Implementación de una Solución Reutilizable para una Funcionalidad de Usabilidad

Francy D. Rodríguez[1], Silvia T. Acuña[2]

[1] Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, España
fd.rodriguez@alumnos.upm.es
[2]Departamento de Ingeniería Informática, Universidad Autónoma de Madrid, Calle Francisco Tomás y Valiente 11, 28049, Madrid, España
silvia.acunna@uam.es

Resumen. La usabilidad es un atributo de calidad y un aspecto crítico en los sistemas de software. Se ha establecido que algunas de las recomendaciones para mejorar la usabilidad dadas desde el campo de la Interacción Persona Ordenador tienen impacto en el diseño de software. En este artículo presentamos la implementación de una solución reutilizable para realizar una funcionalidad de usabilidad con alto impacto en el diseño: Abortar Operación. Desarrollamos tres aplicaciones web como casos de estudio, incluimos esta funcionalidad de usabilidad y buscamos elementos comunes en las implementaciones. Encontramos escenarios de aplicación, responsabilidades, clases, métodos, atributos y trozos de código comunes en los tres desarrollos. Con base en estos hallazgos, proponemos elementos reutilizables para incorporar la funcionalidad de usabilidad en el análisis, diseño y programación. Formalizamos la solución como un patrón de diseño y patrones de programación en tres lenguajes: PHP 5, Java y Visual Basic .NET.

Keywords: Usabilidad del software; Abortar operación; Patrón de diseño; Patrón de programación

1 Introducción

La usabilidad es un atributo de calidad que contempla que usuarios concretos puedan usar un producto satisfactoriamente de forma efectiva y eficiente logrando objetivos específicos en un determinado contexto [1]. La usabilidad es un aspecto crítico en sistemas software interactivos [2], y genera importantes ahorros e incremento de utilidades [3] [4] [5], razones por las cuales cada vez se tiene más en cuenta en el desarrollo de software [6].

La usabilidad de un sistema ha sido abordada ampliamente desde el campo de la Interacción Persona Ordenador (IPO). Las técnicas pertenecientes al campo de la IPO permiten alcanzar un adecuado nivel de usabilidad en un sistema, pero no son fáciles de asimilar desde la Ingeniería de Software (IS).

En IS, la usabilidad era vista como un requisito no funcional relacionado únicamente con la interfaz gráfica, lo que contribuía a que se abordara en etapas

A. Ruíz, L. Iribarne (Eds.): JISBD'2012, pp. 159-172, ISBN:978-84-15487-28-9.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

tardías del ciclo de desarrollo. Posteriormente, algunos autores han argumentado que los aspectos de usabilidad generan restricciones estáticas y dinámicas sobre los componentes de software [7], otros resaltan que la separación entre la capa de presentación y la capa de negocio no es suficiente para garantizar la usabilidad, y que es necesario utilizar otras tácticas en el diseño de software si se quiere obtener un producto usable [8]. En estudios más recientes se ha evaluado la relación entre la usabilidad y los requisitos funcionales, llegando a concluir que, algunas características que mejoran la usabilidad, tienen impacto directo en la funcionalidad del software y en su diseño [6] [9], razón por la cuál se deben abordar desde el inicio del ciclo de desarrollo.

Las características de usabilidad con impacto sobre el diseño son las que implican el desarrollo o construcción de cierta funcionalidad dentro del sistema para mejorar la interacción usuario-sistema, tales como Retroalimentación de progreso, Alertas, Deshacer global o Preferencias. Las características de usabilidad con impacto sobre el diseño requieren componentes de software específicos para cumplir con sus responsabilidades asociadas, no solamente modificaciones a nivel de interfaz de usuario.

En [6] las autoras presentan evidencia empírica de la relación entre usabilidad y diseño de software, identifican características funcionales de usabilidad (CFU) con alto impacto sobre el diseño y miden su impacto en aplicaciones reales. Las funcionalidades identificadas son producto de recomendaciones IPO, cada autor IPO identifica diferentes subtipos para una CFU, cada subtipo se denomina mecanismo de usabilidad (MU). Los MUs tienen un nombre indicativo de su funcionalidad, por ejemplo la CFU Deshacer/Cancelar esta compuesta por cuatro MUs: Deshacer global, Deshacer sobre un objeto específico, Abortar operación y Regresar. En [9] se presentan guías para educir la funcionalidad de MUs con impacto en el diseño.

En este artículo presentamos una solución reutilizable para implementar el MU Abortar Operación (AO). La funcionalidad AO está enfocada en permitir al usuario cancelar una operación, un comando o salir de la aplicación de una forma segura y predecible. Con base en los estudios previos [6] [9] que identifican funcionalidades de usabilidad con alto impacto en el diseño, y proporcionan guías de educción, llevamos a cabo la implementación de la funcionalidad de usabilidad en tres casos de estudio distintos pensando en la reutilización.

