Medición Temprana de la Usabilidad en Ambientes OO-Method

Jose Ignacio Panach Navarrete

Tesina del Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

Dirigida por el **Dr. Óscar Pastor López**

ÍNDICE

1. INTRODUCCIÓN

1.1. Propósito del documento

La usabilidad es una de las características de la calidad del software que más ha sido tratada por la comunidad de la Interacción Persona-Ordenador o HCI (*Human Computer Interaction).* Dentro de esta comunidad se han planteado una amplia variedad de propuestas para medir la usabilidad de las aplicaciones. Son propuestas basadas principalmente en la aplicación final y en el usuario de esta aplicación. En este documento se presenta la instanciación de un método de evaluación de la usabilidad para OO-Method, un entorno de generación automática de código a partir de Modelos Conceptuales. El método de evaluación utilizado intenta acercar los métodos de la comunidad HCI a los de la comunidad de la Ingeniería del Software o SE*(Software Engineering)*. Este método de evaluación de la usabilidad está pensado para aplicarlo en las primeras etapas de construcción del software. El propósito de este trabajo es el de proporcionar un mecanismo para evaluar la usabilidad del sistema a partir de su Modelo Conceptual, sin utilizar la aplicación final ni la participación del usuario.

1.2. Objetivo del proyecto

Tal y como aparece definido en la ISO 9126-1 [19], la usabilidad es una de las características que definen la calidad de un sistema software. Las otras características que conforman la calidad son la *Funcionalidad*, *Fiabilidad*, *Eficiencia*, *Mantenibilidad* y *Portabilidad*. Para asegurar un software de calidad, es necesario definir unas métricas que guíen al analista a la hora de implementar las aplicaciones. Sin la existencia de métricas no se puede saber con certeza si una aplicación es o no usable. Además, estas métricas son útiles para saber cuánto de usable es un sistema. En la propia ISO 9126-1 existen métricas para obtener el valor de la usabilidad de los sistemas.

Además de la ISO 9126-1, dentro de la comunidad HCI son muchos los autores que han propuesto métodos de evaluación de la usabilidad, como es el caso de Shneiderman [40] o Dix [8]. No existe un método de evaluación que sea el mejor en todos los casos, pero los más populares y extendidos son los realizados mediante test de usuarios [38]. En estos test deben participar los que serán los usuarios del sistema y se deben llevar a cabo sobre la aplicación final. Estos test requieren inversión de recursos y tiempo, tanto por parte del analista, como del usuario.

Normalmente estos test de usuario se suelen desarrollar bajo la tutela de un experto en usabilidad dentro de un laboratorio construido específicamente para la evaluación de la usabilidad. Estos laboratorios deben estar equipados con varias cámaras de video, micrófonos y programas para almacenar todas las acciones que haga el usuario en el ordenador. Gracias a este laboratorio se pueden almacenar todas las acciones que haga el usuario en el sistema, además de almacenar sus expresiones y comentarios mientras se hace la prueba para saber cómo reacciona el usuario ante el uso del sistema.

Los test de usuario no están basados en todo el sistema, sino que se centran en una serie de tareas escogidas especialmente dependiendo de los aspectos que se deseen medir. Por lo tanto, además de tener preparado el laboratorio de usabilidad es necesario definir una serie de tareas que debe hacer el usuario. La selección de estas tareas las lleva a cabo el experto en usabilidad.

Una vez el usuario ha realizado todas las tareas, debe por último rellenar un formulario desarrollado también por el experto en usabilidad. Mediante este cuestionario se pretende recabar la opinión del usuario sobre el sistema después de haber interactuado con él. Estas preguntas se hacen con el objetivo de medir la usabilidad del sistema en base a las tareas escogidas. Son preguntas sin ningún tipo de contenido técnico, sino preguntas comprensibles por cualquier usuario sin conocimientos en informática.

Por lo tanto para llevar a cabo un test de usuario hacen falta los siguientes elementos: la aplicación terminada, un usuario, un experto en usabilidad, un laboratorio de usabilidad, la definición de tareas a realizar por el usuario, un formulario para obtener la opinión del usuario sobre la usabilidad del sistema. Tal y como se puede apreciar son muchos los requisitos necesarios para llevar a cabo el test de usuarios y es difícil que se puedan dar todos ellos. Además, el mantenimiento del laboratorio de usabilidad es costoso y no está al alcance de todos los desarrolladores de sistemas. Todo esto hace que sean pocas las aplicaciones implementadas que se sometan a un test de usuario antes de entrar en sus entornos de producción.

La complejidad y el alto coste que trae consigo el test de usuario ha propiciado que ciertos autores del ámbito de la Ingeniería del Software o SE *(Software Engineering)* hayan propuesto una evaluación de la usabilidad temprana. Entre estos autores se encuentran, Folmer [14], Bass [3] y Keinonen [23]. Todos ellos proponen una evaluación de la usabilidad temprana, antes de que el sistema esté implementado.

Siguiendo la tendencia de evaluación en las fases tempranas de construcción del software y el paradigma MDA [31], el presente documento muestra un método de evaluación de la usabilidad a partir del Modelado Conceptual. El método propuesto abstrae los criterios de medición propuestos en la ISO 9126-1 a un mayor nivel de abstracción que el propuesto en esta ISO. Este trabajo define una serie de métricas dentro del espacio del problema, es decir, las métricas utilizadas están basadas en primitivas conceptuales correspondientes al nivel PIM (*Platform Indepent Model)* de MDA, no en aspectos del sistema final, como plantea la comunidad HCI.

Con respecto al test de usuarios planteado por la comunidad HCI, la ventaja principal de la propuesta de medición de la usabilidad a nivel de Modelado Conceptual está clara: es una evaluación muy poco costosa, tanto de recursos como de tiempo. A continuación se detallan todas las ventajas de la evaluación basada en los Modelos Conceptuales:

- La aplicación no debe estar terminada para hacer la evaluación de la usabilidad. Esto implica que se pueda mejorar fácilmente la usabilidad del sistema a partir de los resultados de la evaluación, ya que el sistema aun no se ha construido ni está implantado. Si la evaluación se hace mediante test de usuarios, la aplicación ya debe estar terminada y los cambios para mejorar la usabilidad del sistema serán mucho más costosos, ya que implicarán cambios de análisis, diseño e implementación del sistema.
- No es necesaria la participación del usuario en el proceso de evaluación. Al hacerse la evaluación sobre los Modelos Conceptuales, el usuario no debe realizar ninguna tarea con el sistema ni rellenar ningún formulario. Este hecho favorece la agilidad de la evaluación, ya que la disponibilidad del usuario marca en muchas ocasiones el ritmo de la evaluación de los sistemas si la evaluación se realiza mediante test de usuarios.
- La evaluación no requiere prácticamente ningún recurso. La medición de la usabilidad se hace en base a las primitivas de modelado y esta medición la realiza automáticamente la herramienta encargada de dibujar los Modelos Conceptuales. Por el contrario, en el caso de los test de usuarios, es necesario un gran número de

recursos humanos (expertos en usabilidad y usuarios) y materiales (laboratorio de usabilidad), lo que encarece el proceso de evaluación.

- El analista no debe conocer cómo se lleva a cabo el proceso de evaluación. El método para evaluar el Modelo Conceptual lo lleva a cabo el modelador de sistemas, por tanto el analista no debe conocer los fundamentos del método.

Tal y como se puede apreciar, son muchas las ventajas de aplicar la evaluación de la usabilidad en la fase de Modelado Conceptual. Para poder realizar esta evaluación temprana es necesario utilizar un Modelo de Usabilidad. El Modelo de Usabilidad es una forma abstracta de representar los aspectos (atributos) que influyen en la usabilidad. A partir de este modelo se pueden definir las métricas para medir la usabilidad de los sistemas. Este trabajo ha tomado como Modelo de Usabilidad el publicado en los trabajos de Abrahao [2][11][1]. Este Modelo de Usabilidad parte de los conceptos definidos en la ISO 9126-1 y los amplía con el de otros autores que han trabajado en el mismo ámbito de la usabilidad, como Bastien y Scapin [4] o Nielsen[27]. Esto hace que sea un modelo muy completo.

Dentro del Modelo de Usabilidad hay dos componentes diferenciadas: subcaracterísticas y atributos.

- Subcaracterísticas: cada subcaracterística representa un aspecto general relacionado con la usabilidad.
- Atributos: son artefactos medibles definidos dentro de subcaracterísticas. Cada atributo lleva asociado un mecanismo de medición del aspecto de usabilidad que defina. Una subcaracterística engloba varios atributos.

La descomposición de subcaracterísticas en atributos es necesaria porque los conceptos que engloba cada una de las subcaracterísticas de la usabilidad son demasiado complejos para medirlos directamente sobre ellas. Es necesario hacer una especialización de conceptos por medio de los atributos. Cada atributo solo mide un aspecto de la subcaracterística de forma no ambigua y por lo tanto se pueden definir métricas para calcular el valor de usabilidad de cada uno de los atributos.Puede que existan atributos que al aumentar su valor de usabilidad hagan disminuir el valor de usabilidad de otros, por ejemplo al mejorar la usabilidad del atributo *Autodescripción*, empeora la del atributo *Densidad de información*. Esto es debido a que cuanta más información utilicen los componentes de la ventana para describirse, más cantidad de información se mostrará en la interfaz.

Tanto el Modelo de Usabilidad utilizado, como el método de evaluación de la usabilidad a partir de este modelo, pueden ser utilizados por cualquier método de producción de software que utilice distintos niveles de abstracción para representar el sistema. La definición del método de evaluación es lo suficientemente abstracta como para poder ser utilizada por cualquier método de producción de software.

De todos los métodos de producción de software a partir de Modelos Conceptuales, este documento se ha centrado en OO-Method [32]. OO-Method es un método de generación automática de sistemas a partir de Modelos Conceptuales que sigue las directrices de MDA [31]. Tal y como se plantea en el paradigma MDA, OO-Method distingue entre el espacio del problema (Modelos Conceptuales) y el espacio de la solución (implementación en código). OO-Method dispone de un Compilador de Modelos donde se define la estrategia de generación de código. Gracias a este Compilador de Modelos, a partir de un conjunto de Modelos Conceptuales, OO-Method es capaz de generar aplicaciones en C#, Java, ASP .NET y JSP.

La principal ventaja de utilizar OO-Method con respecto al resto de métodos de producción de software existentes es que los propios Modelos Conceptuales, gracias al Compilador de Modelos de OO-Method, generarán el sistema final sin que el analista tenga que escribir ninguna línea de código. Por lo tanto el medir la usabilidad utilizando las primitivas

conceptuales de OO-Method, es una garantía de que se está midiendo la usabilidad del sistema final.

El Modelo de Usabilidad utilizado para esta medición tiene una amplia variedad de subcaracterísticas y atributos y no todos ellos son medibles en la fase de Modelado Conceptual del sistema. Hay muchos atributos que son meramente subjetivos del usuario que utilizará el sistema y por tanto no pueden ser medidos automáticamente por el modelador. En OO-Method, este es el caso por ejemplo del atributo llamado *Atracción subjetiva*, que mide el grado en que al usuario le gusta el sistema. Además de estos atributos subjetivos, hay otros atributos del Modelo de Usabilidad que no tiene sentido medir porque con la estrategia de generación de código del Compilador de Modelos siempre tendrán el mismo valor de usabilidad. En OO-Method, este es el caso por ejemplo del atributo, *Soporte de cancelación*, que indica la posibilidad de cancelar la ejecución de un servicio mientras se ejecuta. El Compilador de Modelos ya se encarga de dotar a los servicios con esta característica automáticamente, no es necesario que el analista modele esta funcionalidad a nivel conceptual. Por lo tanto su valor de usabilidad será el mismo en todos los sistemas y no tiene sentido el medirlo. La categorización de los atributos entre estos tres tipos depende de la expresividad del método de producción escogido. Habrá métodos que dejen más aspectos de modelado al analista y habrá otros que dejen más decisiones al Compilador de Modelos y menos al analista.

De todos los tipos de atributos que forman el Modelo de Usabilidad, este trabajo se va a centrar en aquellos medibles en los Modelos Conceptuales de OO-Method. Por lo tanto, aquellos atributos subjetivos o que siempre obtengan el mismo valor de usabilidad debido a la estrategia de generación de código del Compilador de Modelos, no se tendrán en cuenta. Este hecho hace que el valor de usabilidad obtenido no sea el mismo valor que se obtendría al realizar un test de usuario sobre la aplicación final, ya que solo tiene en cuenta parte de los atributos del Modelo de Usabilidad. Sin embargo, el número de atributos medibles a partir de los Modelos Conceptuales es lo suficientemente amplio como para predecir mediante ellos el valor de usabilidad de un sistema. Aun así, el método de evaluación propuesto no excluye la utilización de test de usuarios. Una vez se consigan valores de usabilidad buenos mediante el método de evaluación propuesto, se pueden realizar test de usabilidad que midan los atributos subjetivos.

La instanciación del método de usabilidad planteado por Abrahao et al [2][11][1] dentro de un entorno de desarrollo de software dirigido por modelos, se ha definido en tres fases:

1. Definición de métricas: sobre cada uno de los atributos que forman el Modelo de Usabilidad, se definen una o varias fórmulas para calcular el valor de usabilidad de ese atributo. Estas fórmulas están basadas exclusivamente en primitivas de los Modelos Conceptuales y su valor se calcula de forma automática mediante la herramienta de construcción de los Modelos Conceptuales (modelador).
2. Definición de indicadores: mediante estos indicadores se les asigna un significado al valor numérico obtenido a través de las métricas. Los indicadores son los rangos entre los que puede oscilar el valor de las métricas para considerar ese valor como usable o no usable. Estos indicadores se han obtenido de guías de usabilidad [21], heurísticos [27] y criterios ergonómicos [4].
3. Agrupación: en este paso se agrupan los valores de usabilidad de los atributos obtenidos mediante los indicadores. Esta agrupación se utiliza para asignar el valor de usabilidad a las subcaracterísticas a partir de los valores de usabilidad de los atributos que las componen.

De estos tres pasos, el único que depende del método de producción de software escogido es el de definición de métricas. A pesar de la generalidad de la propuesta de medición, las métricas son exclusivas del método escogido. Estas métricas están basadas en primitivas conceptuales de modelado y cada método de producción de software dispone de unas primitivas distintas, ofreciendo más o menos posibilidades de modelado. Por lo tanto las métricas se deben adaptar al método de producción de software sobre el que se vaya a realizar

la medición de usabilidad. Al igual que en el caso de la clasificación de los atributos del Modelo de Usabilidad, la definición de las métricas del método de evaluación es totalmente dependiente del método de producción de software elegido. Estas dos características son las únicas dependientes del método de producción, el resto pueden ser generalizables a cualquiera.

Mediante estos tres pasos se puede obtener el valor de usabilidad de cada una de las subcaracterísticas y de todo el sistema. Una vez se ha hecho el análisis de usabilidad, se muestra al analista una lista con aquellos atributos que no han obtenido valores muy buenos de usabilidad para que el analista mejore estos valores si lo considera oportuno. Además de mostrar qué atributos de usabilidad se deben mejorar, la herramienta de medición de la usabilidad proporciona una pequeña descripción de cómo mejorar estos valores de usabilidad, ya que el analista no tiene porqué conocer qué primitivas afectan a cada uno de los atributos.

Una de las contribuciones de este trabajo es la de llevar a la práctica el proceso de medición de la usabilidad a partir de Modelos Conceptuales basado en los trabajos de Abrahao, pero ésta no es la única contribución. A la hora de estudiar otros Modelos de Usabilidad propuestos en la literatura, se descubrió que el Modelo de Usabilidad utilizado en el método de evaluación, no contemplaba todos los atributos existentes en el ámbito de la usabilidad. Otra de las contribuciones de este trabajo es el de mejorar el Modelo de Usabilidad con nuevos atributos. Para mejorar el Modelo de Usabilidad, este trabajo se ha basado en unos patrones de usabilidad definidos en un proyecto europeo llamado STATUS *(SofTware Arquitectures That support USability)* [42], donde se definen una serie de patrones de usabilidad que se deben aplicar desde la fase de captura de requisitos hasta la fase de implementación.

La elección de los patrones de STATUS es adecuada para mejorar el Modelo de Usabilidad utilizado sobre OO-Method por los siguientes motivos:

- Contienen un gran número de aspectos a tener en cuenta para mejorar la usabilidad de los sistemas.
- Estas mejoras no sólo afectan a la interfaz del sistema, sino a toda su arquitectura. Por lo tanto su definición puede ser abstraída para incorporarla fácilmente al Modelado Conceptual de OO-Method.
- Estos patrones están descritos independientemente de cualquier método de producción de software, por lo tanto pueden ser adaptados a OO-Method.
- Aunque la incorporación al Modelo de Usabilidad de los nuevos atributos derivados de estos patrones están pensados para medirlos en OO-Method, son conceptos lo suficientemente abstractos como para poder ser medidos en cualquier otro método de producción de software.

Ahora bien, no solo basta con modificar el Modelo de Usabilidad para incorporar los nuevos atributos derivados de STATUS. Muchos de los atributos derivados de los patrones de STATUS no están soportados por OO-Method y por lo tanto no hay primitivas conceptuales para representarlos. Al no existir primitivas conceptuales, no hay forma de definir métricas para medir la usabilidad de estos atributos. Por lo tanto, la incorporación de los nuevos atributos al Modelo de Usabilidad implica cambios en el Modelado Conceptual de OO-Method.

La incorporación de nuevas primitivas de modelado lleva también asociado un cambio en el Compilador de Modelos, que se debe modificar para que interprete las nuevas primitivas de modelado y genere el código que represente su funcionalidad. Todos estos cambios son complejos, ya que afectan a todo el proceso de generación de código OO-Method. Por ese motivo este trabajo solo se ha centrado en una familia de patrones de STATUS, así se puede

hacer un barrido detallado a lo largo de todo el proceso de OO-Method, explicando los cambios que se deberían incorporar en cada una de sus fases.

