METODOLOGÍAS DE PRODUCCIÓN DE SOFTWARE: PERSPECTIVAS FRENTE A LOS ENTORNOS VISUALES DE PROGRAMACIÓN ACTUALES

Patricio Letelier, Pedro Sánchez, Antonio Molina

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n, 46071 Valencia - España
email {letelier | ppalma | amolina}@dsic.upv.es

Resumen: Existe en la actualidad un incremento importante en el desarrollo de aplicaciones usando entornos de programación pues proveen un camino rápido para obtener productos software. Existe también un auge relevante en el uso de notaciones y/o metodologías. El problema surge cuando quien analiza y diseña usando una metodología encuentra después un salto conceptual considerable al pasar al entorno de programación, perdiéndose, el vínculo que se supone se mantiene siguiendo enfoques clásicos como el estructurado. Este artículo presenta las causas que favorecen esta situación y plantea una posible forma de abordarlas.

1.- INTRODUCCIÓN

La proliferación en el uso de entornos visuales de programación con las características intrínsicas a dichos entornos hacen necesario el plantearse una serie de cuestiones que cuantifiquen de alguna forma la calidad en el proceso de desarrollo de software asociado. Está reconocido que usando programación orientada a objetos difícilmente se consigue una implementación fiel al diseño por alguna de las siguientes razones: (1) se usa una base de datos relacional para la persistencia de los objetos; (2) el modelado de la interfaz del sistema normalmente no es parte del diseño del sistema; y (3) no se realiza una programación orientada a objetos pura. Cuando el entorno de desarrollo no es

orientado a objeto, el diseño estructurado es posiblemente la opción más adecuada. Incluso cuando el entorno de programación sí es orientado a objeto, si la implementación no es fiel al diseño entonces toda la parte de la aplicación que no es objetual (incluyendo la interfaz) puede ser diseñada siguiendo el enfoque estructurado. Por otro lado, se ha observado que quienes usan una metodología orientada a objetos (como por ejemplo OMT [2]) o notaciones gráficas (como UML [1]) y luego deben dar una solución en un entorno visual (orientado a objetos o no) se encuentran con que el esfuerzo que han empleado en el diseño del sistema les sirve principalmente para comprender bien el sistema. Es decir, pierden la capacidad de poder derivar una solución arquitectónica de manera más o menos directa a partir de dicho diseño.
La pregunta de interés es entonces la siguiente: *¿qué tan fácil y útil es aplicar técnicas de diseño cuando se implementa en un entorno visual*?. Destacamos los siguientes inconvenientes para conseguir el objetivo marcado:

- No existe una solución consolidada al problema si bien los entornos de programación visuales están ampliamente extendidos.
- La combinación entornos visuales de programación + RAD[1], ¿es realmente una solución al problema?.

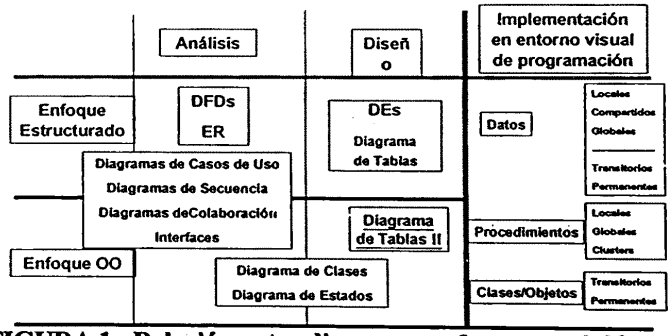

FIGURA 1. Relación entre diagramas y fases metodológicas.

2.- METODOLOGÍA Y DIAGRAMAS UTILIZADOS

La Figura 1 muestra para los enfoques estructurado y orientado a objetos qué diagramas se ven involucrados y en qué fases. En lo que al análisis se refiere, ambos enfoques comparten determinadas notaciones gráficas como los *Casos de Uso*, los *Diagramas de Secuencia* y las notaciones relativas a interfaces. Cada enfoque posee en exclusiva sus propios diagramas (por ejemplo, el

[1] *Rapid Application Development* (¿o quizás *Rapid and Dirty*?).

