleteOldFilesProcess**", para eliminar los archivos de respaldo que lleven determinado tiempo de creados.

**InBox**

La bandeja de entrada se implementa a través de la interfaz "**Inbox**", que define los métodos ya descritos, con la cual se puede usar para acceder a distintos tipos de medios de información, desde archivos en disco hasta servidores FTP y de Correo. Existen ya implementados algunos de estos "**InBox**":

**DirectoryInBox**: se encarga de crear un archivo en un directorio con el contenido del mensaje a través del método "**GetStream**", que devuelve los datos del fichero y le pone de nombre el resultado de evaluar un método escrito en lenguaje C# que se compila en tiempo de ejecución y que se establece por configuración en la propiedad "**CSharpFormatCode**" de la clase "**DynamicObjectFormatter**", este método decide qué nombre darle al archivo partiendo del propio mensaje que se le pasa como parámetro.

**FtpInBox**: es una bandeja de entrada que permite, al recibir el mensaje, enviarlo a un servidor FTP.

**ProcessGuardian**

Este proceso se encarga de monitorear otros procesos que lo usen. Si un determinado proceso demora demasiado en la ejecución de sus operaciones, al pasar el tiempo definido como máximo en la propiedad "**MaximumIdleTime**" del guardián, este notifica que el proceso se ha demorado mas del tiempo previsto y el servicio será capaz de determinar que hacer en ese caso. Para que este guardián funcione correctamente, el proceso que lo use debe ejecutar, (cada vez que vaya a efectuar una operación que pudiera demorar cierto tiempo como por ejemplo conectarse a un servidor), el método "**ProcessCheckPoint**" del guardián, que actualiza el último momento en que se verificó el estado del proceso en ejecución.

**Diseño de servicios**

Con todas las herramientas ya descritas es posible comenzar a crear servicios basados en los procesos que lo componen como unidades de ejecución independiente, utilizando los procesos que provee este *framework* o implementando otros que tengan nuevos requerimientos, y para el caso de los procesos que requieran transporte de

datos, usar las bandejas de entrada y salida ya definidas o implementar las que cumplan con los requerimientos del servicio en específico.

Una vez desarrollados todos los procesos necesarios, se puede crear el servicio que los va a usar. Para diseñar un nuevo servicio, teniendo ya los procesos, solo es necesario crear el servicio e implementar el método "**Start**", por el cual se inicial el servicio. En este método se debe cargar la configuración del servicio que contiene la información necesaria para ubicar y cargar las configuraciones de cada tipo de proceso específico.

En las clases definidas en el *framework* existe una llamada "**ProcessConfigurationItem**" que contiene la información necesaria para cargar la configuración de un proceso específico y "**deserializarla**" o convertirla del archivo texto en formato XML que contiene la configuración del proceso a objetos tipados en un lenguaje como C#. La configuración del servicio está compuesta por una colección de objetos "**ProcessConfigurationItem**" que contienen la configuración de cada proceso del servicio, agrupados en un objeto de la clase "**ServiceConfiguracion**", y de esa forma una vez que se deserialice, se pueden obtener los objetos que representan a cada "**ProcessBase**" que componen el servicio.

**Ejemplo de cómo cargar la configuración del servicio**:

```
if(File.Exists(fconfig))
{
        ServiceConfiguration s = Serializer.XmlDeserialize(fconfig
                                        , typeof(ServiceConfiguration)) as ServiceConfiguration;
 ArrayList processList = new ArrayList();
foreach(ProcessConfigurationItem item in s.ProcessConfigurationItems)
{
        if( File.Exists(item.ProcessConfigurationFile) )
        {
                XmlSerializer ser = item.SerializerSettings.CreateXmlSerializer( item.ContainerType );
                using ( FileStream fs = new FileStream(item.ProcessConfigurationFile,  FileMode.Open) )
                {
                        ProcessBase process = ser.Deserialize(fs) as ProcessBase;
                        processList.Add(process);
                }
        }
}
        config.ProcessList = processList.ToArray(typeof(ProcessBase)) as  ProcessBase [];
        return s;
}
```

**Configuración**

La configuración de un servicio está descrita en un archivo en el formato estándar XML, y contiene la información necesaria para poder acceder a la configuración de cada proceso en específico y poder "**deserializar**" cada una de estas configuraciones en los objetos "**ProcessBase**", que son los procesos finales que se van a ejecutar al iniciar el servicio. Esta configuración está compuesta por los siguientes elementos o "tags" XML:

**ServiceConfiguration**: contiene todos los elementos que describen cómo acceder y "**deserializar**" cada tipo de proceso.