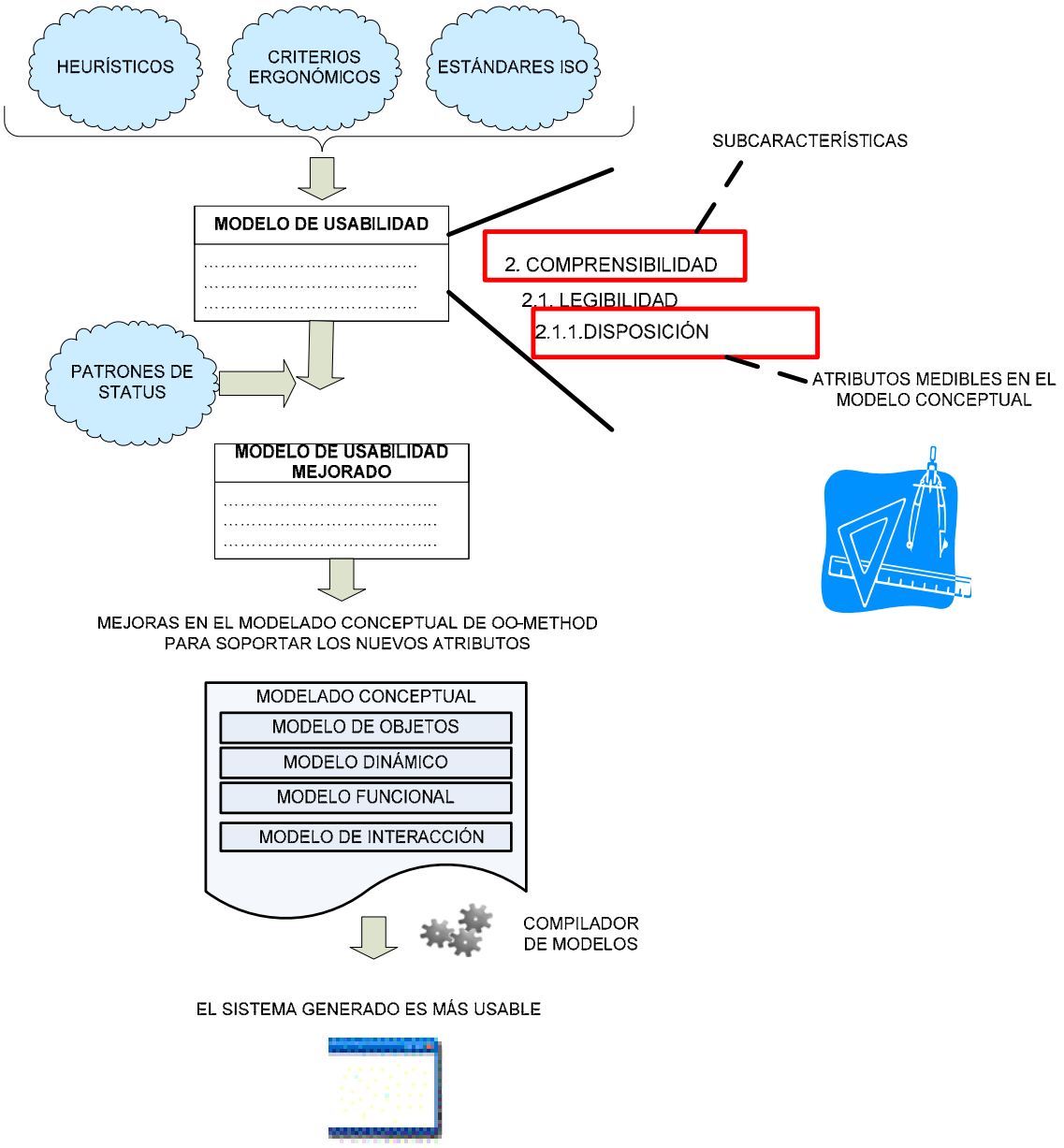

Figura 1. Proceso seguido a lo largo del documento para conseguir aplicaciones usables

Como resumen de todo el trabajo presentado en el documento, la Figura 1 muestra gráficamente el proceso seguido para conseguir aplicaciones más usables de las que actualmente OO-Method genera. En dicha figura se puede apreciar cómo el Modelo de Usabilidad obtenido a partir de heurísticos, criterios ergonómicos y estándares ISO, se divide en subcaracterísticas y atributos. En la figura se muestra como ejemplo la subcaracterística *Comprensibilidad* y el atributo *Disposición*. El proceso de medición de la usabilidad se hace sobre los atributos del Modelo de Usabilidad medibles a través de los Modelos Conceptuales. A partir del Modelo de Usabilidad existente, se añaden nuevos atributos obtenidos a partir de los patrones de usabilidad de STATUS. Este cambio implica que se deban añadir nuevas primitivas de modelado en los Modelos Conceptuales de OO-Method (*Modelo de Objetos,*

Modelo Dinámico, Modelo Funcional y *Modelo de Interacción*) y cambios en el Compilador de Modelos para soportar la generación de código que implemente la funcionalidad de las nuevas primitivas. Al final de todo este proceso, se obtienen aplicaciones mucho más usables de las que ya genera actualmente OO-Method.

1.3. Estructura del documento

1.3.1. Organización

- La sección 2 presenta un estado del arte de otros autores que han definido otros Modelos de Usabilidad y otros métodos de evaluación de la usabilidad, tanto en las fases tempranas como a través de la aplicación final.
- La sección 3 explica qué es una Tecnología de Transformación de Modelos. Como ejemplo se presenta OO-Method, la Tecnología de Transformación de Modelos sobre la que va a estar basada la propuesta de evaluación de la usabilidad.
- La sección 4 presenta el método de Abrahao et al para la evaluación de la usabilidad a partir de los Modelos Conceptuales. Además se presenta, como aportación de este trabajo, las tres fases necesarias para llevar a cabo la evaluación en un entorno de generación de código dirigido por modelos. Estos tres pasos se corresponden con la definición de métricas, indicadores y la estrategia de agrupación necesarias para la evaluación de los atributos del Modelo de Usabilidad.
- La sección 5 demuestra que las fórmulas utilizadas para definir las métricas del método de evaluación de la usabilidad son fórmulas válidas.
- La sección 6 plantea una serie de mejoras en el Modelo de Usabilidad. A partir de una serie de patrones de usabilidad, se incorporan nuevos atributos al Modelo de Usabilidad.
- La sección 7 muestra los cambios que se deberían llevar a cabo en los Modelos Conceptuales de OO-Method para poder medir los nuevos atributos incorporados en el punto anterior en el Modelo de Usabilidad.
- La sección 8 explica los cambios que se deberían hacer en el Compilador de Modelos para soportar la generación del código que implemente la funcionalidad de las nuevas primitivas de Modelado Conceptual definidas en el punto anterior.
- La sección 9 presenta un caso de estudio donde se aplica el método de evaluación de la usabilidad. Además se presenta el uso práctico de las nuevas primitivas de modelado incorporadas a OO-Method gracias a los patrones de usabilidad.
- Por último la sección 10 recoge las conclusiones del documento.

1.3.2. Convenciones tipográficas y notación empleada

Para facilitar la lectura y la comprensión de este documento, se han adoptado las siguientes convenciones:

- Cualquier referencia a un texto incluido en una figura, irá en cursiva, como *Atributos* de la Figura 1.
- Se utiliza la **Negrita** para remarcar aspectos importantes.
- Al definir un término, éste irá en cursiva y empezando por mayúsculas, por ejemplo: *Modelo de Interacción.*
- Palabras extranjeras en *cursiva*.

2. ESTADO DEL ARTE

Tal y como está definido en la ISO 9126-1 [19], la usabilidad es la capacidad del producto software de ser entendido, aprendido y usado por el usuario bajo condiciones específicas. Uno de los primeros trabajos basados en la usabilidad y previo a la definición de la ISO, fue el trabajo de Bastien y Scapin [39], los cuales definieron una serie de criterios ergonómicos que debían seguir las aplicaciones para considerarlas como usables. En este trabajo se elaboró una lista de criterios de usabilidad para describir y clasificar los problemas de usabilidad encontrados por expertos en la evaluación de tareas. La lista de criterios ergonómicos fue elaborada para que la utilizaran personas no expertas en el ámbito de la usabilidad y que redujeran costes en la evaluación de la usabilidad (la no presencia de expertos reduce los costes de la evaluación). Dicha lista fue validada mediante la experimentación, donde se probó que los criterios eran suficientemente detallados y ejemplificados para que los evaluadores de la usabilidad del sistema los comprendieran. Mediante este experimento se mejoraron las definiciones de los criterios ergonómicos y se añadieron nuevos ejemplos prácticos. Mediante otras experimentaciones se concluyó que dichos criterios son útiles para organizar la evaluación de la usabilidad, para aumentar la completitud de la evaluación y para disminuir el número necesario de expertos para detectar carencias de usabilidad en las aplicaciones.

Los grandes grupos en los que se reúnen los criterios ergonómicos son:

- Orientación: se refiere a la capacidad del sistema de avisar, orientar, informar, instruir y guiar a los usuarios a través de sus interacciones con el sistema.
- Carga de trabajo: son todos aquellos elementos de la interfaz que juegan un rol en la reducción de carga cognitiva y en el trabajo que tiene que hacer el usuario en el sistema, aumentando la eficiencia en el diálogo del usuario con el sistema.
- Control explícito: se refiere tanto al procesamiento explícito de acciones de usuario como al control que tienen los usuarios en el procesamiento de sus acciones
- Adaptabilidad: se refiere a la capacidad del sistema de contextualizarse según las necesidades del usuario y de sus preferencias.
- Gestión de errores: se refiere a la capacidad del sistema de prevenir o reducir errores y recuperarse de ellos cuando ocurran.
- Consistencia: se refiere a la forma en la que el diseño de la interfaz se mantiene en contextos similares, a la vez que es diferente cuando se aplica a contextos distintos.
- Significado de los códigos: cualifica la relación entre un término y su símbolo y referencia. Los códigos y los nombres son significativos para los usuarios cuando hay una relación semántica fuerte entre los códigos y los ítems a los cuales se refieren.
- Compatibilidad: se refiere a la combinación entre características de usuario y características de las tareas por un lado, y la organización de las salidas, entradas, y diálogo de una aplicación por otro lado.

Todos estos grupos se subdividen en otros subgrupos formando una estructura jerárquica de criterios ergonómicos. En resumen, las principales características de los criterios ergonómicos son las siguientes:

- Completitud: todos los problemas de usabilidad están cubiertos por alguno de los criterios de usabilidad definidos.
- Independencia: para cada uno de los problemas de usabilidad que se detecten hay una clasificación biunívoca que soporta una independencia del proceso.
- Aplicación global: en la experimentación que acompaña a esta propuesta todos los criterios fueron utilizados y ninguno fue rechazado por inapropiado, cuestionable, o dependiente del contexto.

Además de todas las ventajas que aporta esta propuesta a la comunidad HCI, también se plantean una serie de problemas con el uso de estos criterios. Uno de estos problemas es que el método no especifica un mecanismo para realizar la evaluación. Se deberían incluir descripciones sobre cómo explorar las interfaces del sistema para realizar la evaluación. Otro inconveniente es que estos criterios ergonómicos no se presentan como exclusivos en la evaluación de la usabilidad. Es decir, la evaluación basada en los criterios ergonómicos puede ir acompañada también de otros mecanismos como cuestionarios, entrevistas, etc.

Otra de las propuestas centradas en la teoría empírica y en la experimentación, al igual que el trabajo de Bastien y Scapin, es la de Dromey [9]. El trabajo de Dromey abarca todas las características para obtener un software de calidad, mientras que el trabajo de Bastien y Scapin está solo centrado en la característica usabilidad. La propuesta de Dromey proporciona un conjunto de axiomas y modelos de calidad necesarios para construir modelos comprensibles para generar un producto software de calidad. Cada uno de los atributos de calidad que utiliza en su modelo se corresponde bien con *comportamientos* o bien con *usos*. El *comportamiento* es algo que el software presenta cuando se ejecuta bajo la influencia de un conjunto de entradas. El *uso* es algo que diferentes usuarios hacen con o para el software.

Para la construcción del modelo para la calidad del producto software, Dromey propone dos estrategias distintas: *bottom-up* y *top-down*. La primera estrategia consiste en tratar de enumerar propiedades concretas y clasificarlas según la característica software a la que pertenezca. Una vez hecho esto, en el siguiente nivel se trata de enumerar características software que caractericen cada comportamiento y cada uso. La estrategia *top-down* es una derivación o descomposición para caracterizar o definir propiedades abstractas en términos de comportamientos subordinados, usos y características software.

Los principios que utiliza para guiar la descomposición y la derivación son:

- Un comportamiento puede ser descompuesto y de ahí definido en términos de propiedades subordinadas que pueden ser descritas bien como comportamientos o bien como características software.
- Un uso puede ser descompuesto y de ahí definido en términos de propiedades subordinadas que pueden ser descritas bien como usos o bien como características software.

Dromey instancia su modelo de calidad para la usabilidad. Las propiedades de la usabilidad que utiliza para su definición son:

- Facilidad en el aprendizaje: cuánto de fácil es el sistema para aprender a utilizarlo.
- Transparencia: una vez se ha aprendido su funcionalidad, cuánto de sencillo es el recordarla para futuros usos.
- Operabilidad: cuánto de eficiente y de fácil es el utilizar las funciones del sistema.
- Capacidad de ser receptivo: cualidad del sistema de cumplir las funciones en el tiempo debido.
- Capacidad de personalizar: indica si el sistema puede adaptar las interfaces a las necesidades del usuario.
- Soporte de lenguas extranjeras: capacidad para cambiar el idioma del sistema.
- Comandos susceptibles al contexto: muestra los comandos apropiados dependiendo del contexto.
- Operaciones directas: proporciona comandos para una ejecución directa.
- Teclas de acceso rápido: para aquellas funciones usadas más frecuentemente.
- Consistencia: indica el grado en el que los comandos son consistentes con el entorno donde se ejecuta la aplicación.

Dromey, tomando estas propiedades, las clasifica en tres grupos:

- Usos: facilidad en el aprendizaje, operabilidad, capacidad de personalizar.
- Comportamientos: capacidad de ser receptivo.
- Características software: transparencia, comandos susceptibles al contexto, operaciones directas, teclas de acceso rápido, consistencia.

De estos tres grupos, los comportamientos y los usos son los candidatos para estar en el nivel más alto en la descripción de la usabilidad. Puede que haya comportamientos que actúen como subordinados de otros comportamientos. Para jerarquizar las propiedades de la usabilidad, hay que preguntar a cada una de las propiedades si determina un comportamiento o un uso. Por ejemplo, para la Capacidad de personalizar se pregunta si contribuye a la Facilidad de aprendizaje, a operabilidad, y así sucesivamente. Una posible jerarquía sería la que se muestra en la Figura 2.

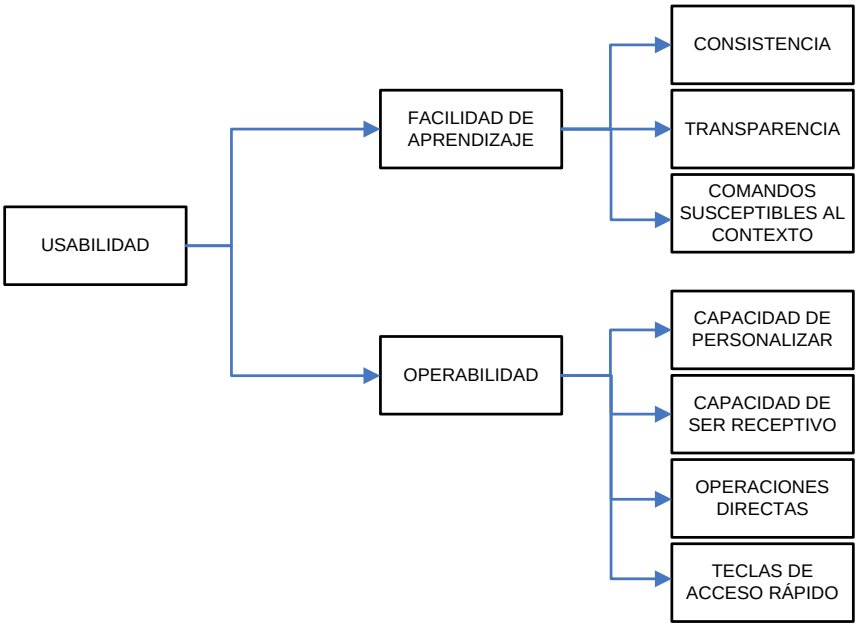

Figura 2. Jerarquía de propiedades de la usabilidad propuesta por Dromey

Como inconveniente al modelo propuesto por Dromey cabe destacar la subjetividad existente a la hora de construir la jerarquía de las propiedades de la usabilidad. Es el analista el que debe formularse a sí mismo las preguntas para ver las dependencias entre las propiedades de la usabilidad. Por lo tanto depende del analista que la jerarquía resultante sea de un tipo o de otro.

Además de los trabajos de Dromey y siguiendo la línea de Bastien y Scapin, existen muchos otros trabajos más recientes que se han centrado en definir aspectos que debería cumplir todo sistema que se considere usable. Siguiendo esta línea se encuentra el trabajo de Nielsen [27]. Nielsen define la usabilidad como parte de la capacidad de un sistema para ser aceptado por el usuario. Según Nielsen, la usabilidad no es una propiedad software unidimensional. La usabilidad tiene múltiples componentes y está asociada a los siguientes atributos:

- Facilidad de aprendizaje: el sistema debe ser fácil de aprender para el usuario de forma que éste puede utilizarlo rápidamente.
- Eficiencia: una vez el usuario haya aprendido a utilizar el sistema, éste debe proporcionar un alto nivel de productividad al usuario.
- Fácil de memorizar: el sistema debe ser fácil de recordar por el usuario de forma que si el usuario está un tiempo sin utilizarlo, sepa usarlo cuando lo necesita sin necesidad de pasar otra vez por la etapa de aprendizaje.

- Errores: el sistema debe tener un bajo número de errores y en caso de que aparezcan, el sistema debe poder recuperarse de ellos fácilmente.
- Satisfacción: los usuarios deben quedar satisfechos al usar el programa. Este es un aspecto totalmente subjetivo.