La solución que proponemos busca proporcionar a un desarrollador herramientas para incluir de forma efectiva, eficiente, con menos errores y al menor costo posible, la funcionalidad de usabilidad AO en un sistema software. Esta solución consta de varios artefactos[2]: un conjunto de escenarios de aplicación, un patrón de diseño y código que implementa el diseño en tres lenguajes de programación (PHP 5, Java y VB .NET). Hemos llamado a la unión de diseño y código, patrón de programación. Los patrones de programación son patrones de bajo nivel de abstracción que implican el desarrollo de partes de componentes o de relaciones entre ellos [10]. Los patrones de programación son una solución por sí mismos y describen cómo implementar aspectos particulares de un patrón de diseño usando las características y potencialidades de un lenguaje de programación. A su vez, la solución reutilizable

[2] Un artefacto software es un objeto o producto de cualquier etapa del ciclo de desarrollo de software, algunos ejemplos son: casos de uso, diagramas de diseño, prototipos y librerías.

propuesta ha sido usada en el desarrollo de dos casos de estudio adicionales por dos desarrolladores diferentes, las implementaciones fueron hechas usando dos de los lenguajes para los cuales se presenta esta solución (Java y VB .NET).

Este artículo se ha estructurado de la siguiente manera: en la sección 2 se analizan trabajos relacionados que abordan la usabilidad mediante patrones. En la sección 3 se describe el método de investigación utilizado, detallando los casos de estudio desarrollados. En la sección 4 analizamos el proceso llevado a cabo para la identificación de escenarios de aplicación. En la sección 5 mostramos en detalle el proceso de formalización de la solución como un patrón de programación. En la sección 6 se hace referencia a la evaluación de la solución propuesta en dos casos de estudio adicionales. En la sección 7 se realiza una discusión acerca de las características de la solución propuesta. Finalmente en la sección 8 se presentan las conclusiones.

2 Trabajos relacionados

En los últimos años se han desarrollado estudios que buscan abordar los aspectos de usabilidad desde las primeras etapas del ciclo de desarrollo de software. En [11] [12] [8] los autores identificaron un conjunto de escenarios de usabilidad para los que la estrategia de separación de la interfaz de usuario no era suficiente para obtener un sistema usable, y definieron patrones arquitectónicos de soporte a la usabilidad. Posteriormente, se hizo un estudio a nivel empresarial usando estos patrones arquitectónicos, y con base en los resultados propusieron un Lenguaje de Patrones basado en Responsabilidades para patrones arquitectónicos de soporte a la usabilidad [13]. Este resultado constituye una descripción general de las responsabilidades que los distintos elementos funcionales deben cumplir, pero no proponen soluciones a un bajo nivel de abstracción para la implementación de los aspectos de usabilidad.

En línea con los trabajos anteriores se encuentra el proyecto STATUS [14], en el que se estudió la relación entre la arquitectura de software y la usabilidad, y se presentó una aproximación para mejorar la usabilidad aplicando un proceso específico de diseño. El proyecto STATUS plantea la incorporación de la usabilidad en momentos tempranos del desarrollo y adelanta el ciclo de evaluación/mejora al momento arquitectónico. Otro trabajo relacionado es en el que se proponen guías para la educción de funcionalidades de usabilidad [9] que apoyan la inclusión de la usabilidad desde la primera fase de desarrollo de software, en concreto desde la definición de requisitos.

Todos estos trabajos proporcionan herramientas para las primeras fases del proceso de desarrollo, pero no para las fases de diseño detallado e implementación. En [15] se presenta una propuesta que llega al nivel de implementación para la funcionalidad de usabilidad Alertas desde un enfoque diferente, la Programación Orientada a Aspectos. Esta propuesta busca solucionar un problema que se presenta con las aproximaciones expuestas anteriormente: el entrelazamiento de la funcionalidad propia de una aplicación con la funcionalidad de usabilidad. Sin embargo, todavía es necesario determinar cuales características de usabilidad pueden ser modeladas como aspectos [16] y evaluar las ventajas de usar este enfoque para la implementación de funcionalidades de usabilidad.

En este trabajo también llegamos a nivel de implementación, utilizando el enfoque de diseño y programación orientada a objetos. La solución reutilizable que se plantea, permite incluir la funcionalidad de usabilidad AO en un sistema software. Aunque se presenta entrelazamiento entre la funcionalidad de usabilidad y la del sistema, se proporcionan componentes reutilizables que encapsulan la funcionalidad del MU AO y se establecen los puntos de enlace con la aplicación. La solución ofrece las ventajas asociadas a la reutilización, tales como: menos errores, fiabilidad (código ya probado), adaptación a cambios en los requerimientos y menor coste de desarrollo.

3 Método de investigación

Para este estudio hemos utilizado un método de investigación inductivo en tres fases, implementando casos de estudio e induciendo a partir de ellos una solución general. Seleccionamos un MU con alto impacto en el diseño. La funcionalidad escogida fue AO que pertenece a la CFU Deshacer/Cancelar. Los criterios para la elección del MU fueron:

- Nivel de impacto sobre el diseño en cuanto a número de funcionalidades afectadas, determinado por las características de los casos de estudio a desarrollar.
- Facilidad de reconocimiento por parte de un usuario cuando use el sistema. Algunos MUs requieren la interacción del usuario para activar su funcionalidad por ejemplo retroalimentación de progreso. Otros MUs requieren que se produzca algún evento en el sistema como es el caso de Alertas, mientras que otros, como AO son opciones visibles para el usuario en todo momento.
- Facilidad de evaluación desde el punto de vista de las recomendaciones IPO. Todos los MUs tienen explicadas las recomendaciones IPO en las que se basan [17], por lo tanto, al final no fue un criterio decisivo para escoger el MU a desarrollar.

