1. Introduccin:

Cuando comenzamos a escribir algo de cdigo en alguno de nuestros proyectos anteriores, encontramos algunos problemas a la hora de ponernos de acuerdo para nombrar nuestros ficheros y crear la jerarqua de directorios. Esto no representa un problema si se trata de un proyecto pequeo; pero si se trata de un proyecto con varios componentes, sera recomendable saber qu es lo que hace cada componente y saber que partes va a tener sin tener que perder ese tiempo en mirar el cdigo fuente, utilizando nombres descriptivos.

Ahora, tenemos entre manos quizs el primer proyecto grande que sabemos que puede ser reutilizado en algn otro proyecto (common vfs). Y es bastante importante que tengamos estas cosas bien claras.

Como herramientas, en el subversion de mspace tenemos ya creado un build para NAnt, en scripts/nant.xml. Ese build es una implementacin de referencia para la estructura que voy a detallar ahora.

2. Descripcin:

Ahora que ya conocemos el problema que hemos tenido; mi propuesta es la siguiente. Es una propuesta que ya he utilizado alguna vez con resultados bastante buenos, a m personalmente me gusta. Tambin expondr las modificaciones que seran factibles de realizar.

2.1. Proyecto:

Utilizaremos proyecto como el programa que vamos a realizar, ese proyecto puede tener 1 o ms componentes. Y cada proyecto debe tener un nombre lo suficientemente descriptivo de lo que ser la aplicacin. Sin tener miedo a poner nombres largos.

2.2. Comunes a proyecto:

As pues, tendremos el directorio raiz, que ser el user.dir. Ese directorio contendr otros directorios, concretamente: lib, src, resources y luego adems existir un bin y un doc que se irn creando conforme utilizemos nant para compilar nuestro proyecto completo.

2.2.1. lib

Lib contendr otras libreras externas de las que dependa nuestro proyecto, si por ejemplo vamos a utilizar la libreria log4net, deberemos copiar el ensamblado a lib. Otra opcin que se barajar es si esa librera est instalada en el gac, como un strong assembly, pero eso ya es otro tema. En principio en esta IR no se da soporte de integracin con el gac. Si ademas en lib vamos a utilizar otros mspace components, debern ser copiados a lib/components. Con su ensamblado y adems otro fichero que describa el interface del component (falta mucho por discutir sobre como sacar esos interfaces).

2.2.2. resources

Aqu vendrn todo aquello que sean recursos para la aplicacin, todava el build de nant no soporta la inclusin de resources en el ensamblado, pero en la prxima versin si que lo har. As pues podemos tomar como recursos, todo aquello que sean iconos, mensajes de i8n, scripts para ejecucin de programas externos, logs, ficheros de configuracin ...

2.2.3. src

El corazn del proyecto, aqui se guardar el cdigo fuente de los componentes de la aplicacin. Ms tarde pasar a hablar sobre el esbozo de modelo de componentes que he pensado y propongo utilizar en los componentes mspace, para alguien que conozca la framework guasaj, le resultar bastante similar.

2.2.4. bin

Aqui se guardaran los binarios de la aplicacin, as como los pares de claves para firmar un ensamblado.

2.2.5. doc

Parece sencillo pensar que es lo que habr aqui.

2.3. src

De nuevo volvemos a src, aqui se explicar como hay que organizar nuestro cdigo fuente. Para no utilizar un sistema como el que utiliza java, es decir, que para cada paquete lleva implicito un directorio, dado que en .NET eso no es una restriccin y adems podemos hacer ms accesible nuestro cdigo fuente.

2.3.1. appname

Lo primero que encontraremos ser un directorio con el nombre de la aplicacin. Ese directorio contendr *solamente* el lanzador de la aplicacin, que sera conveniente llamarlo Launcher.cs y la clase debera ser Launcher. As a secas, y en su main debera hacer una llamada al InitApp del maincomponent.

2.3.2. appname.Components

Este directorio contendr el cdigo fuente de los componentes que hayamos escrito a la hora de escribir la aplicacin. Solo contendr el nombre del componente que est escrito. Si por ejemplo quiero un componente que sea un conector TCPIP, y lo escribo ahi, deberia llamarse tcpipconnector (o algo as similar).

