Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

Separación del código de interfaz de usuario

Martin Fowler

El primer programa que escribí con un salario fue un software de cálculo científico en Fortran. Mientras escribía, noté que el código que ejecutaba el sistema de menú primitivo difería en estilo del código que realizaba los cálculos.

Asi que Separé las rutinas para estas tareas, que valió la pena cuando me pidieron que creara tareas de nivel superior que realizaron varios de los pasos individuales del menú. Podría simplemente escribe una rutina que llame a las rutinas de cálculo directamente sin involucrar los menús.

Por lo tanto, aprendí por mí mismo un principio de diseño que me ha sido muy útil en el desarrollo de software: manter el código de la interfaz de usuario separado de todo lo demás. Es una regla simple, incorporada en más de un marco de aplicación, pero a menudo no se sigue, lo que causa bastantes problemas.

Declarando la separación

Cualquier código que haga algo con una interfaz de usuario solo debe incluir código de interfaz de usuario. Puede recibir información del usuario y mostrar información, pero no debe manipular la información más que para formatearla para su visualización. Un fragmento de código claramente separado (rutinas, módulos o clases separados (según la estructura organizativa de su idioma)) debe realizar cálculos, validaciones o comunicaciones que deben realizarse mediante un fragmento de código claramente separado. En el resto del artículo, me referiré al código de la interfaz de usuario como *código de presentación* y al otro código como *código de dominio.*

Al separar la presentación del dominio, asegúrese de que ninguna parte del código del dominio haga referencia al código de la presentación. Por lo tanto, si escribe una aplicación con una GUI WIMP (ventanas, iconos, mouse y puntero), debería poder escribir una interfaz de línea de comandos que haga todo lo que puede hacer a través de la interfaz WIMP, sin copiar ningún código de la WIMP en la línea de comando.

¿Por qué hacer esto?

Seguir este principio conduce a varios buenos resultados. Primero, este código de presentación separa el código en diferentes áreas de complejidad. Cualquier presentación exitosa requiere un poco de programación, y la complejidad inherente a esa presentación difiere en estilo del dominio con el que trabaja. A menudo, utiliza bibliotecas que solo son relevantes para esa presentación. Una separación clara le permite concentrarse en cada aspecto del problema por separado, y una cosa complicada a la vez es suficiente. También permite que diferentes personas trabajen en piezas separadas, lo que es útil cuando las personas quieren perfeccionar habilidades más especializadas.

Hacer que el dominio sea independiente de la presentación también le permite admitir múltiples presentaciones en el mismo código de dominio, como sugiere el ejemplo de WIMP versus línea de comandos, y también al escribir una rutina de Fortran de nivel superior.

01/$10.00©2001IEEE

0740-7459/

Las presentaciones múltiples son la realidad del software. El código de dominio suele ser fácil de transferir de una plataforma a otra, pero el código de presentación está más estrechamente acoplado al sistema operativo o al hardware. Incluso sin la portabilidad, es común encontrar que las demandas de cambios en la presentación ocurren con un ritmo diferente al de los cambios en la funcionalidad del dominio.

Eliminar el código de dominio también hace que sea más fácil detectar y evitar la duplicación en el código de dominio. Las diferentes pantallas a menudo requieren una lógica de validación similar, pero cuando está oculta entre todo el manejo de la pantalla, es difícil de detectar. Recuerdo un caso en el que una aplicación necesitaba cambiar su fecha de validación. La aplicación tenía dos partes que usaban diferentes idiomas. Una parte tenía la validación de fecha copiada sobre sus widgets de fecha y requirió más de 100 ediciones. El otro hizo su validación en una sola clase de fecha y solo requirió un solo cambio. En ese momento, esto se promocionó como parte de la ganancia masiva de productividad que podía obtener con el software orientado a objetos, pero el antiguo sistema sin objetos podría haber recibido el mismo beneficio al tener una rutina de validación de fecha única. La separación produjo el beneficio.

Las presentaciones, particularmente las WIMP y las presentaciones basadas en navegador, pueden ser muy difíciles de probar. Si bien existen herramientas que capturan los clics del mouse, las macros resultantes son muy difíciles de mantener. Las pruebas de manejo a través de llamadas directas a rutinas es mucho más fácil. Separar el código de dominio hace que sea mucho más fácil de probar. La testabilidad a menudo se ignora como un criterio para un buen diseño, pero una aplicación difícil de probar se vuelve muy difícil de modificar.