Los tres casos de estudio desarrollados son aplicaciones web interactivas basadas en requisitos para sistemas reales. El primero es un sistema de Administración de Indicadores que permite crear indicadores y datos simples, clasificar, consultar e importar datos. Es un sistema hecho en PHP 5[3] con una base de datos MySql. El segundo caso de estudio es un sistema web para generación de variables de pago, permite actualizar y gestionar información para pagos de nómina, calcula información de horas extras, nocturnas, festivas, y días trabajados. El sistema está hecho en VB .NET con una base de datos en Microsoft SQL. El tercer caso de estudio es un sistema de venta de comida saludable. El sistema permite suscripciones, crea y mantiene datos sobre del estado de salud del suscriptor, propone una dieta saludable, y da diferentes opciones para su compra y entrega. El sistema está hecho en Java con una base de datos PostgreSQL.

En la primera fase de la investigación se construyeron los sistemas web asegurando que además de su funcionalidad propia cumplieran con la funcionalidad asociada al

[3] Aunque PHP tradicionalmente no ha sido un lenguaje orientado a objetos, las últimas versiones del lenguaje ofrecen las funcionalidades necesarias para programar con este enfoque, usando clases y objetos.

MU AO. La educción de requisitos se realizó usando la guía de educción para este MU que se puede ver en el Apéndice Web[4]. En cada artefacto generado durante el proceso de desarrollo se marcaron los elementos relacionados con la funcionalidad de usabilidad. En la segunda fase identificamos los elementos comunes en las implementaciones del MU en los tres casos de estudio, y establecimos cuáles podían ser reutilizables. Al analizar la implementación de los tres casos de estudio surgió un elevado número de posibles escenarios de aplicación, cuya identificación no fue evidente al usar la guía de educción. La identificación y documentación de los escenarios solo fue posible a posteriori. Los escenarios son una parte de la solución que resulta ser útil para las fases de educción de requisitos, análisis y diseño.

En la tercera fase del estudio, formalizamos los resultados en forma de patrones. Se propone un diseño único y se adecuan las implementaciones en los tres lenguajes de programación utilizados: PHP 5, VB .NET y Java. La unión del diseño propuesto y los códigos implementados se formalizan como patrones de programación. Como parte de la tercera fase también se extraen "piezas de código" comunes, como un primer paso para la construcción de una biblioteca de componentes para la funcionalidad de usabilidad. Los resultados han sido utilizados por desarrolladores diferentes en casos de estudios independientes para evaluar su aplicabilidad.

4 Relación entre la Funcionalidad de Usabilidad y sus Múltiples Escenarios

La guía de educción del MU AO divide las preguntas en tres niveles: aplicación, operación y comando. A nivel de aplicación, la guía indica que se debe preguntar al usuario si es necesaria una opción para salir de la aplicación, y si es así, cómo se mostrará la opción al usuario. Según la recomendación IPO asociada, la opción de salir debe ser inmediata y obvia, incluso con uso de diálogos modales. Si la opción de salir se solicita cuando hay datos modificados, la opción "Salvar" debe estar presente.

El nivel de operación del MU AO se refiere a acciones que implican la ejecución de uno o varios pasos dentro de una aplicación, cada uno de los cuales requiere interacción con el usuario. Cada acción tiene como consecuencia la modificación del estado de la aplicación, ya sea por modificación de información de la base de datos, cambios en parámetros de configuración, o cambios en las variables de aplicación o de sesión en el caso de aplicaciones Web. Finalmente, el nivel de comando del MU AO se refiere a una instrucción que el usuario da a la aplicación a través de una única interacción, a través de botón, un enlace, eligiendo una opción de menú o cualquier otra opción suministrada por la aplicación.

Después de implementar los casos de estudio analizamos elementos comunes en cada fase del desarrollo para la funcionalidad de usabilidad. Comenzamos analizando los casos de uso de cada aplicación que incluían el MU AO. En todos los casos la funcionalidad del MU estaba definida como un camino alternativo a la funcionalidad principal. Describimos cada camino alternativo asociado al MU usando diagramas de secuencia. Encontramos una relación entre las preguntas de la guía de educción, las posibles situaciones que se presentan en la aplicación, las interpretaciones para un

[4] http://www.grise.upm.es/sites/extras/7/Usability_Elicitation_Pattern_AO.pdf

sistema web y el estado final del sistema. En el Apéndice Web[5] se encuentra una tabla resumen con las relaciones encontradas. En resumen, el comportamiento descrito en los diagramas de secuencia para la funcionalidad asociada al MU AO está definido por cuatro aspectos principales:

- Ubicación de la opción de cancelación (Botón cancelar en un cuadro de dialogo modal o no modal, botón cancelar en un formulario, selección de una opción diferente del menú mientras se realiza una operación, botón limpiar).
- Existencia o no de cambios pendientes por guardar en la aplicación.
- Intención de guardar o no los cambios existentes.
- Existencia o no de errores al momento de guardar.