Algunos de estos componentes son también definidos como propiedades de la usabilidad en el trabajo de Dromey. Sin embargo, Nielsen no trata las propiedades que Dromey llama Operabilidad, consistencia y transparencia. El resto de propiedades de Dromey se encuentran dentro de los componentes de Nielsen. Además de las definiciones de las componentes de la usabilidad, Nielsen aporta un conjunto de 10 heurísticos que debería seguir todo sistema que se considere usable:

1. Diálogo simple y natural: los diálogos no deben contener información que sea irrelevante o innecesaria. Además, la información debe aparecer en un orden lógico y natural.
2. Hablar con el lenguaje del usuario: el lenguaje utilizado se debe expresar en forma clara con oraciones que le sean familiares al usuario.
3. Minimizar al usuario la necesidad de memorizar: el usuario no debe memorizar información de una parte del sistema para utilizarla en otra distinta. La información necesaria en cada interfaz debería ser visible en todo momento, incluyendo la ayuda.
4. Consistencia: los usuarios no se deberían preguntar si diferentes palabras, situaciones o acciones representan la misma cosa.
5. Retroalimentación: el sistema debe mantener informados a los usuarios en todo momento sobre las operaciones que se están realizando.
6. Salidas claramente marcadas: los usuarios a menudo se equivocan de interfaz y necesitan un botón de salida para volver rápidamente a la interfaz anterior.
7. Teclas de acceso rápido: proporcionan un mecanismo de acceso rápido para los usuarios expertos, a la vez que garantizan que los no expertos puedan ejecutar también las mismas acciones.
8. Buenos mensajes de error: los mensajes de error se deben expresar con un lenguaje claro y entendible por el usuario. Además, estos mensajes de error deben describir el problema y proponer posibles soluciones.
9. Prevención de errores: el sistema debe prevenir aquellos errores más comunes que puede ocasionar el usuario
10. Ayuda y documentación: la ayuda debe ser fácil de buscar y estar centrada en las tareas del usuario, con un listado de los pasos que debe seguir el usuario para realizar la tarea y no ser demasiado larga.

El uso de estos heurísticos tiene los mismos inconvenientes que los criterios ergonómicos de Bastien y Scapin. Nielsen no propone un mecanismo concreto para aplicar dichos heurísticos a los sistemas. Además, la evaluación de los heurísticos está muy basada en los test de usuario, y por tanto la experimentación está centrada en el usuario final. Este tipo de evaluación proporciona una información directa del usuario, pero es costosa en cuanto a tiempo y recursos, ya que el sistema debe estar implementado y requiere la participación directa del usuario en el proceso.

También existen trabajos centrados solo en algunas áreas de la usabilidad, como es el caso del trabajo de Shneiderman [40], el cual se centra exclusivamente en las medidas de la usabilidad relacionadas con la *eficiencia* y la *satisfacción*. Para medir ambos aspectos plantea las siguientes medidas de usabilidad:

- Tiempo de aprendizaje: indica cuánto tiempo lleva a los miembros normales de la comunidad de usuarios aprender cómo usar las acciones relevantes para un conjunto de tareas. Nielsen también define esta misma componente.
- Velocidad de realización de tareas: representa cuánto tiempo se tarda en realizar las tareas de medición.

- Porcentaje de errores de los usuarios: indica cuántos y qué clases de errores comete la gente al realizar las tareas de evaluación. Aunque el tiempo en cometer y solucionar errores podría estar incluido en la velocidad de realización de tareas, el manejo de errores es un componente tan crítico del uso de la interfaz que merece un estudio exhaustivo. Es equivalente a la componente de Nielsen llamada *Errores*.
- Retención con el paso del tiempo: indica si los usuarios mantienen bien sus conocimientos después de una hora, un día o una semana. La capacidad de retención parece estar muy ligada al tiempo de aprendizaje; la frecuencia de uso también juega un papel muy importante. Es equivalente a la componente de Nielsen llamada *Fácil de memorizar*.
- Satisfacción subjetiva: muestra si a los usuarios les gustó usar diferentes aspectos de la interfaz. Esta medida se puede determinar mediante una entrevista o con encuestas escritas que incluyen escalas de satisfacción y espacio para comentarios libres. Es equivalente a la componente definida por Nielsen con el nombre de *Satisfacción*.

Estas medidas son insuficientes ya que dejan muchos aspectos de usabilidad sin medir, como puede ser por ejemplo la operabilidad o la consistencia, ambas propuestas por Dromey. Tal y como se verá en las próximas secciones del documento, el Modelo de Usabilidad propuesto en este trabajo abarca muchas más medidas de usabilidad que las propuestas por Shneiderman.

Shneiderman plantea medir estas medidas de usabilidad mediante pruebas de usabilidad con el usuario. El proceso a seguir está mucho más detallado que el propuesto por Nielsen. En el caso de Shneiderman, plantea la necesidad de disponer de un laboratorio de usabilidad dentro del cual se realicen todas las mediciones de la usabilidad. Este laboratorio compromete con la usabilidad a empleados, clientes y usuarios de una organización. Las pruebas de usabilidad que se pueden realizar en este laboratorio son las siguientes:

1. Modelos en papel: se dibuja en un papel el contenido de las interfaces para valorar las reacciones del usuario ante la redacción, composición y secuencia de pantallas.
2. Pruebas de usabilidad ligeras: es una aproximación del tipo “rápido y sin cuidado” en el análisis de tareas, desarrollo de prototipos y pruebas.
3. Pruebas de usabilidad competitivas: comparan una interfaz con las versiones previas o con productos similares de los competidores.
4. Pruebas de usabilidad universal: esta aproximación prueba interfaces con usuarios, hardware, plataformas software y redes muy diversas.
5. Pruebas de campo y laboratorios portátiles: este método de prueba pone a trabajar las nuevas interfaces en entornos realistas durante un periodo de tiempo de evaluación fijo.
6. Pruebas de usabilidad remotas: dado que las aplicaciones Web están disponibles mundialmente, es atractivo dirigir las pruebas de usabilidad en línea, sin incurrir en la complejidad y el costo de llevar a los participantes al laboratorio.
7. Pruebas ¿puedes romperlo?: en esta aproximación los usuarios tratan de encontrar errores fatales en el sistema o, en algunas circunstancias, acabar con él.

El uso de un laboratorio de usabilidad detecta rápidamente los errores en la usabilidad, pero su mantenimiento es muy costoso. Además del uso del laboratorio, Shneiderman propone también los mecanismos tradicionales de:

- Realización de encuestas: son encuestas escritas que el usuario debe rellenar una vez ha experimentado con la aplicación.
- Pruebas de aceptación: cuando se escribe el documento de requisitos o cuando se oferta el contrato se deben establecer criterios explícitos de aceptación. Estos criterios son los que se evalúan mediante esta prueba.
- Evaluación durante el uso en producción: consiste en realizar la evaluación de la aplicación dentro del entorno de producción. En esta evaluación deben participar los

administradores del sistema, personal al servicio del usuario y el equipo de mantenimiento.

De forma similar a los 10 heurísticos de Nielsen, Shneiderman ha definido las ocho reglas de oro para conseguir que un sistema sea usable. Tal y como se puede apreciar, son muy similares a los heurísticos de Nielsen:

1. Esforzarse para conseguir consistencia
2. Habilitar para los usuarios más frecuentes las teclas de acceso rápido.
3. Ofrecer información de retroalimentación al usuario.
4. Diseñar diálogos que no sena ambiguos.
5. Ofrecer prevención de errores y manejo sencillo de los errores.
6. Permitir deshacer las acciones fácilmente.
7. Soporte para un lugar interno de control.
8. Reducir la carga de memoria a corto plazo.

Otra propuesta similar a la de Shneiderman y la de Nielsen, es la elaborada por Dix et al [8]. En su trabajo especifica una serie de principios que pueden ser aplicados al diseño de sistemas interactivos para mejorar su usabilidad. En definitiva lo que pretende es elaborar un catálogo de principios de usabilidad, tal y como se propone el presente documento. Estos principios se agrupan en tres categorías: *facilidad de aprendizaje, flexibilidad* y *robustez.*

- Facilidad de aprendizaje: tiene el mismo significado que le dio Nielsen en su trabajo, es decir, la característica de un sistema que permite a los usuarios noveles entender cómo funciona la aplicación y cómo alcanzar el máximo nivel de rendimiento. El trabajo de Dix et al se describe con mayor detalles los componentes de esta categoría:
    - Capacidad de predecir: es la capacidad del sistema de informar al usuario el efecto futuro que causará cada una de las acciones que realice. Esta información está basada en un historial de interacciones pasadas.
    - Capacidad de sintetizar: evalúa el efecto de acciones pasadas en el actual estado del sistema.
    - Familiaridad: la capacidad del usuario de asociar el nuevo sistema con un conocimiento previo adquirido en el mundo real o en otros sistemas.
    - Capacidad de generalizar: es la capacidad del sistema para que el usuario pueda extender el conocimiento de una interacción específica a partir de otras aplicaciones o situaciones similares.
    - Consistencia: semejanza en el mecanismo de entrada/salida surgido de situaciones similares o tareas objetivo similares.
- Flexibilidad: se refiere a las múltiples formas de intercambio de información entre el sistema y el usuario. Esta categoría está compuesta por los siguientes componentes:
    - Iniciativa de diálogo: permite al usuario libertad respecto a las restricciones del diálogo de entrada de datos del sistema.
    - Multi-tarea: capacidad del sistema de soportar la interacción con más de una tarea en el mismo tiempo.
    - Capacidad de migrar tareas: es la habilidad de pasar el control de una tarea al usuario, al sistema o a ambos.
    - Capacidad de sustitución: permite que valores equivalentes de entrada y salida sean sustituidos unos por otros.
    - Capacidad de personalizar: capacidad del modificar la interfaz de usuario por parte del usuario o del sistema.
- Robustez: un usuario utiliza un ordenador para llevar a cabo una serie de objetivos en su trabajo o en su dominio de tareas. La robustez en esa interacción cubre las

características que soportan el éxito de los objetivos. Esta categoría esta compuesta por las siguientes componentes:

- Capacidad de observación: habilidad del usuario para evaluar el estado interno del sistema desde su representación perceptible.
- Capacidad de recuperación: habilidad del usuario de realizar una acción correcta después de que haya surgido un error en el sistema.
- Capacidad de ser receptivo: cómo el usuario percibe el ratio de comunicación con el sistema.
- Conformidad con las tareas: es el grado en el que los servicios del sistema soportan todas las tareas que el usuario desearía realizar y en la forma en las que el usuario las entienda.

Dix et al aunque detallan con más precisión que Nielsen algunos de los componentes de la usabilidad, no tratan todos los aspectos que trata Nielsen. Dejan de lado componentes meramente subjetivos, como es el caso del componente que Nielsen llama *satisfacción*. Además, la noción de *eficiencia* o *ratio de error* definidas por Nielsen no aparecen en la categorización de Dix et al. En cuanto a la evaluación de la usabilidad, Dix et al plantean más técnicas que los test de usabilidad propuestos por Nielsen. Las técnicas propuestas por Dix et al son estas tres:

1. Estudio en un laboratorio: esta técnica plantea la evaluación de la usabilidad dentro de un laboratorio.
2. Estudio de campo: esta técnica lleva al diseñador o al evaluador dentro del entorno de trabajo del usuario para observar el sistema en acción y evaluarlo en ese mismo entorno.
3. Diseño participativo: es una técnica que abarca todo el proceso de desarrollo, no solo la evaluación de la usabilidad. Esta técnica plantea el diseño del sistema en el lugar de trabajo donde se utilizará, incorporando al usuario no solo como un evaluador, sino como un miembro más del equipo de desarrollo.

El proceso de evaluación de Dix et al es similar al planteado por Shneiderman, pero el trabajo de este último detalla las distintas alternativas, además de detectar muchas otras no planteadas por Dix et al. Por lo tanto en cuanto a técnicas de evaluación, son mejor las planteadas por Shneiderman.

Otros autores han dejado de lado el cómo hacer la evaluación de la usabilidad y se han centrado en definir de forma inequívoca los atributos de la usabilidad. Aunque estos atributos ya fueron definidos en la ISO 9126-1 [19], los atributos propuestos por Fitzpatrick [13] son más consistentes con las necesidades del mundo real y los negocios. Fitzpatrick para identificar los atributos de un producto software usable utiliza la metáfora de tres ramas: la rama de la *calidad software*, la rama de las *obligaciones legales* y la rama de la *interacción persona ordenador*.

- Calidad del software: la calidad del software se define como el grado en el que el software posee una combinación deseada de atributos. Estos atributos se refieren a factores de calidad o características de calidad. Estos factores de calidad incluyen las características: *corrección del software, fiabilidad, eficiencia, integridad, usabilidad, mantenibilidad, posibilidad de probarlo, flexibilidad, portabilidad, reusabilidad* e *interoperablidad*.
- Obligaciones legales: son las obligaciones referidas a las regulaciones relacionadas con la salud y temas de seguridad, pero sobre todo aquellas relacionadas con la seguridad mínima necesaria y requisitos de salud sobre el trabajo delante de un ordenador. Fitzpatrick obtiene esta obligación a partir de la directiva europea relacionada con la regulación del muestreo de información por pantalla.
- Interacción persona ordenador: se describe como el estudio de las personas, tecnología de computadores y la forma en que unos influyen en otros. Esta rama incluye los siguientes factores de calidad: *lo apropiado que es el sistema*, *usabilidad,*

seguridad, adaptabilidad y *facilidad en el aprendizaje*. Dentro de esta rama se encuentran tres dimensiones diferenciadas:

- La dimensión humana: los aspectos que forman esta dimensión son el comportamiento humano, la memoria humana, la habilidad de aprender, la adquisición del conocimiento humano, aspectos cognitivos, percepción humana del trabajo del sistema y cómo este trabajo se puede conceptualizar. Otros temas que también se deben considerar en esta dimensión son el perfil del usuario (incluyendo las habilidades del usuario y las habilidades psíquicas), el conocimiento previo del usuario o si es experto en el dominio, la educación general y la predisposición del usuario hacia las nuevas tecnologías.
- Las capacidades de los ordenadores: los dispositivos utilizados reflejan el estilo de diálogo preferido. Un estilo de diálogo es uno de los métodos por los cuales los usuarios pueden interactuar con el sistema. El alcanzar los objetivos de la comunidad HCI es propiciado por el correcto uso de los dispositivos de entrada/salida junto con las tareas a completar y con las habilidades del usuario.
- La interacción del usuario con el sistema: en esta área hay dos tópicos de interés: (1) *Guías y principios para el diseño de los diálogos*: las guías y principios pueden ser utilizados tanto para la evaluación como para especificar los requisitos de diseño de la interfaz; (2) *Equipo y entorno*: la productividad del usuario y la satisfacción están soportadas por un equipo preparado para realizar las tareas y el entorno adecuado para trabajar.

De las tres ramas planteadas por Fitzpatrick, el presente trabajo solo va a estar centrado en la última, la llamada *interacción persona ordenador*. Aunque en el trabajo de Fitzpatrick los atributos de la usabilidad están mejor definidos que en la ISO 9126-1, aun se les podría describir con mayor nivel detalle, ya que están definidos a un nivel muy abstracto. Además, en los factores de calidad definidos por Fitzpatrick entre las tres ramas, se echa en falta factores totalmente subjetivos por parte del usuario, como por ejemplo *el grado de atracción*.

Tal y como se ha mostrado, existen un gran número de mecanismos para la evaluación de la usabilidad. Es complejo el deducir cómo estos métodos se relacionan entre sí y porqué una técnica puede ser más adecuada que otras en un contexto determinado. Según Van Welie et al [45], el concepto de usabilidad se debe despedazar de forma que permita la comparación desde los puntos de vista teóricos y prácticos.

Para abordar estos problemas, Van Welie et al proponen un modelo de usabilidad basado en tres capas:

1. Capa de usabilidad: esta capa está dividida en tres aspectos: *eficiencia, efectividad* y *satisfacción*. Este nivel es demasiado abstracto para estudiar la usabilidad y no es aplicable en la práctica. Sin embargo, proporciona tres pilares básicos para encontrar la usabilidad de los sistemas.
2. Indicadores de uso: son los indicadores del nivel de usabilidad que pueden ser observados realmente en la práctica cuando el usuario trabaja. Cada uno de estos indicadores contribuye a los aspectos abstractos de la capa de usabilidad. Por ejemplo, un bajo ratio de errores contribuye a una mejor efectividad y buena velocidad de rendimiento indica buena eficiencia.
3. Significados: esta capa no se puede observar en los tests de usuario y no es un objeto de estudio por sí mismo, mientras que los indicadores sí que son aspectos observables. Los significados se utilizan en heurísticos para mejorar uno o más de los indicadores de uso y por lo tanto no son objetivos por ellos mismos. Por ejemplo, la consistencia puede tener un efecto positivo sobre la facilidad de aprendizaje y los avisos pueden reducir los errores. Cada significado puede tener un efecto positivo o negativo en algunos de los indicadores de uso. Dependiendo del heurístico que se utilice, la lista de los significados tendrá unos u otros elementos.

Van Welie et al además de desgranar la usabilidad en un modelo de tres capas, proponen mecanismos de evaluación de la usabilidad basados en dicho modelo. Cuando la evaluación se realice con usuarios, la evaluación se realizará sobre indicadores de uso, mientras que cuando la evaluación se haga durante la etapa de diseño del sistema, la evaluación se hará sobre los significados, haciendo una estimación del impacto que puede tener cada uno de ellos. Van Welie et al plantean dos posibles evaluaciones:

1. Evaluación con usuarios: son pruebas similares a las planteadas por Shneiderman. Es un buen método para obtener datos del uso actual del sistema. Para esta evaluación se deben utilizar escenarios u otras técnicas para obtener buenos indicadores de la usabilidad del sistema. Este método tiene como inconveniente que cuando el nivel de usabilidad no es satisfactorio, los indicadores de usabilidad no sirven para averiguar la razón de esos valores. Como solución a este problema habría que observar los significados en el diseño del sistema.
2. Evaluación durante el diseño: este método es más problemático que la evaluación con usuarios. Los indicadores de uso no se pueden evaluar directamente. En este caso se deben observar los significados que influencian en los indicadores de uso. Usando escenarios y ensayos, cada uno de los significados se puede evaluar observando la forma en la que se presentan en el diseño y estimando los impactos positivos y negativos en los indicadores de uso.