Las dificultades

Entonces, ¿por qué los programadores no separan su código? Gran parte de la razón radica en las herramientas que dificultan el mantenimiento de la separación. Además, los ejemplos de esas herramientas no revelan el precio por ignorar la separación.

En la última década, la herramienta de presentación más grande ha sido la familia de plataformas para desarrollar interfaces WIMP: Visual Basic, Delphi, Powerbuilder y similares. Estas herramientas fueron diseñadas para colocar interfaces WIMP en bases de datos SQL y tuvieron mucho éxito. La clave de su éxito fueron los widgets con reconocimiento de datos, como un menú emergente vinculado a una consulta SQL. Estas herramientas son muy poderosas y le permiten crear rápidamente una interfaz WIMP que opera en una base de datos, pero las herramientas no proporcionan ningún lugar para extraer el código de dominio. Si todo lo que hace es actualizar directamente los datos y la vista, entonces esto no es un problema. Incluso como un fanático de objetos certificado, siempre recomiendo este tipo de herramientas para este tipo de aplicaciones. Sin embargo, una vez que la lógica del dominio se vuelve complicada, resulta difícil ver cómo separarla.

Este problema se hizo particularmente obvio a medida que la industria se trasladó a las interfaces web. Si la lógica del dominio está bloqueada dentro de una interfaz WIMP, no es posible usarla desde un navegador web.

Sin embargo, las interfaces web suelen generar los mismos problemas. Ahora tenemos tecnologías de página de servidor que le permiten incrustar código en HTML. Como una forma de establecer cómo aparece la información generada en la página, esto tiene mucho sentido. La estructura comienza a descomponerse cuando el código dentro de la página del servidor es más complicado. Tan pronto como el código comienza a realizar cálculos, ejecutar consultas o realizar validaciones, se encuentra con la misma trampa de mezclar la presentación con el código de dominio. Para evitar esto, cree un módulo separado que contenga el código de dominio y solo haga llamadas simples desde la página del servidor a ese módulo. Para un conjunto simple de páginas, hay una sobrecarga (aunque yo la llamaría pequeña), pero a medida que el conjunto se vuelve más complicado, el valor de la separación aumenta.

Eliminar el código de dominio también hace que sea más fácil detectar y evitar la duplicación en el código de dominio.

Este mismo principio, por supuesto, está en el corazón del uso de XML. Construí mi sitio web, www.martinfowler.com, escribiendo XML y convirtiéndolo a HTML. Me permite concentrarme en la estructura de lo que estaba escribiendo en un solo lugar, por lo que podría pensar en su formato más tarde (no es que tenga un formato elegante). Aquellos que usan estilos orientados al contenido en procesadores de texto están haciendo casi lo mismo. He llegado al punto en que ese tipo de separación parece natural. Tuve que convertirme en un genio de XSLT, y las herramientas para eso aún no son adolescentes.

El principio general aquí es el de separar las preocupaciones, pero encuentro un principio tan general difícil de explicar y seguir. Después de todo, ¿qué preocupaciones debería separar? La presentación y el dominio son dos preocupaciones separables que he encontrado fáciles de explicar, aunque ese principio no siempre es fácil de seguir. Creo que es un principio clave en un software bien diseñado. Si alguna vez vemos códigos de ingeniería para software, apuesto a que la separación de presentación y dominio estará allí en alguna parte.

Martin Fowler es el científico jefe de ThoughtWorks, una empresa de consultoría y entrega de sistemas de Internet. Durante una década, fue un consultor independiente pionero en el uso de objetos en el desarrollo de sistemas de información empresarial. Ha trabajado con tecnologías que incluyen Smalltalk, C ++, bases de datos relacionales y de objetos, y Enterprise Java con dominios que incluyen arrendamiento, nómina, comercio de derivados y atención médica. Es particularmente conocido por su trabajo en patrones, UML, metodologías ligeras y refactorización. Ha escrito cuatro libros: Analysis Patterns, Refactoring, Planning Extreme Programming y UML Distilled. Contáctelo en fowler@acm.org.