La Orientación a Objetos y sus enfoques como estilo y paradigma

*— Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

Comprender desde las bases la radical diferencia que existe entre la Orientación a Objetos vista como paradigma de programación y la que se observa desde el punto de vista arquitectónico, será la diferencia entre un sistema informático perfecto y uno erróneo.

La Orientación a Objetos parece, en ocasiones, ser un gran misterio difícil de dilucidar. Esto se debe a que **mucho se habla** de ella, **pero muy poco se conoce en realidad**.

Lo cierto es que la Orientación a Objetos (OO) no solo es un paradigma de programación. La Orientación a Objetos, de hecho, puede ser vista, analizada y estudiada desde dos **enfoques** absolutamente diferentes:

- como **paradigma** de programación
- como **estilo** arquitectónico

En una gran parte de la bibliografía, la OO se abarca desde ambas ópticas pero sin hacer una clara distinción. Tal es así, que sendos enfoques llegan a confundirse generando en el lector o estudiante, una idea equivocada sobre qué es en realidad la OO.

Este error, es el que más nos cuesta corregir en el público en general, a los docentes y teóricos que nos dedicamos al estudio de la OO desde sus dos enfoques. Por ello, intentaré realizar dicha distinción utilizando ejemplos concretos, recurriendo así al código antes que a las explicaciones teóricas.

Comenzaré entonces, con un ejemplo clásico de la orientación a objetos, basándome en *el objeto Usuario*.

Si recurrimos a la **OO como estilo arquitectónico**, nos podremos encontrar con la siguiente clase:

```
class Usuario(object):

    def __init__(self):
        self.nombre = ''
        self.clave = ''
        self.email = ''
        self.telefono = ''
```

```
    def registrar(self):
        pass
```

En cambio, si el mismo ejemplo, se abarca desde el paradigma, nos encontraríamos con algo similar a:

```
class Usuario(object):

    def __init__(self):
        self.usuario_id = 0
        self.nombre = ''
        self.datoscontacto = []

    def add_datocontacto(self, datocontacto):
        self.datoscontacto.append(composite(DatoContacto, datocontacto))

    def save(self, clave):
        pass

    def get(self):
        pass

    def destroy(self):
        pass


class DatoContacto(object):

    def __init__(self):
        self.datocontacto_id = 0
        self.denominacion = ''
        self.valor = ''
        self.usuario = 0

    # etc...
```

Como puede verse, **la complejidad del paradigma es lejanamente superior a la sencillez del estilo**.

> *El paradigma es complejo, detallista, minucioso, lógico y racional, mientras que **el estilo es pragmático**.*

Originalmente, la OO **desde el punto de vista arquitectónico,** promovía el uso del paradigma para el diseño de todos los componentes del sistema. Sin embargo, no lo hacía de forma explícita, sino que simplemente se limitaba a definir que **«todo componente del sistema debía ser tratado y manipulado como un objeto»**. Es decir, no se hablaba de «paradigma» sino tan solo de «objetos».

Esto, dio origen a que por efecto de los usos y costumbres, el estilo arquitectónico utilizara como valor prioritario, la sintaxis y elementos de la Programación Orientada a Objetos (POO) pero dejando a un lado **el «paradigma»**, el cual define **«un modo único e indiscutible de diseñar los objetos de un sistema pero no necesariamente todos sus componentes»**.

Indiscutible significa «que no tiene lugar a objeciones».
De esta forma, si yo dijese que el objeto persona «tiene dos ojos» cualquiera podría objetar mi afirmación diciendo que «los ojos no los tiene la persona, sino que los posee la cara de la persona». Y esto último, es indiscutible. Pues nadie podría decir que los ojos NO se encuentran ubicados en la cara. Es entonces, **cuando se llega a este nivel de abstracción inequívoca e inobjetable**, que **los objetos están siendo razonados desde un enfoque racional; desde el enfoque del paradigma de programación**.

Así, puede concluirse (en una forma demasiado simplista y poco ortodoxa) en que *«cualquier objeto sujeto a debate es un objeto visto desde un enfoque arquitectónico pero no teórico»*. Siendo inmensamente más ortodoxos, diríamos que:

> *El estilo orientado a objetos es pragmático mientras que el paradigma de programación es racional*

Copiaré ahora el ejemplo según estilo, resaltando aquello que para el paradigma sería incorrecto:

```
class Usuario(object):

    def __init__(self):
        self.nombre = ''
        self.clave = ''
        self.email = ''
        self.telefono = ''

    def registrar(self):
        pass
```

Objeciones según el paradigma
Para el paradigma de programación, se debe razonar de forma purista y ser muy racional al momento de abstraer los contextos.