Cuando la evaluación refleja que la usabilidad debe ser mejorada, el siguiente problema es encontrar qué significados se deben cambiar y cómo deben ser modificados. En algunos casos puede ser obvio como mejorar la usabilidad pero en otros casos la solución no está tan clara, ya que algunos significados pueden tener un efecto negativo sobre los indicadores de uso, mientras que otros significados pueden causar el efecto contrario. En el caso que la solución no sea obvia, el diseñador debe volver atrás y estudiar el dominio del conocimiento.

Los métodos de evaluación propuestos por Van Welie et al tienen como inconveniente que son demasiado dependientes del dominio del conocimiento. Además, si los valores de usabilidad salen muy bajos con estas evaluaciones, es muy laborioso detectar cómo mejorar estos valores, por lo tanto este método tiene poca aplicación industrial. Los métodos propuestos por Shneiderman o Dix et al son mejor en este sentido y por lo tanto son más sencillos de aplicar en ambientes industriales.

Otros autores han centrado su evaluación de la usabilidad en las etapas tempranas del proceso de producción del software. Los trabajos de esta línea siguen el mismo ideal que el que se pretende conseguir mediante el Modelo de Usabilidad propuesto en el presente documento: la evaluación de la usabilidad en el Modelado Conceptual. Son trabajos que no se centran en evaluar la usabilidad del software mediante tests de usuarios utilizando la aplicación generada, sino que evalúan la usabilidad a partir de modelos. Uno de lo pioneros en iniciar esta tendencia fueron McCall et al. [25]. Según estos autores, la medida de calidad de un sistema debe ser medida durante las fases de requisitos y de diseño. Las medidas de calidad son meras predicciones y están orientadas hacia las fases de desarrollo que se realizan en etapas posteriores hasta la finalización del sistema.

McCall et al proponen una estructura para medir la calidad de los sistemas software basada en tres niveles distintos:

1. Factores: es el nivel superior dentro de la estructura propuesta. En la definición e identificación de estos factores se deben considerar tanto los usuarios como el uso que se le dará al sistema. El usuario necesita un conjunto predefinido de factores para identificar qué características de calidad desea en el sistema que se está desarrollando. Para facilitar este uso, la definición de los factores debe llevar a unas cuantificaciones entendibles por el usuario. La aproximación utilizada para satisfacer ambos requisitos fue evaluar cómo un usuario de un programa observa el producto final de un desarrollo software. Los puntos de vista observados relacionan las actividades del ciclo de vida del producto software. Los factores propuestos por McCall

et al. son: *mantenibilidad*, *flexibilidad*, *capacidad de testear*, *portabilidad*, *reusabilidad*, *interoperabilidad*, *corrección*, *fiabilidad*, *eficiencia*, *integridad* y *usabilidad*.

2. Criterios: para cada uno de los factores, se define un conjunto de criterios. Estos criterios representan atributos que pueden ser medibles durante el desarrollo del sistema.
3. Métricas: las métricas de calidad del software, una vez establecidas, proporcionan medidas para los atributos del software. Estas métricas no se aplican al código que implementa la funcionalidad del sistema, sino a la documentación que soporta su especificación. Las métricas se pueden definir en forma de checklist o fórmulas para contar atributos específicos, como el número de caminos alternativos en un módulo.

Siguiendo la tendencia iniciada por McCall han surgido trabajos más recientes, como los de Folmer y Bosch [14]. Según estos autores la usabilidad debería estar presente en todas las etapas de desarrollo del sistema. Es necesario relacionar la usabilidad con la arquitectura software para poder especificar el diseño de la usabilidad a nivel de arquitectura. Sin embrago, es difícil trazar una línea directa entre los atributos de usabilidad y la arquitectura software. Para que esta relación sea posible, es necesario descomponer los atributos de la usabilidad en elementos más detallados, como por ejemplo “el número de errores cometidos durante la realización de una tarea”, que es un indicador del atributo fiabilidad.

Folmer y Bosch dividen la evaluación de la usabilidad en dos niveles de abstracción: *el espacio del problema* y *el espacio de la solución*.

- Espacio del problema: que a su vez se divide en dos capas:
    - Definición de la usabilidad y sus atributos: en este nivel se define el concepto de la usabilidad y su descomposición en componentes medibles. Esta capa consiste en todas las definiciones o clasificaciones de la usabilidad según los atributos que definen el concepto de usabilidad.
    - Indicadores de usabilidad: esta capa consiste en medidas concretas relacionadas con atributos de usabilidad.
- Espacio de la solución: existen varias guías de usabilidad, herramientas y técnicas que permiten el diseño de aplicaciones usables.
    - Conocimiento del diseño: el espacio de la solución solo contiene esta capa. Esta capa expresa todo el conocimiento del diseño que existe en la comunidad de la ingeniería del software. Consiste en el diseño de heurísticos, estándares de diseño para interfaces, modelos de tareas, técnicas de prototipado y similares. Los patrones de usabilidad también se encuentran dentro del conocimiento del diseño. Los patrones de usabilidad son soluciones probadas de un problema dentro de un contexto determinado

Una vez definidos los dos espacios en los cuales ubicar la usabilidad, es necesario relacionarlos. Esta relación se consigue mediante las llamadas *propiedades de usabilidad*, que relaciona las capas de indicadores de usabilidad con la capa de conocimiento del diseño. Estas propiedades de usabilidad se derivan de heurísticos, principios de diseño y principios ergonómicos que sugieren diseños a un alto nivel. Estas propiedades de usabilidad están directamente relacionadas con decisiones de diseño software. Es necesario remarcar que las relaciones establecidas mediante estas propiedades no son siempre relaciones de uno a uno.

El trabajo de Folmer y Bosch refleja la misma idea que la propuesta que se plantea en el presente trabajo: incorporar la usabilidad desde las etapas tempranas del proceso de desarrollo del software. En el trabajo de Folmer y Bosch no se especifica cómo hacer las transformaciones entre las distintas capas que identifica. En caso de que las transformaciones sean manuales, el trabajo de incorporar la usabilidad en todo el proceso de desarrollo será un trabajo muy costoso. En nuestra propuesta, en cambio, estas transformaciones se harán de forma automática gracias a un Compilador de Modelos, por lo tanto la carga de trabajo para el analista, generada por la incorporación de la usabilidad en el proceso, es poco significativa.

Otro trabajo en la misma línea que el de MacCall et al y muy parecido al de Folmer y Bosch es el de Bass [3]. El objetivo de su trabajo es el de mejorar la usabilidad del sistema a través de decisiones tomadas en la arquitectura del mismo. Para ello es necesario entender la relación entre arquitectura software y usabilidad para asegurar que el sistema es usable.

Según Bass, los expertos en usabilidad determinan qué aspectos de usabilidad son apropiados para una tarea o una aplicación. Posteriormente, los ingenieros del software estudian estos aspectos dentro del contexto de la arquitectura y los implementan dentro de las restricciones de tiempo, coste, seguridad, viabilidad y otras similares.

En su trabajo, Bass presenta una serie de conexiones entre aspectos específicos de la usabilidad, como la posibilidad del usuario a deshacer una acción, y la arquitectura software. Los diseñadores pueden utilizar esta propuesta bien para generar soluciones a los problemas de usabilidad detectados o bien para evaluar los sistemas para aspectos específicos de usabilidad.

Además de identificar los conectores entre aspectos de usabilidad y la arquitectura, Bass identifica *escenarios generales* que definen aquellos aspectos sensibles a la usabilidad. Bass presenta estos escenarios apoyándose en una jerarquía de beneficios humanos, de forma que el equipo de diseño del sistema o los stakeholders pueden evaluar su aplicabilidad sobre el sistema en construcción. Además, Bass proporciona una jerarquía de mecanismos arquitecturales y para cada escenario proporciona un patrón arquitectural que implemente dicho escenario. Además, categoriza estos patrones según sus mecanismos de arquitectura apropiados. Otro aporte del trabajo de Bass es la ubicación de los escenarios dentro de una matriz de beneficios y mecanismos arquitecturales. Los patrones arquitecturales posibilitan que el experto en usabilidad pueda evaluar el impacto de las soluciones propuestas en otros atributos de calidad, como por ejemplo, el rendimiento. Este trabajo, al igual que el de Folmer y Bosch está muy relacionado con la idea principal del trabajo propuesto en el presente documento.

Hay otros trabajos aun más orientados a la evaluación temprana y no tanto a incorporar la usabilidad a lo largo de todo el proceso de desarrollo software. En esta línea está el trabajo de Keinonen [23]. Este autor construye un modelo de usabilidad relacionado con la evaluación del producto. La usabilidad es según Keinonen un aspecto multidimensional. Las dimensiones que componen la usabilidad son:

- Aproximación de diseño: la usabilidad puede ser vista como un conjunto de métodos o aproximaciones de diseño, como la ingeniería de la usabilidad o el diseño centrado en el usuario.
- Atributo del producto: esta dimensión está formada por ejemplos de productos, propiedades de los sistemas, o cualidades que afectan a la usabilidad. Existen muchas guías de usabilidad y guías de estilo que proporcionan instrucciones detalladas para el desarrollo de interfaces de usuario.
- Medición: esta dimensión está formada por las aproximaciones de usabilidad que proporcionan medidas a nivel operacional, sobre objetivos de la usabilidad y sobre las relaciones entre usabilidad, utilidad, aceptación del producto, e impresiones del usuario a la hora de interactuar con el sistema.

El Modelo de Usabilidad propuesto por Keinonen está basado en el concepto de *atributos*. Se definen los atributos como criterios de evaluación subjetivos que se aplican para evaluar la usabilidad de los productos o conocer las preferencias del usuario. Es decir, los atributos son las expectativas depositadas sobre la calidad de la interfaz, la interacción real con el sistema, y la respuesta emocional. Divide los atributos de usabilidad en los siguientes grupos:

- Atributos de interfaz de usuario: son aquellos aspectos de la interfaz perceptibles por los usuarios. En este grupo se encuentran los atributos:

    - Funcionalidad: se refiere a la opinión del usuario respecto a la idoneidad de las características del producto desde el punto de vista de la flexibilidad, versatilidad, exactitud y capacidad de las funciones del sistema.
    - Lógica: se refiere a la opinión del usuario sobre la calidad de la estructura de información interna. Incluye las dimensiones de consistencia, compatibilidad, longitud de secuencias, modalidad del dispositivo, etc.
    - Presentación: se refiere a la opinión del usuario sobre el nivel de calidad de la presentación de la interfaz. Este atributo está influenciado por la familiaridad, facilidad de comprensión, simplicidad, consistencia, intuitividad, etc.
    - Documentación: se refiere a la opinión del usuario sobre la calidad de la documentación presentada en el manual de usuario. Este nivel incluye los otros niveles ya presentados (Funcionalidad, Lógica y Presentación)
- Atributos de interacción: estos atributos reflejan la experiencia personal relacionada con el intercambio de información entre el usuario y el sistema. Está formado por los siguientes atributos:
    - Utilidad: es el grado con el que el usuario considera que utilizando el sistema puede alcanzar un buen rendimiento en su trabajo.
    - Facilidad de uso: indica el grado en que el usuario cree que utilizando el sistema no tendrá que realizar ningún esfuerzo mental.
- Atributos emocionales: son atributos que se refieren a sentimientos del usuario, como comodidad, felicidad, o lo opuesto. Todo ello como resultado de utilizar el sistema. Estos atributos son totalmente independientes de los aspectos funcionales del sistema y no está dividido en categorías.
- Atributos no relacionados con la usabilidad: representa aquellos atributos referentes a la opinión del usuario respecto a la ergonomía y apariencia. La ergonomía se refiere a las capacidades psíquicas y las características del sistema que no son interactivas. La apariencia se refiere al aspecto del sistema, en otro sentido al relacionado con el intercambio de información para fines prácticos. No se incluyen atributos relacionados con el país del usuario o precio del sistema.

Este modelo de usabilidad está organizado en diferentes jerarquías. Cada una de las jerarquías es más adecuada que el resto para un propósito concreto ya que cada una de ellas representa diferentes aspectos de la evaluación de la usabilidad.

Para la medición de la usabilidad mediante su modelo, Keinonen plantea un *test de preferencia*. En este test se plantean varias medidas en un total de seis monitores. El usuario estudia y evalúa cada uno de estos monitores. A partir de estas evaluaciones, se puede derivar indicadores que muestran la importancia de los atributos de usabilidad.

Una vez más, el objetivo del presente documento coincide con el objetivo de otro autor de renombre dentro del mundo científico. Keinonen propone una evaluación temprana al igual que en nuestro trabajo. Aun así existe una diferencia importante entre ambos trabajos. En nuestra propuesta, además de conseguir esta evaluación temprana, los cambios para mejorar los resultados de la usabilidad se hacen fácilmente y en muy poco tiempo gracias a primitivas conceptuales. Es decir, todos los cambios de mejora de la usabilidad se harán en el Modelado Conceptual de OO-Method. Sin embargo, en el trabajo de Keinonen los cambios que son necesarios para mejorar la usabilidad no están especificados y dependen del método de desarrollo empleado.

Además de los trabajos basados en la medición de la usabilidad por medio de Modelos Conceptuales, hay otro conjunto de trabajos dedicados a representar la usabilidad de los sistemas utilizando patrones. En esta línea están los trabajos de la herramienta PIM (Patterns In Modelling) [37]. Es una herramienta que ayuda al analista a aplicar patrones para modelos a partir de los cuales se puede obtener la interfaz del usuario. En analista utiliza estos patrones para combinarlos y crear interfaces de usuario. Estos patrones se aplican sobre un nivel de

modelos, no sobre el propio código que implementa la aplicación. Esta herramienta permite la trazabilidad entre los siguientes modelos:

- Modelo de tareas*:* es una descomposición jerárquica de tareas en subtareas hasta un nivel atómico. E orden de la ejecución se define entre relaciones temporales establecidas entre pares de tareas
- Modelo de diálogo*:* describe la interacción entre el usuario y la máquina. Especifica cuándo el usuario puede invocar funciones, cuando puede seleccionar o especificar entradas al sistema y cuando el sistema puede consultar información.
- Modelo de presentación*:* describe los elementos de la interfaz de usuario que aparecen en diferentes estados del modelo de diálogo. Consiste en una descomposición jerárquica de posibles pantallas del sistema en grupos de objetos de interacción. De esta forma se puede construir una relación entre los modelos de diálogo y presentación, simplemente uniendo elementos de interacción abstractos del Modelo de presentación con los estados de diálogo en los que aparecerán.
- Modelo de organización: asigna atributos de estilo como tamaño, fuente o color a los elementos abstractos del modelo de presentación.

Además, esta herramienta define cuatro pasos en el uso de los patrones:

1. Identificación: se identifica un subconjunto M' del modelo objetivo M. Este subconjunto será reemplazado y extendido por la instancia del patrón seleccionada en el próximo paso
2. Selección: se selecciona el patrón apropiado para aplicarlo a M'
3. Instanciación: se define la estructura concreta del patrón y se asignan valores a partes variables del patrón para adaptarlo a la situación actual
4. Integración: el patrón instanciado se integra en el modelo completo. Se conecta a las otras partes del modelo para crear una pieza de diseño perfecta.

Aunque actualmente ya existen herramientas como PIM, capaces de modelar aspectos de la usabilidad a través de patrones y derivar a partir de estos modelos la interfaz del sistema, no existen herramientas que generen el código de la aplicación por completo, tal y como se pretende con nuestra propuesta. Es decir, que no solo generen la interfaz, sino también toda su funcionalidad. OO-Method posee un Compilador de Modelos que genera aplicaciones totalmente ejecutables, incluyendo todos los aspectos funcionales, de persistencia y de interfaz. La idea que se pretende conseguir al integrar el Modelo de Usabilidad en el Modelado Conceptual de OO-Method es derivar el código que represente la usabilidad junto con el código que ya se obtiene actualmente por medio del Compilador de Modelos. De esta forma se obtienen aplicaciones usables totalmente ejecutables sin apenas esfuerzo por parte del analista.

Finalmente, otro de los trabajos que sigue la filosofía del trabajo de McCall al igual que Keinonen es el de Abrahao et al [1][11][2]. En este trabajo la autora plantéa un método de evaluación de la usabilidad basado en la ISO 9126-1 para aplicarlo sobre Modelos Conceptuales. En dicho método se hace una distinción entre *subcaracterísticas* y *atributos,* tal y como viene diferenciado en la ISO 9126-1. Las subcaracterísticas representan un aspecto general relacionado con la usabilidad, mientras que los atributos representan un aspecto de la usabilidad medible e indivisible.

El método de evaluación planteado por Abrahao es lo suficientemente genérico como para instanciarlo en cualquier entorno de desarrollo. Además, persigue el mismo objetivo que el que se pretende con el presente trabajo, la medición de la usabilidad a partir de Modelos Conceptuales. Gracias a estas dos cualidades de dicho método de evaluación, nos ha parecido el más adecuado para aplicarlo a los Modelos Conceptuales de OO-Method. Más adelante se presenta el método de evaluación con más detalle.

Sin embargo, dicho método de evaluación es tan general que no se puede aplicar directamente sobre cualquier entorno de producción de código. Es por ello que en el presente documento se presenta una serie de pasos necesarios para llevar a la práctica el método de evaluación de la usabilidad dentro de un entorno de generación automática de código a partir de modelos. En las siguientes secciones se detallan estos pasos necesarios.

El Modelo de Usabilidad sobre el que se apoya el método de evaluación propuesto por Abrahao, se construyó además de con la ISO 9126-1, tomando componentes de usabilidad definidas por todos los autores mencionados en este capítulo. Sin embargo, dicho Modelo de Usabilidad tiene mayor nivel de detalle que los modelos de usabilidad comentados. El Modelo de Usabilidad se estructura jerárquicamente (como los criterios ergonómicos de Bastien y Scapin o las propiedades de usabilidad de Dromey) y los nodos hoja de esta estructura son componentes de usabilidad que se pueden medir de forma inequívoca. Esta característica hace que la descripción de estos componentes tenga que ser muy detallada para poder definir la métrica, y por tanto las componentes de usabilidad tienen que ser descompuestas a descripciones muy elementales. Esta descomposición, además de posibilitar la definición de métricas, favorece la comprensión de las componentes de usabilidad por parte den analista. En este caso el Modelo de Usabilidad difiere de la propuesta de Fitzpatrick, cuyo nivel de detalle en la descripción de los atributos de usabilidad es más pobre que la utilizada en el Modelo de Usabilidad de Abrahao.