Por ejemplo, a nivel de aplicación cuando se responde a las pregunta de la guía: ¿Es necesaria una opción para salir de la aplicación? Si la respuesta es positiva, y se atiende a la recomendación IPO de que debe existir la opción de guardar los cambios pendientes, se pueden presentar dos situaciones, que al solicitar salir existan o no cambios por guardar. Si no existen cambios, el sistema pasará al estado siguiente. Si por el contrario, existen cambios por guardar, se debe preguntar al usuario si desea o no guardarlos, si el usuario no desea guardar los cambios se pasa al estado siguiente, pero si el usuario desea guardar pueden darse varias situaciones: que se guarde con éxito, que existan errores de validación o que existan errores de base de datos, en cada caso la aplicación debe quedar en un estado predecible.

Para tener una lectura rápida de las posibles combinaciones para el MU AO construimos árboles de escenarios para cada nivel: aplicación, operación y comando. En la **Fig. 1** se presenta el árbol de escenarios a nivel de aplicación, a este nivel existe una única opción para salir de la aplicación, el árbol muestra la posible casuística que puede generarse dependiendo de si existen o no cambios por guardar, de la decisión del usuario de guardar o no los cambios, y del resultado del proceso de guardado. La parte derecha del árbol muestra las posibles respuestas del sistema y el estado final de la aplicación.

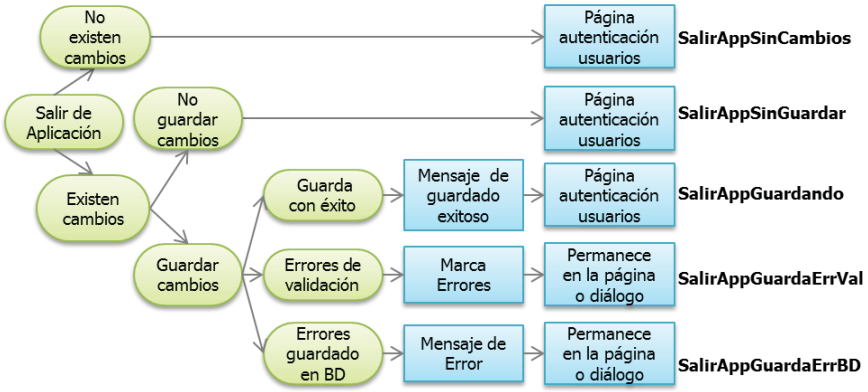

Fig. 1. Descomposición en escenarios del MU Abortar Operación a nivel de Aplicación

[5] http://www.grise.upm.es/sites/extras/7/Relacion_Guia_Arbol_MU_AO.pdf

A cada rama del árbol le dimos un nombre y la describimos usando diagramas de secuencia, por ejemplo, al escenario en que se solicita salir de la aplicación, existen cambios pendientes, se desea guardar los cambios pero se presenta un error de validación, lo llamamos *SalirAppGuardarErrVal* (**Fig. 1)**. El diagrama de secuencia asociado se puede ver en la **Fig. 2**.

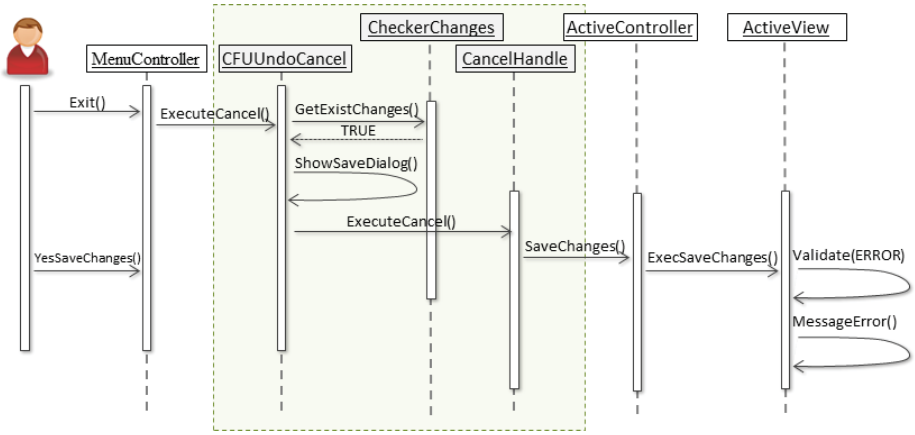

Fig. 2. Diagrama de secuencia para el escenario SalirAppGuardarErrVal

En el apéndice Web[7] se encuentran los tres árboles de escenarios y los diagramas de secuencia para cada rama. Obtuvimos en total 22 escenarios distintos. Nótese que a nivel de aplicación obtuvimos 5 escenarios, pero a nivel de operación el número de escenarios ascendió a 16, principalmente porque a diferencia del nivel de aplicación que solo tiene un posible origen (opción salir), a nivel de operación encontramos cuatro posibles orígenes: botón cancelar en una caja de dialogo, botón cancelar en un formulario, selección de otra opción de la aplicación y botón limpiar.

Durante la elaboración de los diagramas de secuencia identificamos un conjunto de responsabilidades asociadas a la funcionalidad del MU. Se requiere:

- Escuchar las acciones del usuario para determinar cuándo se solicita salir de la aplicación, cancelar una operación o cancelar un comando.
- Conocer en todo momento si existen o no cambios por guardar.
- En caso de que existan cambios en la aplicación, preguntar al usuario si desea o no guardar estos cambios y conocer la acción a seguir dependiendo de la respuesta recibida.
- Conocer cuál es el estado anterior y actual de la aplicación.
- Conocer como guardar los cambios pendientes de la aplicación independientemente de la operación o comando que se esté ejecutando.