- Objeción sobre la propiedad clave: Yo, como usuario, no me caracterizo por mi contraseña. De hecho, nadie debería conocer mi contraseña y las cualidades que «nos definen» son aquellas cualidades que «se perciben».
- Objeción sobre las propiedades e-mail y teléfono: yo como usuario no soy «de e-mail» y si dijese que «tengo un teléfono» debería tratar al mismo como un objeto. Sin embargo, no estoy hablando del «aparato telefónico» sino del número «de contacto». De allí la indiscutible afirmación de que tanto el teléfono como el e-mail son datos de contacto que «pertenecen» al usuario.
- Objeción al método registrar: registrar una acción que puede realizar un ser humano sobre el sistema informático. Por eso, a dicha funcionalidad se la denomina «recurso» (herramienta con la que cuenta el usuario para realizar determinada acción). Los recursos de un sistema, son parte de los componentes del sistema pero no forman parte de los actores. Los actores del sistema, según el paradigma, solo pueden efectuar por sí solos, 3 acciones: crearse (para persistir), leerse y destruirse.

Otro aspecto a tener en cuenta, es que **el sistema informático perfecto** no es aquel que elige la OO desde uno u otro enfoque, sino que **es aquel que sabe como utilizar ambos enfoques de forma equilibrada**.

El estilo arquitectónico tratará a cada uno de los componentes del sistema como un objeto. Esto significará que desde un *helper* hasta un formulario será un objeto para el sistema.

> ***Metáfora:*** *como ayuda para entender los sistemas informáticos equilibrando el uso de objetos,* ***pensar en un sistema como en un rodaje de TV*** *en el que existen actores y otros roles. Entonces,* ***todos los*** *componentes que puedan considerarse «****actores*** *principales y secundarios»* ***se definirán según el paradigma*** *y el resto, lo hará solo arquitectónicamente.*

Es el caso concreto de **MVC donde los actores son «los modelos»** mientras que las vistas, controladores, *helpers* y demás componentes del núcleo del sistema, solo poseen un «estilo orientado a objetos».

Debe además tenerse en cuenta que cuando se ha elegido un estilo orientado a objetos para el diseño del sistema, toda información que deba hacerse persistir deberá -previamente- analizar qué tipo de persistencia aplicaría a cada caso. En estos sistemas puede haber dos tipos de **persistencia**:

- la persistencia **de datos**
- la persistencia **de actores**

Todo lo que se considere «actor» será un objeto en el sentido más purista del término, es decir que será tratado bajo el dogma del paradigma sin excepciones. Según la opinión profesional del programador, los **«objetos puros»** (actores) podrán hacerse persistir en una **base de datos** (SQL o no-SQL), en archivos, etc.

Sin embargo, toda persistencia de **«datos fijos»** (no variables) preferentemente deberá hacerse **en archivos de configuración** (*config*) **o inicialización** (*settings*).

> *Tener en cuenta el tipo de persistencia, será lo que otorgue mayor portabilidad y facilidad de mantenimiento y evolución al sistema.*

Cuando uno se enfrente a determinados datos que requieran persistir en una DB, se estará entonces frente a un objeto que no está siendo tratado de la forma adecuada, ya sea porque se lo está tratando de forma arquitectónica cuando es en realidad un actor o bien, porque siendo un objeto de estilo estamos queriendo darle un tratamiento de actor.

DEFECTIVE BY DESIGN.org

HACKTIVISMO ÉTICO

LAS SIGUIENTES COMPAÑÍAS HACEN DAÑO A LA LIBERTAD DE LAS PERSONAS

amazon.com

Mediante sus plataformas Kindle y Prime. **Leer más »**

Apple utiliza el DRM para controlar iOS y a los usuarios de OS X. **Leer más »**

Microsoft

El DRM se incorpora en el corazón de Windows y muchos servicios como Silverlight, imponen el DRM en los usuarios. **Leer más »**

NETFLIX

No satisfecho con el uso de DRM para su servicio de *streaming*, ahora Netflix está tratando de imponer el DRM en la Web. **Leer más »**

SONY

Sony ha utilizado acciones legales para hostigar e intimidar a las personas que han modificado sus sistemas PS3. **Leer más »**

SI DE VERDAD APOYAS EL SOFTWARE LIBRE, ÚSALO, PROMUÉVELO Y ABANDONA LO PRIVATIVO SIN EXCUSAS. EL CAMBIO ESTÁ EN TUS MANOS.