En cuanto a la evaluación de la usabilidad mediante el Modelo de Usabilidad de Abrahao, el proceso de evaluación es mucho más sencillo que los descritos por Shneiderman o Dix et al. En las propuestas de estos autores, las evaluaciones se hacían con la aplicación ya implementada y en la mayoría de los casos es necesaria la colaboración del usuario y del resto de stakeholders. En cambio, utilizando el método de evaluación propuesto por Abrahao la evaluación se hace sobre el Modelado Conceptual y sin la ayuda del usuario. Esto implica que no es necesario implementar la aplicación ni definir una serie de tareas específicas para hacer las pruebas con usuarios sobre ella, ahorrando tiempo y recursos.

Todas las ventajas que presenta el método de evaluación de Abrahao con respecto al resto de métodos ha hecho que se haya elegido como método para la evaluación que se plantéa en este documento. En el mecanismo de evaluación de la usabilidad propuesto en este documento, la evaluación se hace automáticamente a través de fórmulas matemáticas que calculan el grado de usabilidad de cada una de las componentes del modelo. Estas fórmulas están definidas siguiendo las guías de usabilidad, los heurísticos como los de Nielsen y Shneiderman y los criterios ergonómicos de Bastien y Scapin. Esta evaluación automática hace posible que antes de que se genere el código que implementa la aplicación y sin la ayuda del usuario, el analista tiene una aproximación de lo usable que será la aplicación de forma rápida, sencilla y sin costes, ya que tampoco es necesaria la presencia de expertos en usabilidad durante el proceso de evaluación. Así, en caso de que los valores de usabilidad no sean buenos, el analista puede modificar los modelos para mejorar estos valores.

A continuación se presenta una tabla resumen con todos los autores citados en este capítulo. En ella se muestra para cada autor el Modelo de Usabilidad que utiliza y cómo ha estructurado sus componentes. Además, para aquellos autores que tengan especificado algún método de medición sobre su Modelo de Usabilidad, también se nombre el método.

Autor	Modelo de Usabilidad	Método de Medición
Bastien y Scapin	Criterios ergonómicos	No hay ninguno especificado
Dromey	El Modelo de Usabilidad está dentro de un Modelo de Calidad jerarquizado	No hay ninguno especificado
Nielsen	Componentes que forman un Modelo de Usabilidad	Heurísticos y tests de usuarios

Shneiderman	Centrado en las medidas relacionadas con *eficiencia* y *satisfacción*	- Pruebas basadas en un laboratorio de usabilidad - Ocho reglas de oro en la construcción de un sistema - Encuestas - Pruebas de aceptación - Evaluación en producción
Dix et al	Principios de usabilidad agrupados en *facilidad de aprendizaje, flexibilidad* y *robustez*.	- Estudio en un laboratorio - Estudio de campo - Diseño participtaivo
Fitzpatrick	Agrupa los atributos de usabilidad en *calidad software, obligaciones legales* e *interacción persona ordenador*	No hay ninguno especificado
Van Welie et al	Modelo basado en tres capas: *usabilidad, indicadores de uso* y *significados*	- Evaluación con usuarios - Evaluación durante el diseño
McCall	Estructura basada en tres niveles: *factores, criterios* y métricas	No hay ninguno especificado
Folmer y Bosch	Modelo de usabilidad con dos niveles de abstracción: *espacio del problema* y *espacio de la solución*	No hay ninguno especificado
Bass	Aplicado a patrones arquitecturales para utilizarlos en escenarios concretos	Matriz de beneficios basada en el uso de los patrones
Keinonen	Modelo jerárquico basado en *atributos de interfaz, atributos de interacción, atributos emocionales* y *atributos no relacionados con la usabilidad*	Test de preferencia
Abrahao	Modelo de Usabilidad basado en la ISO 9126-1 para utilizarlo sobre Modelos Conceptuales	Evaluación utilizando los atributos del Modelo de Usabilidad
Radeke et al	Herramienta PIM basada en patrones de usabilidad aplicados a modelos	No hay ninguno especificado

Tabla 1. Resumen de otros Modelos de Usabilidad

Tal y como se puede apreciar en la Tabla 1, hay bastantes Modelos de Usabilidad que no tienen especificado ningún método de evaluación, dejando al analista la decisión del método a utilizar. En cuanto a los modelos de usabilidad, aunque cada divide los atributos de usabilidad en capas con nombres muy diversos, todos ellos coinciden en la mayoría de los atributos que componen el modelo.

3. MÉTODOLOGÍAS DE TRANSFORMACIÓN DE MODELOS

En la historia del software se ha evolucionado ganando en abstracción con el paso del tiempo. Se empezó desarrollando a muy bajo nivel, con un lenguaje muy próximo al código máquina llamado ensamblador. Posteriormente se evolucionó a lenguajes donde se usaba un lenguaje muy próximo al programador, como son los lenguajes de tercera generación. Estos lenguajes se fueron acompañando de componentes gráficos que facilitaban la creación de las interfaces del sistema. Las tendencias actuales pretenden llegar a un mayor nivel de abstracción, donde con solo la construcción de modelos, se pueda generar código que soporte el sistema software a construir.

En el desarrollo de sistemas basados en modelos, dichos modelos pueden verse de dos maneras diferentes: a) como la documentación del sistema; b) como el principal artefacto de desarrollo del sistema. La última propuesta es lo que la industria llama *Model Driven Development* (MDD), donde los sistemas son desarrollados en base a esos modelos. Lo que ocurre durante la transición desde el modelo a la implementación resultante se llama transformación.

Esta aproximación fue usada intuitivamente por muchos desarrolladores, hasta que el *Object Management Group* (OMG) en 2001 dio un nombre específico: *Model Driven Arquitecture* (MDA) [31]. Para ello estandarizó una forma de desarrollar MDD definiendo tres puntos de vista: 1) un punto de vista independiente de la computación; 2) un punto de vista independiente de la plataforma; 3) un punto de vista específico para plataforma.

- El punto de vista independiente de la computación está centrado en el entorno y los requisitos del sistema, y define esta representación con el Modelo Independiente de Computación (*Computation Independent Model,* CIM), donde no hay relación entre el sistema y la implementación.
- El punto de vista independiente de plataforma se centra en la operación del sistema que permanece constante en cualquier plataforma. Se representa en lo que se llama Modelo Independiente de Plataforma (*Platform Independent Model*, PIM).
- Por último, el punto de vista específico para plataforma suministra aquellas características específicas de una plataforma, y son obtenidas definiendo el Modelo de Plataforma Específico (*Platform Specific Model,* PSM). La Figura 3 muestra una aproximación a MDA

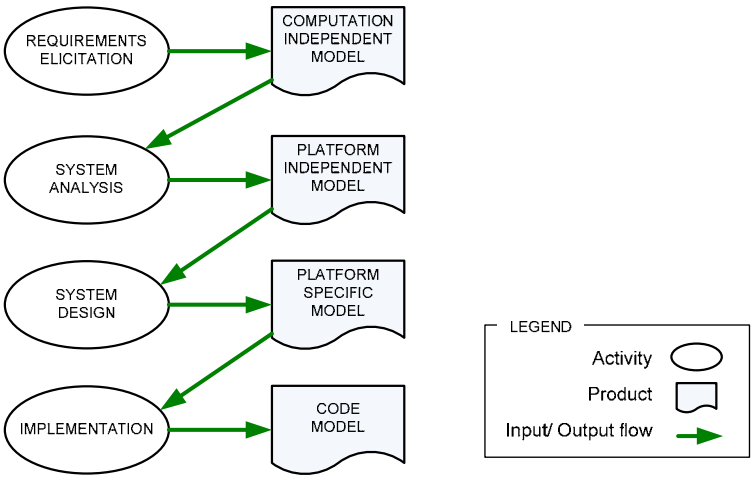

Figura 3. Aproximación MDA

Como se define en la OMG, una transformación de modelos es el proceso de conversión de un modelo a otro que representa el mismo sistema [31]. Normalmente el modelo objetivo es un nivel de abstracción más bajo que los modelos originales, y además, está más cerca de la implementación final. Usando transformaciones consecutivas, se llega a un modelo ejecutable del sistema, es decir, al código fuente del sistema.

Las transformaciones se pueden aplicar manualmente, con la asistencia de un ordenador o automáticamente. Aunque el estándar no es restrictivo, propone automatizar las transformaciones siempre que sea posible.

Sin tener en cuenta el grado de automatización, las transformaciones tienen que ser especificadas inequívocamente usando algún lenguaje. De nuevo hay una gran variedad, desde descripciones en lenguaje natural a especificación QVT [36]. El estándar MDA usa el término *mapping* para estas especificaciones, pero se pueden llamar *reglas de transformación*.

Entre varias aproximaciones de transformación de modelos que se pueden tomar, la Transformación del Metamodelo merece nuestra especial atención. Las reglas de transformación se definen en términos del metamodelo origen y destino, como se muestra en la siguiente figura.

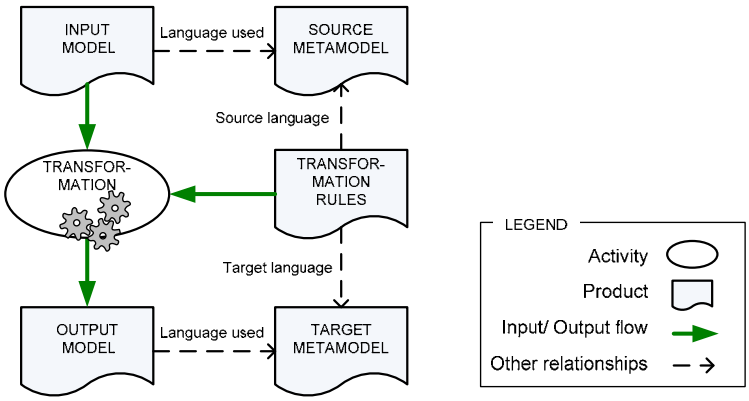

Figura 4. Transformación del Metamodelo

3.1. OO-Method

El Modelo de Usabilidad propuesto, aunque es utilizable dentro de cualquier proceso de desarrollo de software, se diseñó para ser utilizado en una metodología de transformación de modelos como es OO-Method [32]. OO-Method es un Compilador de Modelos capaz de generar código desde un conjunto de modelos conceptuales de forma automática. Esta técnica se basa en la separación del espacio del problema (Modelado Conceptual) y del espacio de la solución (el código que implementa el sistema). Esto sitúa a OO-Method como una metodología para implementar herramientas que sigan las directrices de MDA [31] ya que separa la especificación conceptual de las aplicaciones de sus posibles implementaciones software. OO-Method está formado básicamente por dos modelos: el Modelo Conceptual y el Modelo de Ejecución.
El *Modelo Conceptual* está dividido en cuatro vistas complementarias:

- El *Modelo de Objetos* permite especificar la estructura de las clases identificadas en el dominio del problema, así como las relaciones estructurales y las relaciones de agente sobre los servicios de las clases.
- El *Modelo Dinámico* y el *Modelo Funcional* se centran en el comportamiento del sistema. En el Modelo Dinámico se representan las posibles secuencias de eventos que pueden ocurrir en la vida de los objetos. El Modelo Funcional se utiliza para especificar el efecto que tienen los eventos sobre el estado de los objetos.

- Por último, el *Modelo de Presentación* [26] ofrece una descripción abstracta de la interfaz del sistema. Este modelo está dividido en tres niveles, donde en cada nivel hay definidos unos patrones concretos para crear este modelo (Ver Figura 5):
    1. **Nivel 1**. Árbol de jerarquía de acciones (AJA): siguiendo el Principio de Aproximación Gradual expresa como la funcionalidad será presentada al usuario que acceda al sistema.
    2. **Nivel 2**. Unidades de interacción (UIs): modela las unidades de interacción que el usuario deberá emplear para llevar a cabo sus tareas. Existen cuatro tipos: (1) UI de servicio: modela la presentación de un diálogo cuyo objetivo es que un usuario lance un servicio; (2) UI de instancia: modela la presentación de los datos o estado de una instancia. Se define sobre una clase y se le proporciona un conjunto de visualización (información a ser mostrada), acciones (lanzadas sobre el objeto) y navegación (alcanzabilidad entre instancias); (3) UI de población: modela unidades de interacción cuyo objetivo es mostrar un conjunto de instancias para una clase dada. Se pueden establecer mecanismos de filtrado y ordenación para facilitar la selección y consulta de objetos; (4) UI de maestro / detalle: modela unidades de interacción más complejas de presentación cabecera-despliegue que se construyen a partir de unidades de interacción más sencillas.
    3. **Nivel 3.** Patrones elementales (PEs): Permiten restringir y precisar el comportamiento de las diferentes unidades de interacción. Los patrones elementales son los siguientes: Introducción, Selección definida, Información complementaria, Dependencia, Recuperación de estado, Agrupación de argumentos, Filtro, Criterio de ordenación, Conjunto de visualización, Acciones y Navegación.

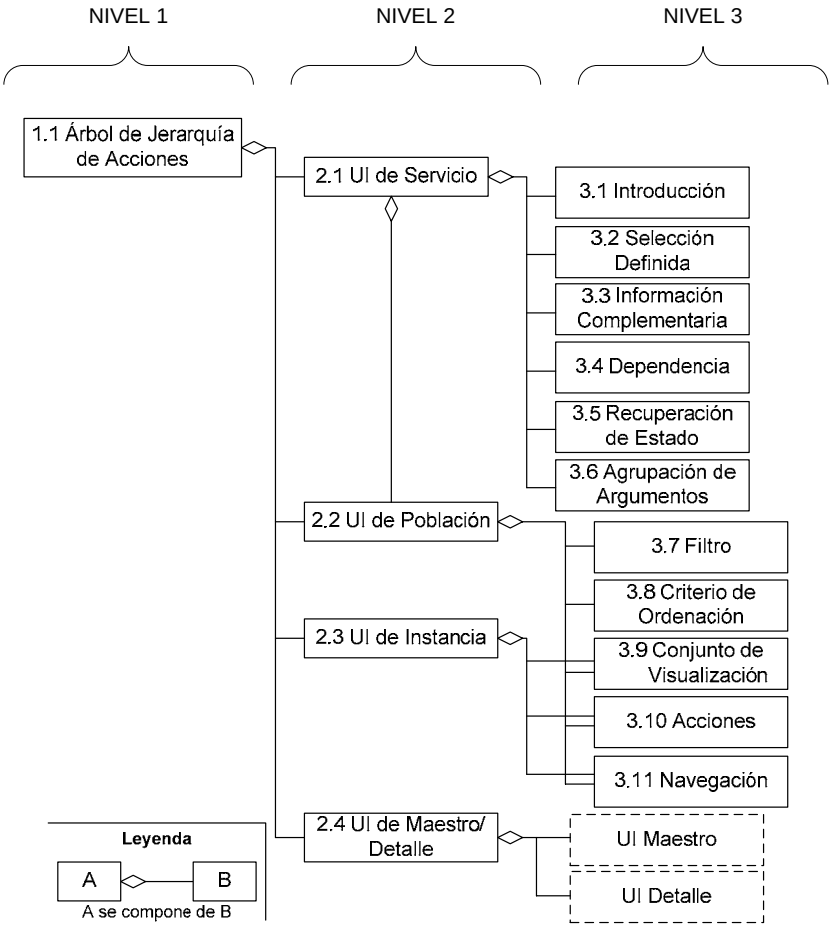

Figura 5. Modelo de Presentación de OO-Method

Por otro lado, el *Modelo de Ejecución* establece las reglas de transformación de un Modelo Conceptual a su representación software correspondiente en una plataforma tecnológica concreta.

OO-Method tiene su herramienta industrial llamada OlivaNOVA [5]. En la actualidad se está trabajando en mejorar el Modelo de Presentación de OlivaNOVA para enriquecerlo con mayor expresividad. Para ello se ha propuesto trasformar el Modelo de Presentación de OlivaNOVA en un Modelo de Interacción [33], que está compuesto por dos vistas que representan la interacción con el usuario a distintos niveles de abstracción:

- **Modelo de Interacción Abstracto:** define la interfaz sin tener en cuenta aspectos concretos de visualización. Consiste en un modelo que representa la interfaz de manera que sea tan independiente como sea posible de los tipos de interacción y las peculiaridades de las plataformas. Por ejemplo, una lista de opciones se puede implementar usando un dropdownlistbox, un checkbox o un radiobutton. A este nivel de abstracción solo podemos definir que el elemento es una lista, no el componente en que se transformará. Este Modelo de Interacción Abstracto estaría formado por la mayoría de los patrones de presentación del actual Modelo de Presentación de OO-Method.
- **Modelo de Interacción Concreto:** define detalles de la interfaz. Es un modelo de interfaces de usuario que especifica su representación con elementos que pueden ser percibidos por el usuario final de una manera que es tan independiente como sea posible de la plataforma. En este modelo se decide por ejemplo, si una lista de opciones será un dropdownlistbox, un checkbox o un radiobutton. Cada uno de los cambios que se pueden aplicar a los componentes de una interfaz se define por medio de *parámetros*. Estos parámetros definen qué valores se pueden definir y modificar para cada uno de los componentes de la interfaz.