Diagrama de Flujo de Datos en el estructurado y *el Diagrama de Clases* en el orientado a objetos). Lo curioso es que se hace uso de diagramas de tablas en la parte del diseño orientado a objetos. Es decir, no existe en el mercado persistencia objetual, al menos, de uso comercial y lo suficientemente extendida. Es importante se consideren entonces los siguientes puntos:

- Se debe facilitar el mantenimiento y reuso. Uno de los grandes beneficios del enfoque orientado a objetos es la facilidad que provee la herencia en cuanto al reuso de aplicaciones minimizando los cambios necesarios.
- Asumimos que la persistencia es implementada usando una Base de Datos Relacional (que como hemos comentado es lo que ocurre actualmente en la mayoría de los casos).
- Pocas aplicaciones son programadas siguiendo un enfoque orientado a objetos estricto. Es más, gran parte de ellas sólo utilizan los objetos del entorno, siendo el resto programación no objetual.

La Figura 2 recoge los cambios en el esquema anterior para tener una situación más ideal. A partir de los requisitos obtenidos se genera un diseño mixto entre orientado a objetos y estructurado. La parte de implementación queda de la misma forma dado que viene prefijado por la tecnología actual. Dicho de otro modo, considerar OMT o UML es factible modelando en el análisis las clases que pertenecen al dominio del problema únicamente. En la fase de diseño se incorpora al conjunto de clases anterior aquellas clases que pertenecen al dominio de la solución (una lista de objetos de cierta clase, etc.).

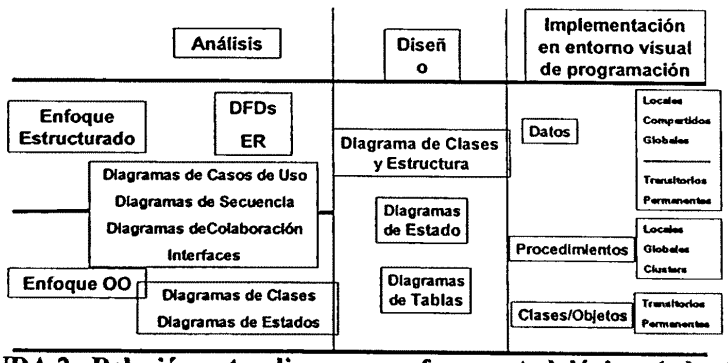

FIGURA 2. Relación entre diagramas y fases metodológicas (adaptado).

3.- PROPUESTA DE NOTACIÓN PARA EL DISEÑO

Los siguientes pasos presentan de forma resumida cómo integrar la notación del diseño orientado a objetos incorporando las características propias de los entornos de programación visuales:

1. Para cada clase del diseño orientado a objetos asocie un formulario de introducción/visualización de datos. La Figura 3 es un ejemplo de esta representación. Este diagrama de clases da por tanto la relación de inclusión[2] entre los distintos componentes de la aplicación, es decir, la visión estática.

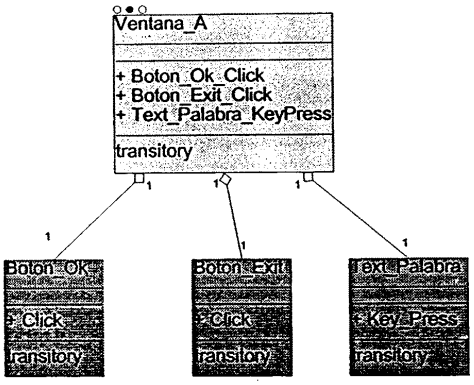

FIGURA 3. Un formulario para la clase "Ventana_A".

2. El siguiente paso es construir un diagrama (mezcla de diagrama de interacción y de estructura) para cada clase en el que se exprese la dinámica de la clase que se modela con respecto a los métodos o procedimientos que involucra, el resto de clases referenciadas y el acceso a la información persistente almacenada en la base de datos (relacional, por supuesto). La Figura 4 es un ejemplo de dicho diagrama. En los arcos se consideran distintas notaciones dependientes del mecanismo de comunicación considerado.

[2] La notación que se ha considerado es la de la agregación por estar el concepto íntimamente relacionado.