**ErrorCountStop**: representa el valor o cantidad de errores para los cuales el servicio deja de funcionar hasta que es reiniciado por el usuario que lo administre.

**ProcessConfigurationItems**: tiene la lista de los elementos "**ProcessConfigurationItem**" para cada proceso.

**ProcessConfigurationItem**: este elemento contiene la información para un proceso específico.

**ProcessConfigurationFile**: representa la dirección física en la estación que corre el servicio, donde está ubicado el archivo de configuración de un proceso.

**ContainerTypeName**: el nombre de la clase del proceso.

**ContainerAssembyName**: el nombre del ensamblado que contiene la clase del proceso.

**SerializerSettings**: contiene los elementos que definen las configuraciones o especificaciones necesarias para "**deserializar**" el objeto "**ProcessBase**". Este tag es opcional, ya que solo se usa para el proceso "**MessagePipe**" que contiene las propiedades de la bandeja de entrada y de salida que pueden ser clases que implementen las interfaces definidas para esto.

**OverrideAttributes**: incluye un conjunto de atributos "**ElementAttributes**" que definen que tipo de clases se permiten usar en cada tipo de bandeja del "**MessagePipe**".

**ElementAttributes**: tiene un atributo "**OwnerTypeName**" que representa el nombre completo del proceso, en este caso es el "**QuarkFramework.Transport.MessagePipe**" y otro llamado "**ElementName**" que describe el nombre de la propiedad de la clase "**MessagePipe**", a la cual se le va a definir qué tipos o clases puede ser incluidas, para este caso puede ser "**InboxObject**" u "**OutboxObject**".

**Attributes**: contiene la lista de atributos que describen cuáles son las posibles clases que implementan la interfaz de cada tipo de bandeja y que se usan para la propiedad que corresponde con el tipo de bandeja según la etiqueta anterior "**ElementAttributes**".

**ElementAttribute**: especifica a través de los atributos "AssemblyName" y "TypeName" el nombre del ensamblado donde se encuentra la clase que implementa una de las interfaces de las bandejas del "MessagePipe" y el nombre exacto de la clase ejemplo: "QuarkFramework.Transport.Files.DirectoryInbox".

Para un ejemplo de servicio que se esté desarrollando, que contenga un proceso que es una tubería de mensajes o "MessagePipe", que tiene una bandeja de salida que es un "DirectoryOutBox" donde se encuentran los archivos que serán recibidos por una bandeja de entrada en un FTP o "FtpInBox", el archivo de configuración quedaría definido de esta forma:

```
<ServiceConfiguration>
  <ErrorCountStop>100</ErrorCountStop>
  <ProcessConfigurationItem>
    <ProcessConfigurationFile>c:\ServicioEjemplo\DirectoryToFTP.config.xml</ProcessConfigurationFile>
    <ContainerTypeName>QuarkFramework.Transport.MessagePipe</ContainerTypeName>
    <ContainerAssembyName>QuarkFramework</ContainerAssembyName>
    <SerializerSettings>
      <OverrideAttributes>
      <ElementAttributes OwnerTypeName="QuarkFramework.Transport.MessagePipe"
ElementName="InboxObject">
        <Attributes>
          <ElementAttribute AssemblyName="QuarkFramework"
TypeName="QuarkFramework.Transport.FTP.FtpInbox" />
        </Attributes>
      </ElementAttributes>
      <ElementAttributes OwnerTypeName="QuarkFramework.Transport.MessagePipe"
ElementName="OutboxObject">
        <Attributes>
          <ElementAttribute AssemblyName="QuarkFramework"
TypeName="QuarkFramework.Transport.Files.DirectoryOutbox"/>
        </Attributes></ElementAttributes></OverrideAttributes></SerializerSettings></ProcessConfigurationItem>
 </ProcessConfigurationItems>
</ServiceConfiguration>
```

**Tratamiento de errores**

Durante el proceso de ejecución de cualquiera de las unidades de proceso del servicio, pueden ocurrir errores o "**Exceptions**" (Excepciones), que deben ser manejados para darles un tratamiento adecuado según el tipo que sea y en la situación en que se encuentre el proceso.