La Figura 6 muestra el proceso de generación de código de un Compilador de Modelos instanciado para el caso de OlivaNOVA y su equivalencia con las capas definidas en el estándar MDA. En el Modelado Conceptual vemos los 4 modelos utilizados para representar el sistema: *Modelo de Objetos*, *Dinámico*, *Funcional* y de *Interacción*. El Modelo de Interacción se ha dividido en dos modelos distintos: el *Modelo de Interacción Abstracto* y el *Modelo de Interacción Concreto*. La capa formada por todos estos modelos se llama *Platform Independent Model* en MDA. El Compilador de Modelos, que equivale a la capa *Platform Specific Model* de MDA, es el encargado de aplicar las transformaciones de modelo a código. Por último, aplicando estas transformaciones, se crea un código dividido en tres capas: de *interfaz*, de *lógica,* y de *persistencia* que equivale a la capa MDA llamada *Code Model.*

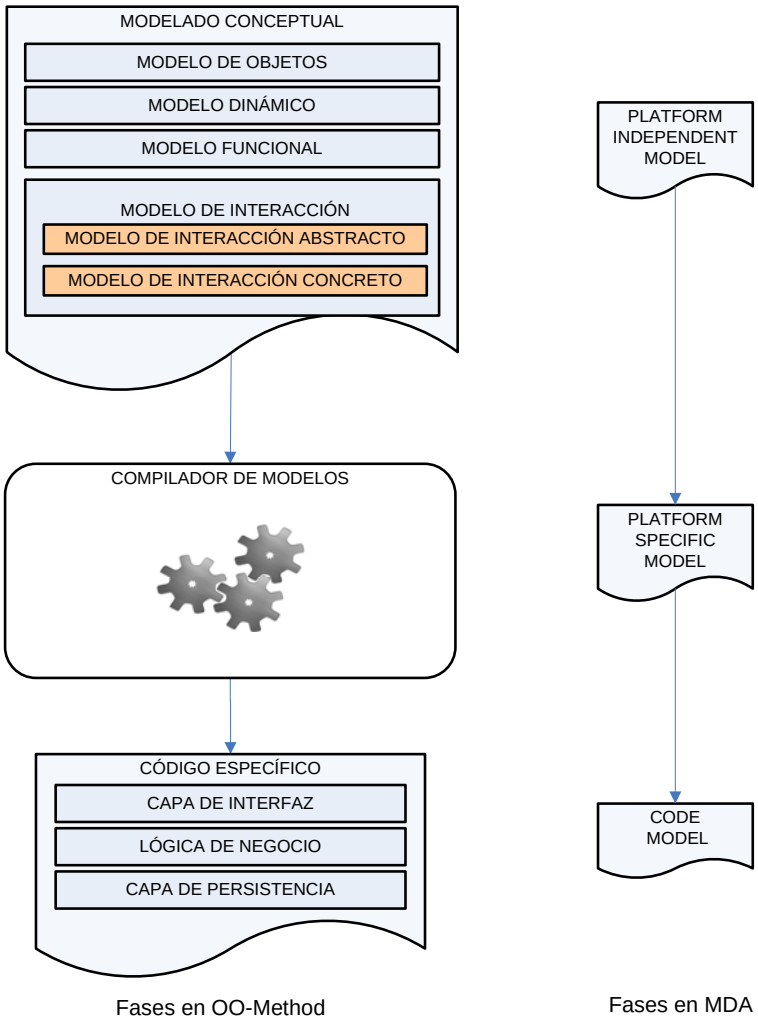

Figura 6. Proceso de generación automática de código para OlivaNOVA

El código generado con OlivaNOVA tiene una arquitectura Cliente / Servidor, tal como muestra la Figura 7. Al generar el código con el Compilador de Modelos, el sistema se divide en dos aplicaciones, el cliente y el servidor. La parte servidora es la parte que contiene la capa de la lógica del negocio y la de persistencia. La capa de interfaz está en el cliente. Este tipo de arquitectura tiene la ventaja de que un mismo servidor puede dar servicio a varios clientes. La parte servidora se instala en una máquina, a la cual pueden hacer máquinas que tengan instalado el cliente. Este acceso se hace a través de Internet.

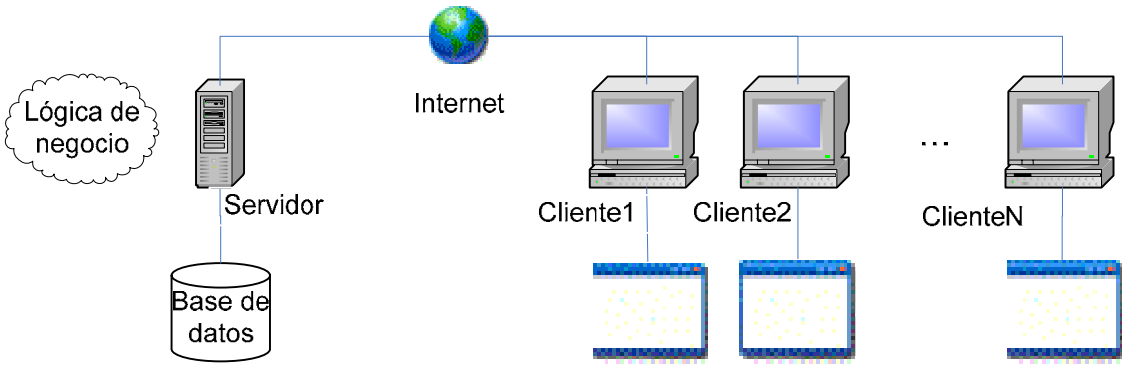

Figura 7. Arquitectura Cliente / Servidor de OlivaNOVA

3.2. Generación de código en OlivaNOVA

El Compilador de Modelos genera un Código Específico para resolver un problema concreto en un sistema. Sin embargo, tal y como se ha explicado anteriormente, este código se puede abstraer a un Código Genérico válido para cualquier solución que se implemente. Este código no implementa ninguna lógica de negocio, simplemente muestra las clases, las relaciones entre sí y una breve descripción de la funcionalidad de la clase, sin entrar en la lógica de sus métodos. En este apartado se presenta a modo de ejemplo, las clases en Código Genérico que se generan a partir de algunos del patrón del Modelo de Interacción de OlivaNOVA llamado Unidad de Interacción de Servicio.

La Figura 8 muestra cómo el Código Genérico implementa el patrón de servicio del Modelo de Interacción de OlivaNOVA. La clase *Service wrapper* llama a las clases del servidor que contienen el código de los servicios que forman el sistema. Esta clase se encarga de hacer de conector entre las clases del cliente y las del servidor. Los servicios que puede ejecutar el usuario son mostrados usando la clase *Form*. Existe un *Form* por cada servicio. Estas ventanas de servicio tienen dos botones, *OK* y *Cancel*, para ejecutar el servicio o cerrar la ventana respectivamente.

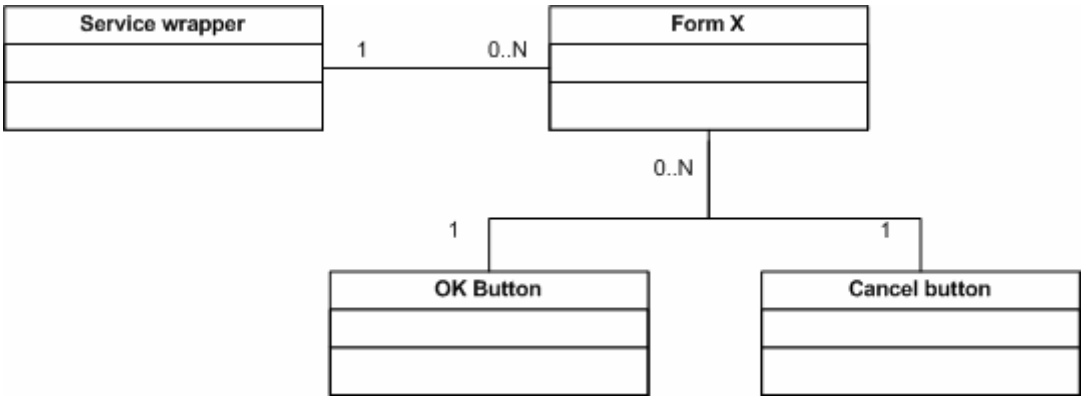

Figura 8. Unidad de Interacción de Servicio en Código Genérico

4. INSTANCIACIÓN DE UN MÉTODO PARA LA EVALUACIÓN DE LA USABILIDAD

4.1. Modelo de Usabilidad

El método de evaluación utilizado en este trabajo está basado en un Modelo de Usabilidad. Los modelos de usabilidad propuestos en la ISO/IEC 9126-1 [19] y la ISO/IEC 9441-11 [18] son modelos generales que describen una serie de métricas para medir la usabilidad de todo tipo de sistemas software. Sin embargo, estas métricas son demasiado genéricas y no están lo suficientemente detalladas como para definir fórmulas con las cuales medir la usabilidad. Además no tienen en cuenta aspectos de autores que han definido otros modelos de usabilidad. Por este motivo se propuso, en trabajos de Abrahao et al [2][11] [1], un Modelo de Usabilidad que partiendo de ambas ISO, detalla los aspectos de usabilidad con mayor nivel de detalle con la finalidad de definir métricas precisas. Además, también incorpora aspectos definidos por otros autores. Este nuevo Modelo de Usabilidad se definió basándose en dos primitivas:

- Subcaracterísticas: son conjuntos que agrupan aspectos de la usabilidad con relación entre sí estructurados jerárquicamente.
- Atributos: se definen dentro de las subcaracterísticas dependiendo del aspecto de la usabilidad que representen. Cada uno de los atributos tienen una métrica definida mediante la cual se puede medir el grado de usabilidad del mismo.

Para obtener este Modelo de Usabilidad, se analizaron varias taxonomías de criterios de usabilidad propuestas en la literatura [14][17][39], entre las que destacan la de Bastien y Scapin [4] y sus criterios ergonómicos validados empíricamente.

1. **Facilidad de aprendizaje**	3.1.2. Multiplicidad de Instalación
1.1. Facilidades de Ayuda	3.1.3. Capacidad de Actualización
1.1.1. Completitud de Documentación	3.1.4. Transparencia de Actualización
1.1.2. Documentación Multiusuario	3.2. Validación de datos
1.1.2.1. Distinción por Capacitación	3.3. Capacidad de Control
1.1.2.2. Distinción por Rol	3.3.1. Aplazamiento de Edición
1.2. Predecibilidad	3.3.2. Soporte de Cancelación
1.2.1. Grado de Significación de Iconos	3.3.3. Ejecución Explícita
1.2.2. Título de Iconos o Enlaces	3.3.4. Soporte de Interrupción
1.2.3. Determinación de la Acción	3.3.5. Soporte de Deshacimiento
1.3. Realimentación Informativa	3.3.6. Soporte de Rehacimiento
1.4. Facilidad de Memorización	3.4. Capacidad de Adaptación
2. **Comprensibilidad**	3.4.1. Adaptabilidad
2.1. Legibilidad	3.4.2. Adaptatividad
2.1.1. Tamaño de Fuente	3.5. Consistencia
2.1.2. Contraste	3.5.1. Comportamiento Constante de Controles
2.1.3. Disposición	3.5.2. Permanencia de Controles
2.2. Facilidad de Lectura	3.5.3. Estabilidad de los Controles
2.2.1. Agrupación Cohesiva de la Información	3.5.4. Consistencia de Orden
2.2.1.1. Agrupación por Disposición	3.5.5. Consistencia de Etiquetado
2.2.1.2. Agrupación por Formato	3.6. Gestión de Errores
2.2.2. Densidad de la Información	3.6.1. Prevención de Errores

2.3. Familiaridad de conceptos 2.3.1. Significación del Etiquetado 2.3.2. Internacionalización 2.4. Reducción de la Carga de Trabajo 2.4.1. Brevedad 2.4.1.1. Inicialización de Valores 2.4.1.1.1. Completitud de Valores Iniciales 2.4.1.1.2. Modificabilidad de Valores Iniciales 2.4.1.2. Minimización de acciones 2.4.2. Autodescripción 2.5. Orientación al usuario 2.5.1. Calidad de los Mensajes 2.5.2. Navegabilidad 3. **Operabilidad** 3.1. Capacidades de Instalación 3.1.1. Facilidad de Instalación	3.6.2. Recuperación de Errores 3.7. Monitorización del Estado del Sistema 4. **Grado de atracción** 4.1. Metáfora 4.2. Uniformidad del Color del Fondo 4.3. Uniformidad del Color de la Fuente 4.4. Uniformidad del Estilo de la Fuente 4.5. Uniformidad del Tamaño de la Fuente 4.6. Uniformidad en Disposición de Elementos 4.7. Atracción Subjetiva 5. **Conformidad** 5.1. Cumplimiento con ISO/IEC 9126 (partes 1 y 3) 5.2. Cumplimiento con ISO 9241-10 5.3. Cumplimiento con Guía de Estilo de Microsoft 5.4. Cumplimiento con Guía de Estilo de Java

Tabla 2. Modelo de usabilidad propuesto

Tal y como se muestra en la Tabla 2, las subcaracterísticas entre las que están agrupadas los atributos son cinco:

1. **Facilidad de Aprendizaje:** es la capacidad del sistema para que el usuario aprenda a utilizarlo. Hace referencia a los atributos de un producto software que facilitan que el usuario aprenda su manejo. Cuanto mayor sea el valor de esta subcaracterística, más sencillo será el aprendizaje del usuario. Los atributos de esta subcaracterística se agrupan en 4 grandes grupos:
   - Facilidades de ayuda: son las utilidades que proporciona el sistema para ayudar al usuario. Por ejemplo, asistentes, o documentación.
   - Predecibilidad: indica la facilidad con que un usuario puede determinar el resultado de sus acciones futuras.
   - Realimentación informativa: el sistema debe responder al usuario cada vez que éste solicite alguna acción. Es una forma de informar al usuario de que el sistema ha reaccionado con su petición.
   - Facilidad de memorización: es la capacidad del usuario por retener la forma en la que funciona el sistema. Está fuertemente ligado con los atributos del grupo “Facilidades de ayuda”, ya que cuantas más facilidades de ayuda tenga el sistema, mejor será su memorización.
2. **Comprensibilidad:** es la capacidad del producto software de permitir al usuario entender si el software le resulta adecuado y cómo puede usarlo para realizar tareas determinadas en unas condiciones de uso concretas. Los atributos se agrupan en 5 grandes grupos.
   - Legibilidad: la cualidad que permite distinguir un carácter alfanumérico de los demás e identificarlo.
   - Facilidad de lectura: la facilidad con que un usuario puede percibir, buscar y entender la información mostrada en pantalla.
   - Familiaridad de conceptos: el grado en el que los conceptos mostrados en el sistema son conocidos por el usuario.
   - Reducción de la carga de trabajo: es la capacidad del sistema para reducir las acciones que el usuario debe realizar en el sistema.

- Orientación al usuario: es la capacidad del sistema para indicar al usuario dónde se encuentra dentro de la aplicación y las acciones que puede hacer en cada momento.

3. **Operabilidad:** se define como la capacidad del producto software de permitir al usuario manejarlo y controlarlo. Se refiere a los atributos que facilitan el control del usuario y la operación. Los atributos de esta subcaracterística se agrupan en siete conjuntos:
   - Capacidad de instalación: mide la forma en la que el sistema se instala para poder ejecutarlo.
   - Validación de datos: es la capacidad del sistema para asegurar que los datos introducidos por el usuario son válidos. Un ejemplo serían las máscaras de entrada.
   - Capacidad del control: mide el grado de control que el usuario tiene sobre la ejecución del sistema.
   - Capacidad de adaptación: mide el grado en que el sistema se puede adaptar a las necesidades del usuario.
   - Consistencia: los atributos de este grupo miden la estabilidad y coherencia de los controles del sistema.
   - Gestión de errores: mide la forma en la que el sistema controla los errores que suceden en la ejecución de una acción. Este grupo incluye atributos tanto para medir el comportamiento del sistema ante un error, como la forma en la que se muestra el mensaje de error al usuario.
   - Monitorización del estado del sistema: es la capacidad de informar al usuario de los eventos que suceden en el sistema y que son independientes de las acciones que ejecuta el usuario. Por ejemplo, el mensaje que se muestra cuando la batería del portátil está baja o el icono de que ha llegado un mensaje nuevo a nuestra cuenta de correo. Mide la capacidad de crear mensajes de aviso que muestran el estado del sistema sin prestar atención a las acciones que el usuario ordene ejecutar.
4. **Grado de atracción:** es la cualidad de medir cuanto de atractivo es el sistema para el usuario. Esta subcaracterística no agrupa sus atributos en grupo, ya que no dispone de gran cantidad de ellos. Los atributos que incluye son siete:
   - Metáfora: representa gráficamente el dominio del sistema. El sistema toma una apariencia visual concreta para que el usuario vea la aplicación como el dominio que implementa.
   - Uniformidad del color de fondo: indica el grado de homogeneidad en el color de fondo utilizado en todas las pantallas del sistema.
   - Uniformidad del estilo de la fuente: indica el grado de homogeneidad en el estilo de la fuente utilizado en todas las pantallas del sistema.
   - Uniformidad del tamaño de la fuente: indica el grado de homogeneidad en el tamaño de la fuente utilizado en todas las pantallas del sistema.
   - Uniformidad en la disposición de elementos: indica el grado de homogeneidad en la disposición de los elementos en cada una de las pantallas.
   - Atracción subjetiva: muestra el grado de atracción que cada usuario tiene con respecto al sistema. Es una medida totalmente subjetiva. Por ejemplo, hay usuarios que se pueden sentir más atraídos por un sistema que utilice el color azul que por un sistema que utilice el color rojo.
5. **Conformidad:** es la capacidad del producto software para observar los estándares, convenciones, guías de estilo o reglas relacionadas con la usabilidad. Cada uno de los atributos de esta subcaracterística mide el grado de cumplimiento de una guía o estándar específico.

Tal y como se muestra en la Tabla 2, los atributos medibles son de naturaleza muy diversa. El objetivo de este trabajo es el de determinar métricas para estos atributos de forma que sean

medibles en las primeras etapas de desarrollo. Sin embrago, no todos estos atributos son medibles en una etapa temprana. De la Tabla 2 se pueden extraer tres tipos de atributos, dependiendo de la fase del desarrollo del sistema software en la cual se puedan medir:

- Atributos medibles en el **Modelo Conceptual**: en este grupo se pueden ubicar aquellos atributos a los cuales se les puede definir una fórmula matemática basada en primitivas de Modelado Conceptual. Este tipo de atributos muestra un valor de usabilidad totalmente objetivo y preciso, según las fórmulas matemáticas. La precisión de estos atributos depende en buena medida de lo preciso que se haya definido la fórmula matemática que les proporciona un valor. El valor de usabilidad obtenido mediante estos atributos depende de la forma en que el analista haya construido el Modelo Conceptual.
- Atributos medibles en el **Compilador de Modelos**: son todos aquellos atributos que siempre tienen el mismo valor, ya que el Compilador de Modelos genera los aspectos que miden siempre de la misma forma (independientemente de cómo se haya creado el Modelado Conceptual). Por lo tanto, son atributos medibles y se les puede definir una fórmula matemática basada en la estrategia de generación de código de OO-Method. Sin embargo, el valor obtenido mediante estas fórmulas siempre sería el mismo y por lo tanto, carece de relevancia su medición. Para aquellos métodos de producción de software que se implementen a mano y no dispongan de un Compilador de Modelos, los atributos de este tipo son aquellos que dependen exclusivamente de las decisiones que tome el programador.
- Atributos medibles en la **Aplicación Final**: son aquellos atributos subjetivos que sólo pueden ser medidos por el usuario. Son atributos que dependen de los gustos o de la percepción del mundo que tenga el usuario. Normalmente, para medir este tipo de atributos se realizan tests de usuario [20] tomando varias personas para la experimentación. Los usuarios realizan todos ellos las mismas tareas sobre un sistema determinado y posteriormente rellenan un cuestionario mediante el cual se extraen los valores de usabilidad para cada uno de los atributos.