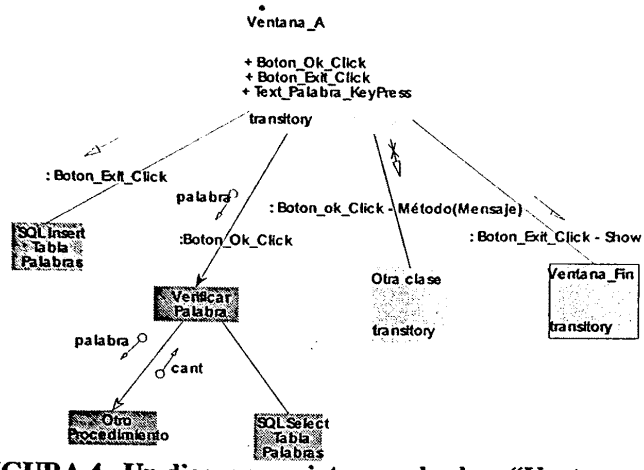

FIGURA 4. Un diagrama mixto para la clase "Ventana_A".

3. Por último se construye un diagrama de estados que capta cómo los distintos eventos que pueden ocurrir en la clase se habilitan/deshabilitan durante la vida del objeto que representa la clase (véase la Figura 5).

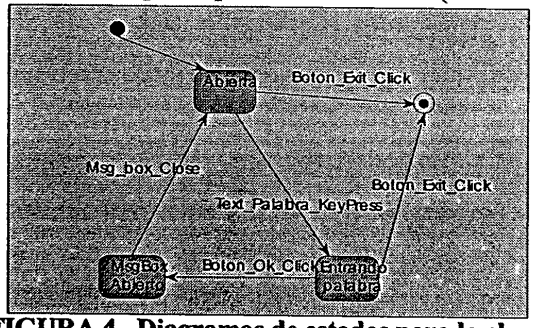

FIGURA 4. Diagramas de estados para la clase.

4.- RESUMEN DEL PROCESO

Podemos resumir lo anterior en los siguientes pasos y/o recomendaciones en cuanto al diseño con vistas a la implementación en un entorno visual de programación:

- Cada formulario es una agregación de los componentes descritos gráficamente. Se debe entonces dibujar la relación formulario-componentes en un Diagrama de Clases separado.

- Cada formulario tiene asociado un Diagrama de Estados correspondiente a la secuencia de eventos que pueden ocurrir. Este diagrama determina los eventos relevantes (o inhabilitación de componentes).
- Los procedimientos no establecidos como métodos de objetos se representan usando Diagramas de Estructura, invocados desde métodos de clases. En la llamada se debe indicar el nombre del método que hace la invocación y los parámetros utilizados.
- Cada procedimiento o clase (incluyendo formularios) que acceda a tablas tendrá un módulo que representa la sentencia SQL aplicada a dicha tabla.
- La comunicación entre clases se establece mediante "links", la conexión entre módulos y clases (o viceversa) se establece mediante "call".
- En un formulario, además de los métodos asociados a eventos de los componentes pueden existir procedimientos privados que serán descritos del mismo modo que un método del formulario.
- Las tablas que corresponden a la implementación de persistencia de clases sólo son accedidas mediante métodos de los respectivos objetos. Está es la mínima fidelidad exigida para la implementación de objetos.
- Usar filosofía de arquitectura de tres niveles, separando adecuadamente *presentación*, *aplicación* y *datos*. Los procesamientos más cercanos al contexto de la aplicación deberían modelarse como métodos de clases del problema o módulos separados del formulario.

5.- CONCLUSIONES

El incremento en el uso de entornos visuales de programación frente al auge en la consideración de metodologías orientadas a objetos hace necesario un esfuerzo en poder compaginar ambos de manera razonable. El problema asociado a los sistemas RAD es que el diseño viene a recoger aspectos del dominio del problema, trivializando otros relativos, entre otros, a la definición de interfaces. Este trabajo propone un método para no obviar estos aspectos.

6.- BIBLIOGRAFÍA

[1] Rational Software Corporation. *UML Notation Guide*. Versión 1.1. September 1997, http://www.rational.com/uml.

[2] J. Rumbaugh, M. Blaha,, W. Premerlani, F. Eddy, W. Lorensen. *Object Modelling and Design*. Publ. Prentice-Hall, 1991.