De forma general la clase "**ProcessBase**" realiza un tratamiento por defecto para cualquier proceso del servicio, el cual consiste en notificar el evento "**OnException**", para que el servicio pueda dar de forma global un tratamiento o tomar alguna decisión ante un error ocurrido en alguno de sus procesos, y luego de ser notificado el evento, el

proceso genera un "**log**", este consiste en un archivo que contiene todos los mensajes de error y su fecha, ocurridos durante todo el proceso de ejecución del servicio.

En el caso del proceso "**MessagePipe**", si ocurre un error durante la obtención, transmisión o eliminación de los mensajes que se intercambian entre las bandejas "**OutBox**" e "**InBox**", se debe tener en cuenta que al ser este un proceso que cada vez que se ejecute puede gestionar la transferencia de un conjunto de mensajes y por cada uno pudiese ocurrir alguna excepción. Por tanto en dependencia de cómo se diseñe esta "**tubería**", se podrá o no detener todo el procesamiento de los mensajes ante determinada situación específica. Para esto existe una tipo de excepción llamada "**ProcessException**" que contiene información como: el proceso que lanzó la excepción, el propio objeto "**Exception**", un mensaje para este nivel de la excepción y una propiedad llama "**Handled**" mediante la cual cada bandeja al lanzar su propio "**ProcessException**" puede decidir si "**maneja**" o no el error ocurrido, en el caso de manejar la excepción, esta se escribe en el "**log**" de errores, pero no detiene el proceso de transporte de los siguientes mensajes, si no se desea manejar la excepción ocurrida, entonces además de registrarse la excepción, se detendrá el proceso de transmisión.

**Aplicación del framework en el desarrollo de algunos servicios**

A continuación se muestra como este *framework* se usó en la implementación de dos servicios, los cuales se describen como ejemplo de una forma concreta de desarrollo, estos son el: Servicio de Notificación de Mensajes y el Servicio de Notificación de Cambio de Datos.

**Servicio de Notificación de Mensajes**

Este servicio permite la notificación a personas, entidades o instituciones de determinados eventos o cambios de estado que ocurren con los procesos que puede realizan en determinada entidad, y está orientado principalmente al envío de mensajes ya sea por correo electrónico, por SMS (*Short Message Service*) [3], o algún otro mecanismo de notificación que surja luego. De esta forma el destinatario siempre estará informado en el momento preciso en que ocurran estos eventos. A continuación se muestra el diagrama de flujo para este servicio:

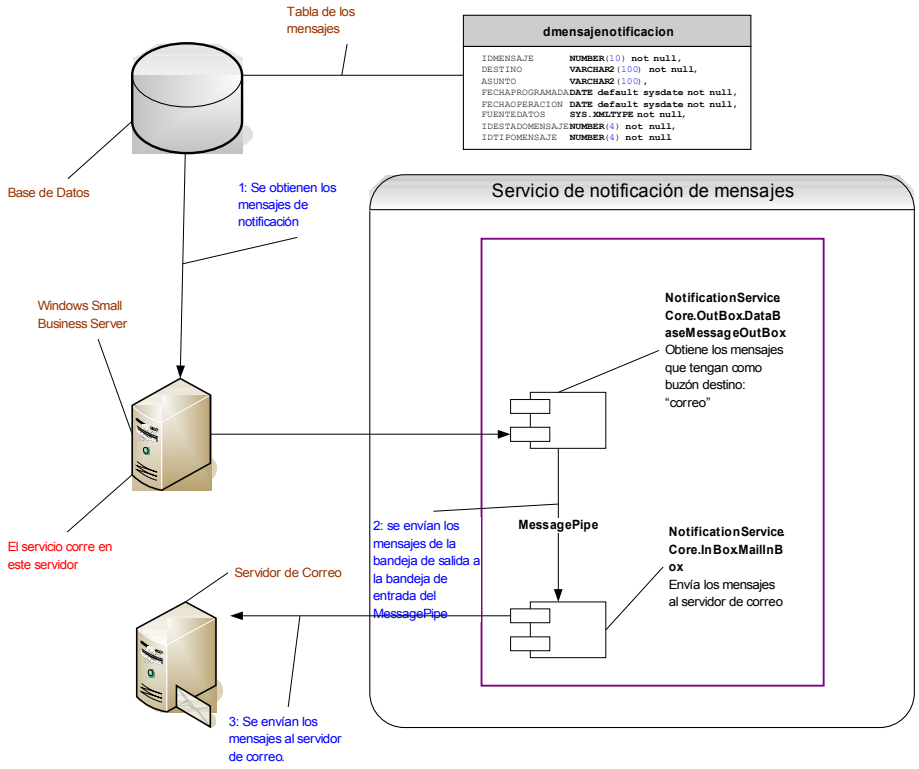


Fig. 3. Flujo para el Servicio de Notificación de Mensajes.

### Flujo de Procesos y Tuberías

Este servicio está compuesto solamente de un proceso de tipo "**MessagePipe**", que tiene en la bandeja de salida un "**DataBaseMessageOutBox**", que es una bandeja creada para conectarse a una Base de Datos y obtener los mensajes listos para enviar a las personas o instituciones, y una bandeja de entrada "**MailInBox**", creada para enviar correos usando protocolo SMTP (*Simple Mail Transfer Protocol* por sus siglas en inglés) [4].

El flujo del servicio comienza cuando los mensajes son obtenidos de la base de datos en la estructura de tablas definida para este caso. El mensaje se crea transformando la información de un evento específico dentro del sistema en formato XML y usando archivos en formato XSLT (*Extensible Stylesheet Language Transformations*) [5], que permiten, a través de un motor de transformaciones de XML, obtener el cuerpo del mensaje final en texto HTML para ser enviado por correo al usuario cuyos datos de dirección electrónica ya están almacenados.

### Servicio de Notificación de Cambio de Datos

El segundo servicio está dedicado a las instituciones o entidades externas a la entidad que lo utiliza, a los cuales se les notifica de cambios que ocurren en los datos de determinados "**conceptos**" o esquema de información que se maneje en el sistema que provee el servicio. Este mecanismo se implementa con el envío de los datos relacionados con los cambios en los "**conceptos**" a través de servidores FTP en los cuales los clientes interesados podrán obtener los datos que se han modificado en el sistema y de esa forma tener la información actualizada que necesitan en sus procesos.

**Flujo de Procesos y Tuberías**

Este servicio está compuesto por tres procesos principales, un "**MessagePipe**" que contiene en la bandeja de salida un "**DataBaseMessageOutBox**" y en la bandeja de entrada un "**DirectoryInbox**", luego hay un proceso de compresión de archivos y finalmente otro "**MessagePipe**", que tiene en la bandeja de salida un "**DirectoryOutbox**" y en la de entrada un "**SFtpInBox**".

El flujo del servicio comienza cuando los mensajes de cambios de datos en determinadas entidades de la base de datos del sistema, están listos para enviarse y son obtenidos por la bandeja de salida "**DataBaseMessageOutBox**", la cual los envía a la bandeja de entrada "**DirectoryInbox**" que los va guardando en formato XML, organizados en carpetas en dependencia del tipo de notificación que sea, es decir del tipo de entidad de la base de datos que se estén notificando sus cambios, ejemplo: Cedulado.

Constantemente el proceso de compresión de archivos está verificando la existencia de nuevos archivos XML creados por el "**DirectoryInBox**" y los va comprimiendo hacia las carpetas de cada tipo de notificación, separados de los XML. Luego el "**DirectoryOutBox**" le envía, como mensajes, los archivos compactados al "**SFtpInBox**" para que los envíe al servidor SFTP, que finalmente publicará los cambios de datos de las entidades que se notifiquen.