El Modelo de Usabilidad puede ser utilizado por cualquier método de producción de software. Sin embargo, la distinción entre estos tres tipos de atributos es totalmente dependiente del método de producción de software escogido. Dependiendo de la expresividad de sus modelos, cada atributo puede ser medido en el Modelo Conceptual, en el Compilador de Modelos o en la Aplicación Final.

A continuación se presenta la distinción de atributos para el caso particular de OO-Method. El criterio de distinción se ha basado en el Modelado Conceptual de la herramienta OlivaNOVA tal y como se encuentra actualmente implementada, es decir, teniendo en cuenta el Modelo de Objetos, Dinámico, Funcional y el de Presentación. La Tabla 3 muestra el Modelo de Usabilidad propuesto haciendo la distinción entre los tres tipos de atributos existentes. Los atributos medibles en el Modelo Conceptual llevan el distintivo MC, los medibles en el Compilador de Modelos el distintivo C y los medibles en la Aplicación Final el distintivo AF.

1. **Facilidad de aprendizaje**	3.1.2. Multiplicidad de Instalación**(C)**
1.1. Facilidades de Ayuda	3.1.3. Capacidad de Actualización**(C)**
1.1.1.Completitud de Documentación**(MC)**	3.1.4. Transparencia de Actualización**(C)**
1.1.2. Documentación Multiusuario	3.2. Validación de datos**(C)**
1.1.2.1. Distinción por Capacitación**(C)**	3.3. Capacidad de Control**(C)**
1.1.2.2. Distinción por Rol**(C)**	3.3.1. Aplazamiento de Edición**(C)**
1.2. Predecibilidad	3.3.2. Soporte de Cancelación**(C)**
1.2.1. Grado de Significación de Iconos**(C)**	3.3.3. Ejecución Explícita**(C)**
1.2.2. Título de Iconos o Enlaces**(C)**	3.3.4. Soporte de Interrupción**(C)**
	3.3.5. Soporte de Deshacimiento**(C)**
	3.3.6. Soporte de Rehacimiento **(C)**

1.2.3. Determinación de la Acción**(MC)** 1.3. Realimentación Informativa**(C)** 1.4. Facilidad de Memorización**(AF)** 2. **Comprensibilidad** 2.1. Legibilidad 2.1.1. Tamaño de Fuente**(C)** 2.1.2. Contraste**(C)** 2.1.3. Disposición **(C)** 2.2. Facilidad de Lectura 2.2.1. Agrupación Cohesiva de la Información 2.2.1.1. Agrupación por Disposición**(MC)** 2.2.1.2. Agrupación por Formato**(C)** 2.2.2. Densidad de la Información**(MC)** 2.3. Familiaridad de conceptos 2.3.1. Significación del Etiquetado**(MC)** 2.3.2. Internacionalización **(C)** 2.4. Reducción de la Carga de Trabajo 2.4.1. Brevedad 2.4.1.1. Inicialización de Valores 2.4.1.1.1. Completitud de Valores Iniciales**(MC)** 2.4.1.1.2. Modificabilidad de Valores Iniciales **(C)** 2.4.1.2. Minimización de acciones**(C)** 2.4.2. Autodescripción**(AF)** 2.5. Orientación al usuario 2.5.1. Calidad de los Mensajes**(MC)** 2.5.2. Navegabilidad**(MC)** 3. **Operabilidad** 3.1. Capacidades de Instalación 3.1.1. Facilidad de Instalación**(C)**	3.4. Capacidad de Adaptación**(C)** 3.4.1. Adaptabilidad**(C)** 3.4.2. Adaptatividad **(C)** 3.5. Consistencia 3.5.1. Comportamiento Constante de Controles**(C)** 3.5.2. Permanencia de Controles**(C)** 3.5.3. Estabilidad de los Controles**(C)** 3.5.4. Consistencia de Orden**(MC)** 3.5.5. Consistencia de Etiquetado**(MC)** 3.6. Gestión de Errores 3.6.1. Prevención de Errores**(MC)** 3.6.2. Recuperación de Errores**(C)** 3.7. Monitorización del Estado del Sistema **(C)** 4. **Grado de atracción** 4.1. Metáfora**(AF)** 4.2. Uniformidad del Color del Fondo**(C)** 4.3. Uniformidad del Color de la Fuente**(C)** 4.4. Uniformidad del Estilo de la Fuente**(C)** 4.5. Uniformidad del Tamaño de la Fuente**(C)** 4.6. Uniformidad en Disposición de Elementos**(C)** 4.7. Atracción Subjetiva**(AF)** 5. **Conformidad** 5.1. Cumplimiento con ISO/IEC 9126 (partes 1 y 3) **(C)** 5.2. Cumplimiento con ISO 9241-10**(C)** 5.3. Cumplimiento con Guía de Estilo de Microsoft**(C)** 5.4. Cumplimiento con Guía de Estilo de Java**(C)**

Tabla 3. Modelo de usabilidad propuesto distinguiendo los tres tipos de atributos

De los tres grupos anteriormente descritos, este trabajo se va a centrar en los atributos medibles en el Modelado Conceptual. La medición de atributos de este grupo tiene como ventajas principales las siguientes:

- No es necesario generar la aplicación para medir la usabilidad del sistema. La medición se puede automatizar en la fase de Modelado Conceptual, ganando el analista en tiempo y eficiencia.
- Es un tipo de medición de la usabilidad que no requiere la participación de usuarios, por lo tanto se ahorra el esfuerzo de preparar los tests de usuario y el tiempo que se debe dedicar a que los usuarios realicen las tareas con el sistema y rellenen las encuestas.
- Los resultados obtenidos mediante esta evaluación producen una retroalimentación inmediata al analista, el cual puede mejorar el Modelado Conceptual del sistema sin necesidad de retocar el código de la aplicación final.

Sin embargo, es importante resaltar que también hay un inconveniente importante con la evaluación temprana: la no participación del usuario en el proceso de evaluación de la usabilidad. Por lo tanto los atributos que solo pueden ser medibles en la Aplicación Final no se incluyen en el análisis de la usabilidad. Luego, solo estamos midiendo una parte de toda la usabilidad del sistema.

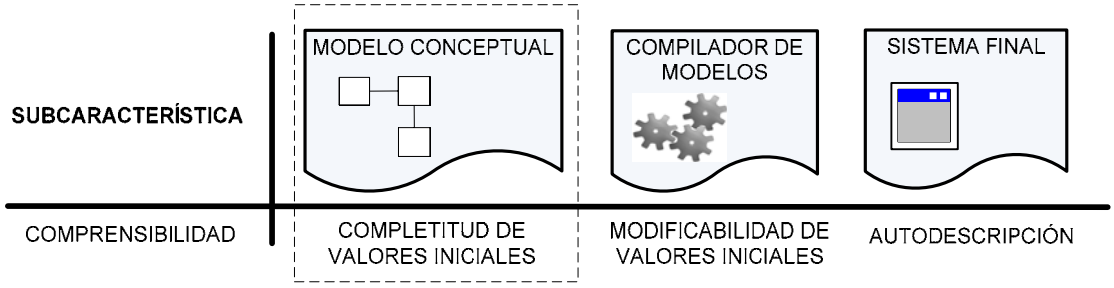

Figura 9. Descomposición de subcaracterísticas en atributos

La Figura 9 muestra este inconveniente gráficamente. La subcaracterística *Comprensibilidad* se mide a través de varios atributos, entre los que se encuentran *Completitud de valores iniciales, Modificabilidad de valores iniciales* y *Autodescripción*. Estos atributos se miden respectivamente en el *Modelo Conceptual*, en el *Compilador de Modelos* y en el *Sistema Final*. De estos tres atributos, con la evaluación temprana solo se puede medir el primero de ellos. El atributo *Modificabilidad de valores iniciales* no aporta un valor relevante para la medida de la usabilidad ya que siempre tendrá el mismo valor. En cambio el atributo *Autodescripción* tendrá un valor variable dependiendo del usuario que evalúe el sistema mediante tests de usuario en la aplicación final. Por lo tanto el valor de usabilidad que se obtendrá con la evaluación temprana de la usabilidad para la subcaracterística *Comprensibilidad* no será el valor real, sino solo una parte, el obtenido del Modelado Conceptual.

Esta evaluación temprana, aunque no obtiene el valor final de la usabilidad del sistema, es una buena estimación de ella, y por lo tanto se puede utilizar para predecir la usabilidad del sistema resultante. En base a esta evaluación temprana, se pueden hacer los cambios correspondientes en el Modelado Conceptual para mejorar la usabilidad que tendrá la aplicación software final.

4.2. Atributos que miden la usabilidad en una etapa temprana

De todos los atributos que forman el Modelo de Usabilidad representado en la Tabla 2, esta sección se va a centrar en aquellos atributos que se pueden medir en el Modelado Conceptual. Para cada uno de estos atributos, se ha definido una fórmula matemática que obtendrá un valor numérico que representará el valor de su usabilidad.

Sin embargo, el hecho de obtener un valor numérico para cada uno de los atributos no indica cuánto de usable es el atributo. Cada una de las fórmulas usadas como métricas de usabilidad obtendrá valores numéricos que oscilarán entre distintos rangos dependiendo del atributo al que pertenezcan. Por lo tanto, a cada atributo se le debería añadir un mecanismo para interpretar cada uno de los valores obtenidos mediante las métricas.

Como mecanismo para dar un significado al valor numérico obtenido mediante las métricas, se ha optado por definir indicadores para cada uno de los atributos. Estos indicadores asignan un valor cualitativo al valor numérico obtenido mediante las fórmulas matemáticas. Para ello, en cada atributo se definen cinco rangos entre los cuales puede oscilar el valor numérico obtenido mediante la fórmula. A cada uno de estos rangos, se les asigna el valor cualitativo que toma el atributo. Los cinco valores cualitativos que puede tomar un atributo son: MB (Muy Bueno), B

(Bueno), R (Regular), M (Malo), (MM) Muy Malo. De esta forma se le asigna una semántica a cada una de las medidas obtenidas mediante las fórmulas matemáticas.

A continuación, para cada uno de los atributos de usabilidad medibles en la fase de modelado se presentan las fórmulas que devuelven un valor para medir la usabilidad. Además, también se presentan los indicadores definidos en cada uno de los atributos para dotarlos de significado.

4.2.1. Métricas

En esta sección se muestran las métricas para cada uno de los atributos medibles en las fases tempranas del desarrollo software. Dichas métricas se presentan agrupadas por la subcaracterística a la cual pertenecen cada uno de los atributos. El valor de usabilidad de cada subcaracterística viene determinado por el valor de usabilidad de cada uno de los atributos que contiene.

Todas las métricas aquí presentadas se basan en primitivas del Modelado Conceptual de OO-Method, pero pueden ser extrapoladas a cualquier método de generación automática de código basado en modelos conceptuales. Es decir, tomando las propuestas de este trabajo como base, las métricas se pueden adaptar a cualquier proceso de generación de código que utilice un Modelado Conceptual como base de desarrollo.

Es importante remarcar que hay varios atributos de usabilidad que se podrían medir en el Modelado Conceptual, pero que actualmente no se pueden medir ya que se basan en primitivas que no están actualmente tratadas por OO-Method. Por ejemplo, el atributo “Grado de significación de los iconos”, ya que OO-Method no contempla la posibilidad de incluir iconos mediante sus modelos. En esta sección sólo se estudiarán los atributos que pueden ser medidos mediante el Modelado Conceptual que OO-Method tiene en la actualidad. Sin embargo, en secciones futuras se propone incluir nuevas primitivas de modelado en el Modelado Conceptual de OO-Method para medir este tipo de atributos.

Las métricas propuestas en esta sección están basadas en una fórmula matemática, luego el valor que se obtiene es totalmente objetivo. Esto favorece que se puedan aplicar las métricas de forma automática, obteniendo la medición de forma instantánea y sin que el analista tenga que realizar ningún tipo de cálculo. De esta forma, el analista puede calcular el valor de la usabilidad del sistema antes de generarlo. Las métricas que formarían parte del algoritmo de medición son las siguientes:

4.2.1.1. Facilidad de aprendizaje

- **Completitud de documentación**: este atributo indica si el sistema tiene documentación asociada que el usuario puede utilizar. En la herramienta de modelado que utiliza OO-Method (OLIVA**NOVA**), en algunas de las primitivas de modelado se puede añadir un mensaje de ayuda. El porcentaje de mensajes de ayuda que se hayan escrito en todo el Modelo Conceptual, dará la medida de usabilidad de este atributo.

$$\forall x \in Atributos \vee Servicios \vee Argumentos \vee PatronIntroduccion \vee SeleccionDefinida \vee IIU \vee PIU \vee SIU \vee MD \vee Acciones \vee Navegaciones \vee CriterioOrdenación \vee Filtro:$$

$$\frac{\sum MensajeAyuda(x)}{\sum x} = CDD$$

Utilizar este atributo como medida de la usabilidad implica modificar el actual Compilador de Modelos de OO-Method. Actualmente los mensajes de ayuda escritos

en las distintas primitivas de modelado no son tratados por el Compilador de Modelos y por tanto no son incorporados en el sistema. Sin embargo, el Compilador de Modelos se podría mejorar para que los mensajes de ayuda introducidos en el modelo generaran la documentación del sistema.

- **Determinación de la acción:** este atributo se utiliza para medir la facilidad con que el usuario puede comprender el significado de sus acciones antes de que las ejecute. El mecanismo más importante para que el usuario interprete la funcionalidad de la acción es la etiqueta que tenga ésta asociada. En OO-Method, las acciones que puede realizar un usuario se modelan mediante los servicios de las clases en el Modelo de Objetos. Las etiquetas que presentarán las acciones en la aplicación final se introducen como alias de los servicios en el Modelo de Objetos. En caso de que el analista no introduzca alias, la etiqueta utilizada en la aplicación final es el nombre del servicio. Normalmente el nombre del servicio no es un buen identificador del servicio para el usuario, ya que está restringido a 32 caracteres y no acepta espacios en blanco ni caracteres especiales. Por ello, los servicios con alias por defecto (el nombre del servicio) son poco usables.

  Además de las acciones que implementan la lógica de negocio del sistema, también hay una serie de acciones en las aplicaciones generadas con OO-Method que ayudan al listado de instancias. Estas acciones son el Filtrado y la Ordenación de un conjunto de instancias. El nombre de estas acciones se define mediante un alias en el Modelo de Presentación. El hecho de incluir un alias en la definición de la primita “Filtro” y “Criterio de ordenación” distinto del nombre, ayuda a entender el resultado que se obtendrá al ejecutar ambos mecanismos.

$$\forall x \in Servicio \vee Filtro \vee Ordenacion : \frac{\sum AliasDist\,\mathrm{int}\,oDeNombreYdeNulo(x)}{\sum x} = DA1$$

  Otra primitiva que afecta a la medida de este atributo es la llamada “Información Complementaria” en OO-Method. Por defecto, cuando se selecciona un objeto para dar valor a un argumento objeto valuado, solo se muestra al usuario el OID del objeto. El OID no es un valor representativo de la instancia para el usuario. El patrón de Información Complementaria es una primitiva del Modelo de Presentación que muestra para un objeto seleccionado como argumento objeto valuado, el valor de algunos de sus atributos. La proporción de patrones de “Información complementaria” por argumento objeto valuado en una UI de Servicio es una buena medida también para predecir lo que producirá una acción.

$$\forall x \in InformacionSuplementaria, \forall y \in ArgumentoObjetoValuado : \frac{\sum x}{\sum y} = DA2$$

4.2.1.2. Comprensibilidad

- **Agrupación por disposición:** este atributo mide si los elementos de la interfaz que tienen algo en común se encuentran agrupados de alguna forma. Esta agrupación cobra importancia cuando la cantidad de información mostrada en la interfaz es grande. Automáticamente el Compilador de Modelos hace algunas agrupaciones de elementos. Por ejemplo, en el caso de una UI Maestro-Detalle, las UI de Población que la forman están agrupadas por pestañas si éstas superan el número de dos unidades. De esta forma todas las instancias de una misma UI se encuentran dentro de una pestaña en la interfaz final.

  En cambio, a la hora de organizar los campos de entrada utilizados en la ejecución de un servicio, su agrupación depende totalmente del analista. Por defecto, el modelador de OO-Method introduce todos los argumentos en el mismo grupo. Es el analista el encargado de agruparlos coherentemente a través de la primitiva de modelado perteneciente al Modelo de Interacción llamada “Agrupación de argumentos”. Esta primitiva es necesaria cuando el número de argumentos que se deben rellenar para ejecutar el servicio de una UI de Servicio es elevado. Además, tal como indica [21] la

agrupación de ítems similares mejora la lectura y resalta las relaciones entre diferentes grupos de datos. Siguiendo las guías de usabilidad [24], no hay que recargar la pantalla con muchos componentes (argumentos en el caso de OO-Method). Más de 20 argumentos por grupo recargaría mucho la pantalla. Además, cuando hay tantos argumentos es muy probable que se pueda establecer alguna relación entre ellos para agruparlos en distintos grupos y hacer la interfaz más usable. Por lo tanto, la fórmula utilizada para medir este atributo contará la media del número de argumentos que se mostrarán entre cada uno de los grupos de argumentos definidos. Más adelante ya se interpretará el resultado de esta fórmula mediante los indicadores de este atributo.

$$\forall x \in GrupoDeArgumentos(Argumentos) : \frac{\sum_{i=1}^{n} x_i}{\sum x} = APD$$

- **Densidad de la información:** indica la forma en que el sistema muestra al usuario la cantidad adecuada de información por cada interfaz. Este atributo es un buen indicador de que no se muestre demasiada información al usuario. Siguiendo las guías de usabilidad [24][28] el usuario no debería utilizar la barra de desplazamiento para visualizar la información del sistema. Toda la información mostrada debe caber en la interfaz sin utilizar ningún mecanismo de desplazamiento.