Para cubrir las responsabilidades identificadas se definió un conjunto de componentes, que se pueden ver en la **Tabla 1** y que son usados en los diagramas de secuencia. En la **Fig. 2**. se ven sombreados los tres componentes relacionados con la funcionalidad de usabilidad.

[7] http://www.grise.upm.es/sites/extras/7/Escenarios_MU_AO.pdf

Tabla 1. Responsabilidades de los componentes del MU Abortar Operación

Componente	*Responsabilidad*
CheckerChanges	Mantiene e informa si existen cambios por guardar en la aplicación
CancelHandle	Guarda cambios en caso de que se aborten operaciones y deja al sistema en el estado adecuado (predecible y seguro para el usuario).
CFUUndoCancel	Recibe la solicitud de abortar operación (salir o cancelar), consulta al componente CheckerChanges si existen cambios, pregunta al usuario si desea guardar cambios y llama al método adecuado en cada caso.
HistorySteps	Mantiene y proporciona información del estado anterior y el estado actual del sistema.

Para generalizar los diagramas de secuencia usamos dos patrones reconocidos: el Patrón Modelo, Vista, Controlador y el patrón Fachada. La fachada se usa como punto de entrada a la funcionalidad de usabilidad, la vista se refiere a la interfaz con el usuario, el controlador recibe los eventos de usuario y hace las solicitudes a los componentes correspondientes, y el modelo maneja las reglas de negocio.

El conjunto de escenarios identificados describe toda la funcionalidad descubierta para el MU AO en los tres casos de estudio. Concluimos que el uso de la guía de educción resulta aún demasiado general para acometer la implementación de la funcionalidad de usabilidad. Al realizar las preguntas de la guía quedan demasiadas opciones abiertas sobre las que el desarrollador debe decidir, lo que significa posibles omisiones de escenarios de aplicación y demoras y/o errores en el diseño o implementación. El uso de los árboles de escenarios y de la descripción de cada escenario a través de diagramas de secuencia es una herramienta útil desde el momento de la educción de requisitos.

5 Solución Reutilizable Propuesta para la Funcionalidad de Usabilidad Abortar Operación

En esta sección describimos el proceso usado para obtener elementos reutilizables a nivel de diseño e implementación. Además de los escenarios de aplicación descritos en la sección 4, en los desarrollos finales de los tres casos de estudio encontramos coincidencias entre atributos, métodos y clases. A nivel de clases, encontramos que:

- Los tres diseños tienen una clase que funciona como fachada. Consecuencia de que los tres casos se construyeron pensando en ser soluciones reutilizables.
- En los tres casos existe una clase que tiene la responsabilidad de conocer si existen cambios pendientes. La clase tiene un solo atributo y tres métodos con funcionalidad similar. El atributo permite saber si hay cambios pendientes, y los tres métodos permiten modificar y consultar el valor del atributo.
- En los tres casos existe una clase que encapsula los métodos principales para responder a una solicitud, ya sea salir de la aplicación, o cancelar una operación o comando. Conoce cómo guardar cambios pendientes y cuál es el estado en el que debe quedar el sistema después de la solicitud.
- Hay otra clase que aparece en dos de los tres casos de estudio, encargada de mantener la información del estado anterior y actual del sistema. En el caso de VB

.NET no existe una clase con éste propósito porque utiliza características propias de la tecnología.

A nivel de atributos, encontramos que podían variar según la tecnología usada, pero también coincidían en varios puntos. Por ejemplo, en la clase que encapsula los métodos principales para atender una petición de AO, en los tres casos de estudio se usa un atributo para mantener la información necesaria para salvar los cambios, aunque se implementa de forma distinta en cada uno. Lo mismo sucede con la instrucción para cerrar un cuadro de dialogo o para salir de la aplicación.

También se encontraron coincidencias en otro tipo de decisiones de diseño, es el caso del uso de una fachada (CFUUndoCancel) definida como un Singleton, debido a que varios atributos en la solución debían ser únicos por sesión. El patrón Singleton asegura que solo existe una instancia de la clase y por lo tanto una única vía para actualizar sus datos. En el caso del MU AO hay datos que deben ser únicos por sesión: la existencia o no de cambios pendientes por guardar, el último estado de la aplicación, la forma de salvar los cambios y cuál es el cuadro de dialogo activo.

La principal diferencia que encontramos fue el manejo de los estados del sistema. En el caso de Java, por ejemplo, se utiliza una clase a nivel de servidor para mantener la información del estado anterior y actual del sistema, debido principalmente a la tecnología usada (JavaServerFaces). Mientras que en con VB .NET y PHP 5 no se usaron clases, sino variables de sesión.

Al finalizar el análisis de los tres diseños concluimos que muchos atributos, métodos y clases cumplían con las mismas responsabilidades y por lo tanto se podían unificar en un solo diseño que se puede ver en la **Fig. 3**. En cuanto a los elementos diferentes no son excluyentes, por el contrario se complementan para plantear un diseño que cubra todos los escenarios de aplicación descubiertos.