En el servidor existe un "**script**" en lenguaje Python que mueve los archivos compactados de cada notificación, hacia los buzones de los clientes, desde donde se consumen estos archivos, que ya contienen la información que necesitan, a través del protocolo FTPS. Este paso no es necesario, pero es puede usar para el caso en que se quiera lograrla una integración por ejemplo con un servidor en un Sistema Operativo Linux. La siguiente figura muestra el flujo de los procesos para este servicio:

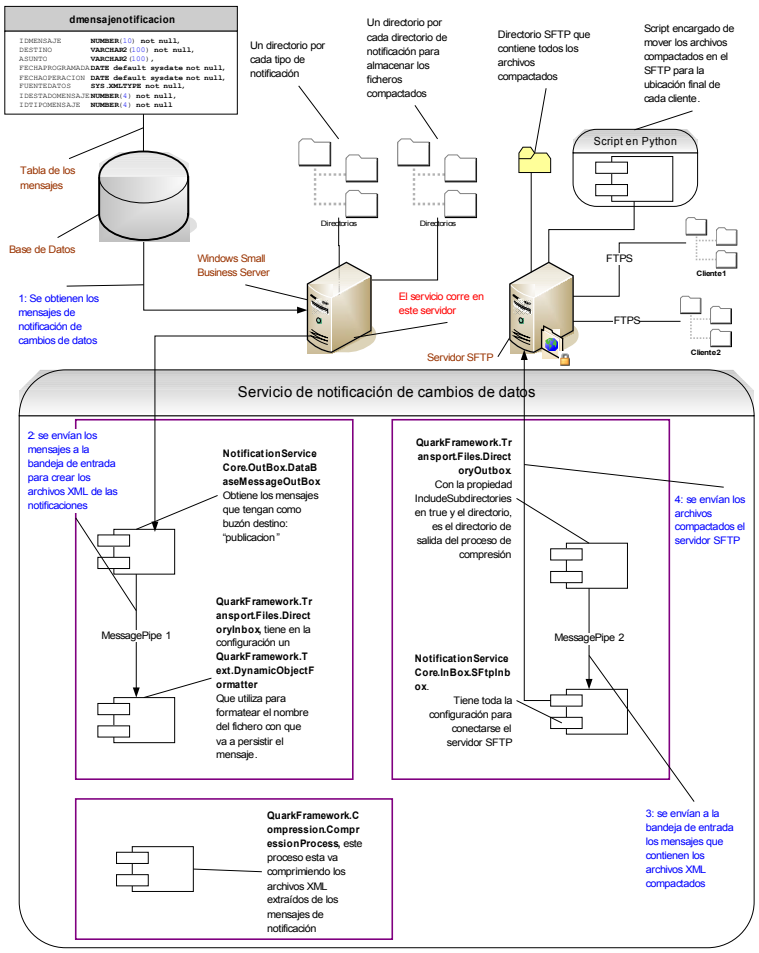


Fig. 4. Flujo para el Servicio de Notificación de Cambio de Datos.

## Conclusiones

Este trabajo permitió el estudio y análisis de un mecanismo general para la implementación de servicios configurables y extensibles, a través de la definición y desarrollo de los procesos que lo componen, con el objetivo de aumentar sus posibilidades de procesamiento. La propuesta de este *framework* define una metodología y determinadas métricas que ajustan el desarrollo de servicios de forma tal que los desarrolladores de un sistema solo deban ocuparse en extender sus funcionalidades o combinarlas a través de las configuraciones de los componentes que necesiten. Se logró una solución independiente de la plataforma por lo que es posible usarla o implementarla para otras tecnologías de desarrollo, en dependencia de los requerimientos y características de cada Sistema Operativo.

Con los servicios desarrollados se ha podido ofrecer una solución a la comunicación y retroalimentación de una entidad con entidades externas, instituciones o personas, ampliando de esta forma el conjunto de servicios que se puede brindan y mejorando su propio funcionamiento.

**Referencias Bibliográficas**


[1] Wikipedia. File Transfer Protocol, 2008. [Disponible en: http://es.wikipedia.org/wiki/FTP]

[2] Microsoft. Introduction to Windows Service Applications, 2008. [Disponible en: http://msdn2.microsoft.com/en-us/library/d56de412(VS.80).aspx]

[3] Wikipedia. Servicio de mensajes cortos, 2008. [Disponible en: http://es.wikipedia.org/wiki/Servicio_de_mensajes_cortos]

[4] Wikipedia. Simple Mail Transfer Protocol, 2008. [Disponible en: http://es.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol]

[5] Wikipedia. XSL Transformations, 2008. [Disponible en: http://en.wikipedia.org/wiki/XSLT]