2.4. Jerarquia en un componente:

Dentro de cada componente y adaptandonos ya un poco al modelo de componentes, tenemos los siguiente subdirectorios: Bo, Dao, Exception, Factory, Form, Interfaces, Resources, Vo

2.4.1. Bo

Aqu se implementar toda la lgica de negocio de nuestro componente, no tiene porque contener slo una sola clase. Pero deber contener un punto de entrada al modelo que contendr los casos de uso de nuestra aplicacin, y esos casos de uso sern los que deben ser implementados de nuestro interface. Como se implementen esos casos de uso, es tarea del programador del componente. Puede utilizar ms clases que estn dentro de Bo (la idea es tener todo el business object (Bo)) en este namespace.

2.4.2. Dao

Aqu estar todo aquello que sea un Data Access Object, es decir, objetos que sean para acceder a datos; esto es conexin con una base de datos, clases para leer xml. *No se debe acceder a Dao desde el Form!!!*

2.4.3. Exception

Aqu se meteran las excepciones que definamos nosotros para nuestro Bo y guasaj implementa tambin un gestor de esas excepciones con el mtodo processException (Exception exception). El gestor de excepciones es til para tener centralizado todo el cdigo de nuestras excepciones y saber dnde se gestionaran los errores.

2.4.4. Factory

Aqui estarn las factorias que utilizaremos para crear los objetos, existen 2 filosofias que seguir ac (las que yo propongo). De entrada, como a mi me han dicho y explicado:

> Todo es un singleton hasta que se demuestre lo contrario.

Es preferible hacer todo un singleton y si necesitamos varias instancias ya no deber ser un singleton (evidentemente). Y la otra es que cuanto menos llamemos a los constructores dentro del BO mejor, que mejor. Es preferible utilizar factoras para construir los objetos, dado que estos pueden ser extremadamente complejos y que a veces podemos necesitar cambiar un objeto en tiempo de ejecucin. (Esto se aplica bastante bien a la hora de crear y utilizar un Strategy).

2.4.5. Form

Aqui se ubicar todo aquello que sea parte de la vista. No importa la framework que vayamos a utilizar, nos dar lo mismo utilizar GTK o Windows.Forms o wxWidgets (no han sido consideradas las WebApps). Tiene que ser transparente y el modelo de componentes para ello obliga a implementar un ViewDispatcher para hacer esto de forma transparente.

2.4.6. Interfaces

Aqui se ubicaran los interfaces que se deban implementar en las clases del Bo. Ya son de sobras conocidas las ventajas que aportan los interfaces a nuestras jerarquas de clases.

2.4.7. Resources

Los recursos *propios* de cada componente, es como el resources de ms arriba, pero con un mbito ms reducido a cada componente. Probablemente tambin se empaqueten en otro ensamblado los resources.

2.4.8. VO

El value object es uno de los patrones ms sencillos y poderosos que he conocido. Para los que hayan programado en Java a este nivel, les resultar muy similar a un Bean, pero bastante ms sencillo. Simplemente estas clases son valores, es decir, estas clases no contienen acciones ni mtodo, slo propiedades. Un ejemplo claro puede ser la necesidad de crear una configuracin y llamar a los mtodos del Bo con una configuracin especifica segn como haya sido configurado. Asi pues utilizamos una clase ConfigurationVO, que contendr por ejemplo un valor string Nombre getter; setter; y segn sea ese valor se comportar de uno u otro modo la aplicacin. Para rellenar los valores de aqui, es til crearse un Dao que especificamente rellene el Vo. Por ejemplo si tenemos que utilizar una serializacin XML de esos valores, es fcil hacer que se serialice o que acceda a los campos del xml el dao y no el Vo.

A. Sobre el modelo de componentes:

Aqui voy a dar unas pequeas pinceladas sobre el modelo de componentes que vamos a utilizar para evitar el desacople. Est muy basado en Guasaj framework para Java, realmente yo cuando programo con Java ya slo lo hago con Guasaj; por las ventajas que tiene, pero por supuesto tambin tiene carencias que he visto y que me gustara solucionar.