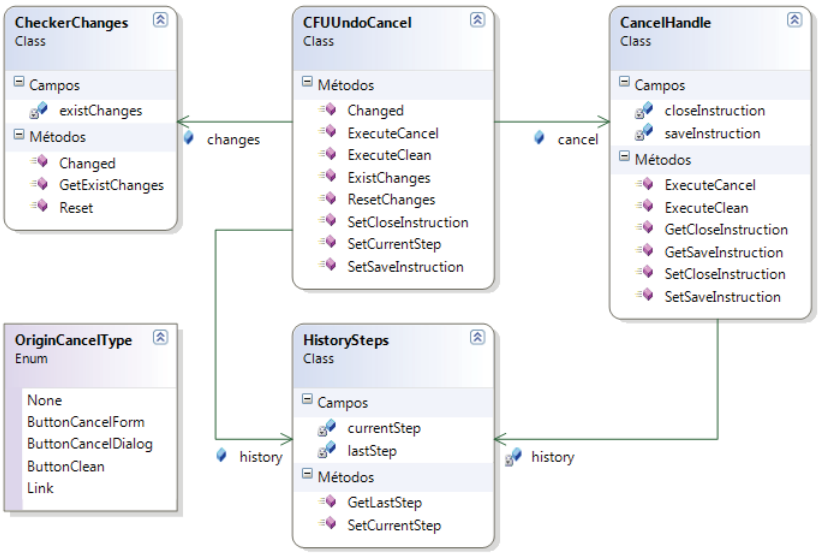

Fig. 3. Diagrama de clases unificado para el MU Abortar Operación

A nivel de programación, en los tres casos de estudio se encontró que una proporción significativa de la lógica está en el cliente. Al ser sistemas web, la implementación a nivel de cliente esta en el mismo lenguaje script para los tres casos: Javascript. Se hicieron algunas modificaciones para unificar los tres códigos script y se obtuvo un código único para los tres sistemas, que cubre todos los escenarios encontrados. En la **Tabla 2** se puede observar el código de uno de los componentes comunes: Checker Changes. En cuanto al código de la parte servidor, las diferencias están dadas por la tecnología utilizada, el diseño se modifica para adecuarse a la tecnología, aunque los componentes tengan las mismas responsabilidades.

Tenemos entonces una propuesta de diseño único y las implementaciones realizadas en tres lenguajes de programación distintos. La unión entre el diseño y la implementación la hemos formalizado como un patrón de programación. En el apéndice Web mostramos los patrones de programación para el MU Abort Operation en VB .NET[11], Java[12] y PHP 5[13].

Tabla 2. Código de la clase que mantiene información de si existen o no cambios pendientes

```
//Class: CheckerChanges
//Description: Mantiene información de cambios
function CheckerChanges(){
    this.ExistChanges = false; }
CheckerChanges.prototype.Changed = function() {
    this.ExistChanges = true; }
CheckerChanges.prototype.Reset = function() {
    this.ExistChanges = false;}
CheckerChanges.prototype.GetExistChanges = function()
{ return this.ExistChanges;}
```

Los patrones de programación que proponemos cubren todos los escenarios descubiertos para el MU AO, sabemos que en nuevas implementaciones aplicarán subconjuntos de ellos, razón por la cual no siempre se usará todo el código implementado. El código ejemplo asociado a los patrones se ha documentado para que sea fácil escoger las partes que sean útiles a un desarrollador dependiendo del alcance de la funcionalidad del MU que necesite implementar.

6 Evaluación

La solución propuesta ha sido aplicada en dos casos de estudio adicionales elaborados por diferentes desarrolladores con experiencia en programación. Los desarrolladores elaboraron los casos de estudio como parte de su Trabajo de Fin de Máster en la Universidad Politécnica de Madrid. Los casos de estudio fueron diferentes entre si, basados en requisitos reales para sistemas web y usando lenguajes de programación distintos: Java y VB .NET. Para la educción de requisitos los desarrolladores usaron

[11] http://www.grise.upm.es/sites/extras/7/PP_AO_VB_NET.pdf
[12] http://www.grise.upm.es/sites/extras/7/PP_AO_Java.pdf
[13] http://www.grise.upm.es/sites/extras/7/PP_AO_PHP.pdf

las mismas guías de educción que se usaron en este estudio [9] [17], pero adicionalmente utilizaron los artefactos reutilizables resultado de este trabajo:

- Escenarios de aplicación. En la educción de requisitos constituyen una guía para incluir escenarios que no son evidentes al usar la guía de educción, y evitar que se quede por fuera alguno de ellos a la hora de hablar con los usuarios. En el análisis sirven para entender la funcionalidad de usabilidad y como interactúa con el resto del sistema. Permite estimar la complejidad que se agregará al sistema.
- Patrón de diseño. Proporciona una descripción de los componentes requeridos para cumplir con las responsabilidades asociadas al Mecanismo de Usabilidad.
- Patrones de programación. Muestran una solución real utilizando el patrón de diseño con una tecnología específica. Proporcionan trozos de código reutilizables.

Según los desarrolladores, mediante un cuestionario de validación de la propuesta, concluyen que a pesar del tiempo que invirtieron inicialmente para entender la solución, el resultado final fue positivo, ya que los sistemas implementados cumplen con la funcionalidad de usabilidad, y que usarán la solución en futuras implementaciones.