La primera pregunta es: .NET ya tiene su propio modelo de componentes, porqu no lo utilizas? El modelo de componentes de .NET es un buen modelo, pero quizs sea demasiado complejo para poder llevar a cabo estas aplicaciones, no slo es un modelo de componentes, sino que tambin es un contenedor y el tema de los contextos no creo que sea necesario para realizar estas aplicaciones. Tambin decir, que exactamente lo que se va a utilizar no es un modelo de componentes completo, sera algo as mas parecido a una framework para el modelado de componentes y la organizacion del cdigo, ayudando al desacoplo entre componentes y manteniendo las dems caractersticas de una programacin orientada a componentes. Compatibilidad binaria, gozando de independencia de lenguajes, un mspace component debera ser accesible desde cualquier lenguaje .NET, y por supuesto cada instancia del componente es nica, cada componente debe ser un singleton (para variar).

No soy un experto en guasaj, pero he podido ver un poquito las ideas que esa framework conlleva, cada componente llevar unos casos de uso, y la clase que implementa esos casos de uso debe heredar de DefaultComponentHandler (en guasaj). Un componente se llama desde un configuration con el mtodo getComponentByName (string cmpName); una vez conseguido el DefaultComponentHanlder del componente, utiliza el patrn executor para ejecutar los casos de uso, que por reflexin acceder a los interfaces y lo ejecutar, adems de ejecutar una response a la vista donde haya sido definida en el interface. Es decir, el flujo de ejecucin es el siguiente: Conseguimos el componente, y hacemos un .execute

del mtodo con los parametros que sean necesarios ese caso de uso, en su interface xml lleva un response, que ser ejecutado a continuacin del caso de uso y ese response debe ser implementado en la clase que le digamos al interface (si no Exception) adems cada componente lleva asociado un ExceptionManager que se encargar de localizar las excepciones y de tratarlas. As pues conseguimos un desacople casi perfecto (por no decir perfecto) entre componentes y por supuesto un caso de uso puede responder a una vista, o a otra; incluso crear nuevas instancias de la vista con el ViewDispatcher y adems asociado a ello control de concurrencia sobre los componentes.

As pues, tendremos tambin un config indicando los componentes que existen y la localizacin de sus interfaces, logrando as tener nuestro ensamblado del componente por un lado, y por el otro la configuracin.

Esto puede tener tambin un problema, que es que un mal uso del fichero xml puede llevar a errores, por este motivo el xml debe ser bastante sencillo para localizar los errores que nos pueda dar.

Tambin tiene una restriccin fuerte, pero espero que sea solucionada pronto con el vfs; es la posibilidad de invocar componentes que estn en otra mquina; pero esto es bastante ms complejo y si queremos hacer esto no deberamos utilizar esta framework, sino un modelo de componentes completo. Dado que nosotros no creo que vayamos a utilizar esto dado que las aplicaciones que vamos a realizar tienen un mbito de aplicaciones de escritorio.

Otro problema que he visto es la notificacion de eventos, pero creo que esto es ms por carencias del propio lenguaje Java frente a .NET y tener event como un tipo propio del lenguaje. Con este event podemos hacer estas notificaciones de una buena manera, pero *no se deben definir eventos en el interface!!* debe ser algo ms interno al componente.

Finalmente el problema puede ser la redireccion a varias vistas lo cual el xml debe proveer un medio para decir a que vistas debe ser dirigida la response de un caso de uso. Lo cual, ampliando la definicin del xml debe ser una tarea sencilla.

B. Requerimientos

La tcnica que se va a utilizar conlleva un uso bastante grande de reflexin esto no es ningn problema.

El xml debe ser tratado de un modo rpido, pero dado que se debe permitir que los componentes no sean grandes, sino que sean algo parecido a piezas de un puzzle en nuestra aplicacin, el poder utilizar DOM para parsear el xml no es ningn problema.

Adems es preferible en todas las clases que estn fuera del ComponentHandler, hacer un get del componente para realizar las acciones; es decir, si en el form tengo un boton que lo clicko y debe realizar un caso de uso del componente, no llamar al Bo; sino que obtener el componente y utilizar el executor que lleva.