Algunas de las cuestiones planteadas tenían que ver con la cantidad de esfuerzo adicional requerido en cada fase del desarrollo para incorporar la funcionalidad de usabilidad usando los patrones propuestos. En este sentido los desarrolladores encontraron que el uso de los patrones requiere en primera instancia un tiempo adicional considerable tanto para entender su funcionamiento como para incorporar los elementos a la solución. Los datos de esfuerzo variaron entre los desarrolladores debido entre otras cosas a que usaron diferentes modelos de desarrollo, uno usó el modelo tradicional en cascada y otro el modelo de desarrollo incremental.

También se pidió a los desarrolladores que propusiesen posibles mejoras a la solución y detallaran sus impresiones en cada momento del proceso de desarrollo. Se plantearon varias sugerencias entre las que se encuentran: mejor documentación del código y creación de aplicaciones demo, también propusieron e implementaron soluciones alternativas para el mismo diseño. Otras de sus conclusiones fueron: los escenarios son útiles y fáciles de usar una vez se entiende su propósito, tanto como complemento a las guías de educción, como en la etapa de análisis. El patrón de diseño fue el más sencillo de usar y fue aplicado al 100%.

Los desarrolladores también detallaron la forma en que usaron el código suministrado. Para la parte cliente se presentaron dos situaciones, una en la que el código fue usado como caja negra, es decir, pudieron usar el código sin modificarlo, y otra, en la que fue necesario hacer pequeñas adaptaciones. Las modificaciones fueron necesarias por incompatibilidades entre las tecnologías o versiones del lenguaje. En cuanto a la parte del servidor, no fue posible usar el código como caja negra, sin embargo, sí se pudo usar un esquema de copiar/pegar.

7 Discusión

Después de implementar el MU AO en los tres casos de estudio, encontramos que existían múltiples escenarios de aplicación dependiendo de las respuestas de los usuarios a las guías de educción. Debido a que las tres tecnologías utilizadas para

implementar los casos de estudio son orientadas a objetos, se usan clases, métodos y atributos, lo que permitió tener elementos similares de comparación. Durante el desarrollo de los casos de estudio se marcaron los elementos que tenían que ver con la funcionalidad de usabilidad lo que también facilitó la búsqueda de similitudes.

Como se ha expuesto en las secciones 4 y 5, el análisis de la funcionalidad del MU comienza identificando los escenarios de aplicación, al unir todos los escenarios de aplicación obtuvimos una descripción detallada de la funcionalidad que debía cubrir la solución reutilizable que fuésemos a proponer.

Del análisis de las responsabilidades necesarias para cumplir con la funcionalidad de usabilidad en cada caso estudio, resultaron un conjunto de responsabilidades comunes para cubrir todos los escenarios de aplicación del MU. Se definieron componentes para cumplir con estas responsabilidades. En algunos casos, el resultado fue un componente para cubrir una responsabilidad y en otros casos, un componente para dos o tres responsabilidades similares. Con base en los componentes identificados se propone un diseño que busca ser reutilizable. El diseño encapsula el conocimiento necesario para cubrir cualquiera de los escenarios del MU.

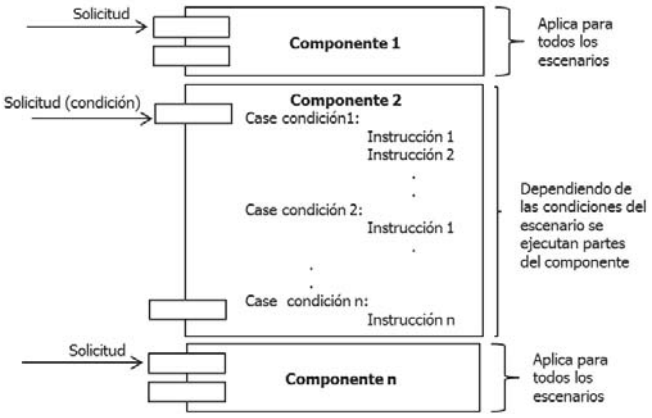

Fig. 4. Estructura de los componentes propuestos para el MU Abortar Operación

Cada componente se diseñó como una clase. La funcionalidad de un escenario no es cubierta por un componente específico, si no por la interacción de varios componentes y la interacción esta descrita por el diseño de clases. Como se muestra en la **Fig. 4** en algunos de los componentes no fue necesario hacer ninguna diferenciación relacionada con condiciones dadas por los escenarios, el código cumple con su responsabilidad asociada sin hacer referencia a los escenarios. En otros casos fue necesario hacer diferenciaciones según algunas condiciones dadas por los escenarios por lo tanto existirán partes del código que no se ejecuten.

Debido a que los sistemas software desarrollados son aplicaciones web, algunos de los componentes identificados son para la parte cliente y otros para la parte servidor. La parte del cliente se implementó usando JavaScript en los tres casos de estudio dando como resultado código similar y comparable, pero la implementación de los componentes de la parte servidor dependen del lenguaje y su comparación llego solo a nivel de diseño. Las “piezas de código”, que son iguales en los tres casos,

corresponden a la capa cliente, están implementadas en lenguaje JavaScript y cubren todos los escenarios documentados. La encapsulamos en un solo archivo y consideramos que es un primer paso para la construcción de una biblioteca de componentes para la usabilidad.

8 Conclusiones

En este artículo, detallamos la obtención y validación de una solución reutilizable para la implementación de la funcionalidad de usabilidad Abortar Operación. A través de implementaciones reales encontramos que existen elementos comunes que pueden generalizarse como artefactos reutilizables para diferentes fases del proceso de desarrollo. Los resultados de este estudio están limitados a aplicaciones web altamente interactivas y desarrolladas bajo el paradigma orientado a objetos.

La funcionalidad que abarca la solución reutilizable está limitada a los escenarios de aplicación encontrados en los casos de estudio, es posible que al desarrollar nuevos casos de estudio surjan nuevos escenarios de aplicación. La documentación de los escenarios a través de diagramas de secuencia es una herramienta útil desde el inicio del ciclo de desarrollo. En la fase de educción y especificación de requisitos permite verificar que se tengan en cuenta todos los casos posibles en los que puede aplicar la funcionalidad de usabilidad. En la fase de análisis permite evaluar la forma en que se verán afectadas las funcionalidades del sistema software y proporciona una idea clara de cómo realizar su implementación.

El patrón de diseño propuesto encapsula la funcionalidad necesaria para cubrir con las responsabilidades descritas en los escenarios de aplicación. Será necesario modificar el diseño según la tecnología en que se implemente, aunque encontramos que el código para la parte cliente puede ser común a cualquier aplicación web. Los patrones de programación serán más útiles cuando se usen las mismas versiones del lenguaje de programación. Formalizar los resultados como un patrón de diseño permite describir la solución a nivel más general y los patrones de programación proporcionan código útil o en su defecto una guía para su implementación en otros lenguajes de programación.

La aplicación en otros casos de estudio llevada a cabo por desarrolladores distintos, permitió encontrar fallos en la documentación y la necesidad de proporcionar aplicaciones demo adicionales al código. También permitió observar que muchos de los artefactos reutilizables proporcionados fueron útiles y que aunque en una primera implementación se requiera un tiempo adicional para el entendimiento y aprendizaje del patrón, son potencialmente reutilizables en nuevas implementaciones. Como es inherente a la definición de patrón, este estará en mejoramiento continuo. En cada nueva implementación se podrán incluir nuevas funcionalidades, mejoras al diseño y código en otros lenguajes o versiones.

Como continuación de este trabajo, aplicaremos el mismo método expuesto para obtener soluciones reutilizables para otras funcionalidades de usabilidad con alto impacto en el diseño. Buscaremos obtener un repositorio de patrones y componentes para facilitar la implementación de funcionalidades de usabilidad y/o establecer un procedimiento para facilitar su obtención.

Agradecimientos. Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España a través de los proyectos Tecnologías para la Replicación y Síntesis de Experimentos en IS (TIN2011-23216) y Go Lite (TIN2011-24139).

Referencias

1. ISO Std. 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals. Part 11: Guidance on Usability (1998)
2. Constantine, L., Lockwood, L.: Software for use: A practical Guide to the Models and Methods of Usage-centered Design. Addison Wesley (1999)
3. Donahue, G. M.: Usability and the Bottom Line. IEEE Software, 18, 22-30 (2001)
4. Nielsen, J.: Return on Investment for Usability. Alertbox. http://www.useit.com (2003)
5. Chrush, M.: The Whiteboard: Seven Great Myths of Usability. Interactions 7, 13-16 (2000)
6. Juristo, N., Moreno, A. M., Sanchez-Segura, M.: Analysing the impact of usability on software design. J. System and Software 80:9, 1506-1516 (2007)
7. Perry, D., Wolf, A.: Foundations for the study of software architecture. ACM Software Engineering Notes 17:4, 40-52 (1992)
8. Bass, L., John, B.E.: Linking usability to software architecture patterns through general scenarios. Journal of Systems and Software 66:3, 187-197 (2003)
9. Juristo, N., Moreno, A. M., Sanchez-Segura, M.: Guidelines for Eliciting Usability Functionalities Software Engineering. IEEE Transactions on Software Engineering 33, 744-758 (2007)
10. Bushmann, F., Meunier, R., Rohnert, H.,Sommerlad, P.: Pattern - Oriented Software Arquitecture. A system of patterns. John Wiley & sons Ltd., England (1996)
11. Bass, L., Jhon, B.E.: Supporting usability through software arquitecture. IEEE Computer 34(10), 113-115 (2001)
12. Bass, L., Bonnie, E., Kates, J.: Achieving Usability through Software Architecture. Technical Report CMU/SEI-2001-TR-005. Software Eng. Inst., Carnegie Mellon Univ. (2001)
13. Bonnie E., Bass, L., Golden, E., Stoll, P.: A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns. Proceedings of EICS. Pittsburgh, PA, USA (2009)
14. STATUS Project. Software Architecture that supports Usability, http://www.grise.upm.es/rearviewmirror/projects/status/index.html (2004)
15. Campos G., Acuña, S. T., Macías, J. A.: Implementación de Propiedades de Usabilidad con Impacto en el Diseño Mediante la Programación Orientada a Aspectos. Actas del XI Congreso Internacional de Interacción Persona-Ordenador, Valencia, España (2010)
16. Pinto, M., Fuentes, L.: Aspect-Oriented Modeling of Quality Attributes. ECSA, 2008. LNCS Vol. 5292, pp. 334-337 (2008)
17. Juristo, N., Moreno, A.,Sanchez-Segura, M.: Usability Elicitation Patterns (USEPs), http://www.grise.upm.es/sites/extras/2/ (2006)