  En OO-Method, este atributo se debe medir teniendo en cuenta distintas primitivas de modelado que se encargan de modelar el conjunto de información mostrada en una interfaz. Las primitivas a tener en cuenta se pueden agrupar en dos conjuntos: primitivas utilizadas en las UI de Servicio para agrupar los argumentos que el usuario debe rellenar para ejecutar el servicio; y las primitivas utilizadas para listar instancias en las UI de Población.

  La primitiva de modelado que gestiona la agrupación de argumentos en una UI de Servicio es la llamada “Agrupación de argumentos”. Tal y como se ha comentado en fórmulas anteriores, no deberían existir grupos de argumentos con más de 20 elementos para ajustarse a las guías de usabilidad. Por lo tanto la fórmula utilizada en el atributo “Agrupación por disposición” para contar los argumentos de entrada puede ser también un indicador de la usabilidad del atributo “Densidad de la información”.

$$\forall x \in GrupoDeArgumentos(Argumentos) : \frac{\sum_{i=1}^{n} x_i}{\sum x} = DI1$$

  Las primitivas que se pueden utilizar en la medición de este atributo para el caso de las UI de Población son varias:

  - Número de servicios en una UIP o UII: si existe gran cantidad de servicios que se pueden realizar sobre una instancia seleccionada, habrá que utilizar una barra de desplazamiento para mostrar todos los servicios. Con el Compilador de Modelos de OO-Method, en caso de que existan más de 13 servicios, el usuario debe utilizar la barra de desplazamiento. Por lo tanto para medir la usabilidad de este atributo es necesario obtener el promedio de servicios asociados a la UI de Población o a la UI de Instancia. Los servicios se asocian a las UIs mediante Elementos de Acción. Es decir, para obtener una métrica basta con contar los Elementos de Acción definidos para el patrón de Acción asociado a cada una de las UI de Población y de Instancia.

$$\forall x \in UIP \vee UII : \frac{\sum_{i=1}^{n} ElementosAccion(x_i)}{\sum x} = DI2$$

  - Número de atributos en una UIP o UII: el número de atributos de una clase que se mostrarán en la interfaz se modela mediante la primitiva “Conjunto de visualización”. Para que no se tengan que utilizar barras de desplazamiento [24][28], un conjunto de visualización usable, no debería contener más de 12 atributos (asumiendo que cada atributo contendrá unos 10 caracteres). En OO-

Method, 12 atributos es la media de atributos que se pueden visualizar sin necesidad de utilizar la barra de desplazamiento. Por lo tanto, la fórmula que mida este aspecto debe obtener el promedio de atributos que contienen los conjuntos de visualización de las UI de Población y de Instancia.

$$\forall x \in UIP \vee UII : \frac{\sum_{i=1}^{n} AtributosDelConjuntoVisualización(x_i)}{\sum x} = DI3$$

- Número de navegaciones en una UIP o UII: en las aplicaciones generadas con OO-Method, las navegaciones asociadas a una instancia se modelan mediante una botonera, donde cada botón es una navegación disponible. En el caso que existan más de 12 navegaciones (asumiendo que cada alias de la navegación ocupa 10 caracteres), será necesaria una barra de desplazamiento para mostrar todos los botones de la navegación y por tanto la usabilidad de la aplicación será menor [24][28]. El mecanismo de navegación se modela a través de la primitiva de modelado llamada “Navegación”. Por lo tanto, la fórmula para medir esta propiedad debe obtener el promedio de Elementos de navegación por UI de Población y de Instancia.

$$\forall x \in ElementoDeNavegación, \forall y \in UIP \vee UII : \frac{\sum x}{\sum y} = DI4$$

- Número de filtros definidos: uno de los mecanismos que tiene OO-Method para la búsqueda de instancias en una lista son los Filtros. La definición de esta primitiva se utiliza en aquellas UI de Población que albergarán varias instancias. La fórmula para medir este aspecto de la usabilidad debe por tanto obtener el promedio de UIs de Población que tengan al menos un filtro definido [21][27]. Es importante remarcar que puede que existan UIs de Población en las cuales la no existencia de Filtros no indica una baja usabilidad, ya que la lista de instancias tendrá pocos elementos. Por eso, el porcentaje de uso de los Filtros no debe de ser del 100% en todas las UIs de Población para garantizar una buena usabilidad, pero sí que debe haber un elevado porcentaje de los mismos.

$$\forall x \in UIPconAlMenosUnFiltro, \forall y \in UIP : \frac{\sum x}{\sum y} = DI5$$

- Número de variables de filtro: los filtros utilizados en las interfaces utilizan una serie de argumentos para el filtrado llamados “Variables de filtro”. Mediante estas variables, el usuario introduce los parámetros para realizar el filtrado. Si existen gran cantidad de variables de filtro, queda menos espacio para el listado de instancias y por lo tanto, se necesitará utilizar una barra de desplazamiento para visualizar las instancias [24][28]. Por lo tanto, la existencia de más de 4 variables de filtro disminuye la usabilidad del sistema. La fórmula utilizada para medir este aspecto de la usabilidad debe calcular el promedio de variables de filtro utilizadas en cada uno de los Filtros del sistema.

$$\forall x \in VaribleDeFiltro, \forall y \in Filtro : \frac{\sum x}{\sum y} = DI6$$

- Número de criterios de ordenación: en OO-Method existe una primitiva para ordenar las instancias listadas según ciertas propiedades de los objetos. El usuario al ordenar las instancias puede localizar más fácilmente la instancia buscada [41][27]. Por lo tanto, la usabilidad del sistema aumentará si se utiliza el criterio de ordenación para listados donde participen muchas instancias. Al igual que el uso de los filtros, el uso de un criterio de ordenación solo tiene sentido para listados con muchas instancias. La fórmula para obtener un valor que mida este aspecto solo debe obtener el porcentaje de UIs de Población con un criterio de ordenación definido.

$$\forall x \in UIPconAlMenosUnCriterioOrdenacion, \forall y \in UIP : \frac{\sum x}{\sum y} = DI7$$

- **Significación del etiquetado:** este atributo se encarga de medir que las etiquetas mostradas al usuario sean significativas. Las etiquetas asociadas a la ejecución de alguna acción se miden a través del atributo "Determinación de la acción" anteriormente explicado. El atributo "Significación del etiquetado" mide si el resto de etiquetas de la interfaz son significativas. En concreto se encarga de medir si son significativas las etiquetas de los campos de entrada de datos y las etiquetas de cada uno de los elementos del conjunto de visualización. Estos dos tipos de etiquetas se obtienen por defecto de los alias definidos para cada uno de los atributos del Modelo de Objetos. Sin embargo, el analista puede cambiarlos manualmente en los alias de cada uno de ellos respectivamente. En caso de que el analista no introduzca ningún alias para el atributo, la etiqueta mostrada al usuario en ambos casos será el nombre del atributo.

  La medición automática de este atributo solo puede estar asociada a una valoración sintáctica del etiquetado, ya que semánticamente solo puede ser evaluado por el analista. Esta métrica, aunque parezca vaga, permite hacerse una idea de la significación del etiquetado, porque si un alias está relleno, significa que el analista ha introducido un valor que ha considerado como el más idóneo para ese atributo, y por lo tanto se puede intuir que será semánticamente adecuado.

$$\forall x \in AtributoConAliasDefinido, \forall y \in Atributos \frac{\sum x}{\sum y} = SDE$$

- **Completitud de valores iniciales:** este atributo mide el uso que se ha hecho de los valores iniciales en el modelado del sistema para la introducción de datos por parte del usuario. El uso de los valores iniciales facilita la introducción de datos y ayuda al usuario a entender el tipo de valor que debe introducir y por lo tanto hace aumentar la usabilidad del sistema. En OO-Method, los valores por defecto se definen en los argumentos de entrada de un servicio. De esta forma, los valores por defecto se mostrarán en los campos de entrada de una UI de Servicio.

  Sin embargo, no en todos los argumentos de los servicios es recomendable utilizar el mecanismo de valores por defecto. Por lo tanto, con que haya una mínima porción de argumentos con valores iniciales, ya indica que el analista se ha preocupado de definir valores iniciales allá a donde era más conveniente. La fórmula para obtener la porción de argumentos con valores por defecto es la siguiente:

$$\forall x \in ArgumentoConValorInicial, \forall y \in Argumento : \frac{\sum x}{\sum y} = CVI$$

- **Calidad de los mensajes:** este atributo mide la calidad de los mensajes que se muestran al usuario, por ejemplo cuando se produce un error en la ejecución. En OO-Method muchos de los mensajes que se muestran son genéricos y los crea el Compilador de Modelos sin la intervención del analista. En cambio, hay casos en los que el analista introduce en la fase de análisis el texto que se mostrará. Este es el caso del mensaje de error que se muestra cuando el usuario solicita ejecutar un servicio cuya precondición es falsa. En OO-Method, los únicos mensajes que se muestran al usuario y son configurables, son los mensajes asociados a errores en la ejecución de los servicios.

  Para medir este atributo con total precisión, dicha medición la debería realizar el propio usuario en la aplicación final. Ya que es el usuario el que puede indicar si ha entendido o no los mensajes. Por lo tanto, es un atributo basado fundamentalmente en la semántica de los mensajes, y escaparía a la medición automática que se puede realizar en el Modelado Conceptual. Sin embargo, es posible hacer una predicción de la calidad de estos mensajes basándose en la longitud de estos mensajes. Según las guías de usabilidad [24], la longitud de los mensajes debería estar entre 6 y 20 palabras. Los mensajes cuya longitud esté comprendida en ese rango se puede asumir que son mensajes de calidad. De esta forma, la medición de la usabilidad de este

atributo está sujeta al promedio de las longitudes de todos los mensajes introducidos por el analista que se mostrarán al usuario en los mensajes de error.

$$\forall xMensajeError : \frac{\sum_{i=1}^{n} Longitud(x_i)}{\sum x} = CDM$$

- **Navegabilidad:** este atributo mide la usabilidad de la funcionalidad de navegar hacia información relacionada con la que se muestra en la actual interfaz. En OO-Method las posibles navegaciones se hacen solo sobre instancias mostradas en la UI de Población y de Instancia. De una instancia se puede navegar hacia instancias relacionadas, siempre que exista una relación estructural entre ellas definida en el Modelo de Objetos. Si no se define un Patrón Elemental de Navegación para una UI de Población o de Instancia, por defecto se crean navegaciones de la clase representada en la UI hacia las clases con las que ésta esté directamente relacionada en el Modelo de Objetos.

  Para medir este atributo de usabilidad, se define una métrica que mida las navegaciones definidas explícitamente por el analista, es decir, aquellos que no están definidas por defecto. Normalmente las navegaciones por defecto no suelen ser las más adecuadas para la lógica de negocio por tres motivos:

  1. No contemplan la navegación indirecta, es decir, la navegación en la que la clase destino se alcanza atravesando varias clases del Modelo de Objetos.

  2. Genera navegaciones de la clase origen hacia todas las clases que estén relacionadas estructuralmente con ella. Esto hace que se muestren al usuario navegaciones que no debería visualizar o que le sean de poca ayuda y por lo tanto solo recarguen la funcionalidad del sistema innecesariamente.

  3. Las navegaciones por defecto no tienen definido un alias que aporte un significado a la navegación. En las navegaciones por defecto se toma el alias de la clase destino de la navegación. Este nombre puede que no sea el más indicado para representar el destino de la navegación. Solo se garantiza que la etiqueta de la navegación es semánticamente correcta si ésta la introduce explícitamente el analista, y para esto la navegación no puede ser la existente por defecto.

  Por todo ello, se contarán las navegaciones definidas por el analista entre todas las UIs de Población y de Instancia del sistema. A mayor número de navegaciones definidas, mejor usabilidad tendrá el sistema final.

$$\forall x \in NavegacionesDefinidas, \forall y \in UIP \vee UII : \frac{\sum x}{\sum y} = NV$$

4.2.1.3. Operabilidad

- **Consistencia de orden:** este atributo se utiliza para medir si el orden de los elementos estáticos de la interfaz están en el mismo orden a lo largo de todo el sistema. Los elementos son ubicados en la misma posición en todas las aplicaciones generadas por el Compilador de Modelos. Por ejemplo, los filtros en la parte superior de la pantalla de una UI de Población, las navegaciones en la parte inferior de la pantalla de una UI de Población o de Instancia, etc. Sin embargo, sí que se puede modelar el orden de los componentes de algunos de los Patrones Elementales del Modelo de Presentación. Las métricas que se pueden definir sobre algunos Patrones Elementales son:
  - Orden en los argumentos: los servicios de crear y modificar instancias de una misma clase tienen los mismos argumentos. Los únicos argumentos que no están en el servicio de modificar son aquellos que representan a un atributo constante y que solo están en el evento de creación. Al tener prácticamente los mismos argumentos, sería más sencillo para el usuario que en ambas pantallas los campos de entrada aparecieran en el mismo orden. Sin embargo, puede ser que el orden de los campos de entrada que se muestren al usuario sean distintos en las UI que implementan ambos servicios. El orden de los

campos de entrada en una UI de servicios viene determinado por el que se define en la UI de Servicio. Si no se modifica este orden, por defecto coincide con el orden en que se definieron los argumentos del servicio.

$$\forall x \in MismoOrden(Crear, Modificar), \forall y \in ClasesCon(Crear, Modificar) : \frac{\sum x}{\sum y} = CO1$$

- Orden de los servicios: las aplicaciones que mantienen un orden similar en la disposición de los controles que realizan una acción similar, son más usables que las que no mantienen este orden [27]. En OO-Method, las UI de Población y de Instancia muestran una lista de botones (Elementos de Acción) que invocan a los servicios definidos para las instancias de la que representan. Aunque cada clase contiene un conjunto de servicios característicos de ella misma que se derivan de la lógica del negocio, existen otros servicios que son comunes en casi todas las clases. Son los servicios de crear, modificar y eliminar.

  El orden de aparición de los botones que lanzan estos servicios debería ser el mismo en todas las UIs. Este orden lo define el analista en el Patrón Elemental de Acción del Modelo de Presentación y por tanto puede ser medible en el Modelo Conceptual. En caso de que el analista no defina el patrón de Acción, el orden en el que se muestran las acciones es el mismo orden que tienen los servicios de la clase sobre la que se han definifo las acciones.

  La métrica utilizada consiste en obtener el porcentaje de UIs que mantienen en su patrón de Acción el orden lógico de: crear, modificar y eliminar.

$$\forall x \in AccionConEvento(Crear, Modificar, E\lim inar) : \frac{\sum Ordenados(x)}{\sum x} = CO2$$

- Orden de los campos: los campos utilizados para identificar a las instancias deberían ser los primeros en mostrarse [24]. Esto facilitaría una identificación rápida de las instancias por parte del usuario, ya que leyendo el primer campo, identificaría la instancia. En OO-Method se pueden diferenciar dos tipos de campos:

  1. Campos de entrada: son los campos que el usuario rellena mediante datos para la ejecución de un servicio. En OO-Method estos campos se corresponden con los argumentos de la UI de Servicio. El orden en que se muestran estos argumentos se define en las propiedades de la UI de Servicio. Por lo tanto es ahí donde se puede calcular el porcentaje de UI de Servicio con orden en sus argumentos. En caso que el analista no defina ningún orden, el orden de los argumentos en la UI de Servicio viene dado por el orden en que se definieron los argumentos del servicio en el Modelo de Objetos.

  2. Campos que muestran atributos del objeto: son campos que muestran el valor de los atributos de la instancia en la UI de Población y de Instancia. El orden en que se visualizan los atributos de las instancias viene determinado por el Patrón Elemental llamado Conjunto de Visualización. Este patrón se utiliza en las UI de Población y de Instancia para mostrar atributos de la clase. En caso de que el analista no defina un Conjunto de Visualización, el orden viene impuesto por el orden en el que se han definido los atributos de la clase representada mediante la UI. La métrica para medir este aspecto es obtener el porcentaje de UIs que muestran sus atributos identificadores en primer lugar, es decir, aquellas UIs que tienen sus atributos ordenados. Para ello solo hay que contar los Conjuntos de Visualización que estén ordenados (tanto los definidos por el analista como los que hay por defecto), y el número de UIs de Población y de Instancia.

  La fórmula que engloba los dos tipos de campos es la siguiente:

$$\forall x \in ConjuntoCamposAsociadoaUI, \forall y \in UIP \vee UII \vee UIS : \frac{\sum Ordenados(x)}{\sum y} = CO3$$

- **Consistencia de etiquetado:** este atributo indica si las distintas etiquetas de un sistema que tienen el mismo significado, se llaman de la misma forma. Aspectos de la interfaz con la misma semántica deberían tener el mismo nombre, de forma que el usuario los identificara como iguales. Este atributo se mide utilizando varias métricas. Cada una de estas métricas se define sobre un elemento concreto del Modelo de Presentación de OO-Method.
    - Consistencia en las navegaciones: todas las navegaciones hacía instancias de la misma clase deberían tener definido el mismo alias. De esta forma el usuario detectaría que la instancia destino es en realidad la misma en todos los casos en que coincida el alias. La fórmula para medir este aspecto obtiene el porcentaje de navegaciones que apuntan al mismo destino con el mismo alias.

$$\forall x \in DestinoDeNavegaciónYaExistente : \frac{\sum AliasIguales(x)}{\sum x} = CE1$$

    - Consistencia en las acciones: las acciones en OO-Method se definen a través del Patrón Elemental de Acción. El patrón de Acción está compuesto por elementos de acción. Cada uno de estos elementos de acción invocan a una UI de Servicio. Las etiquetas de los elementos de acción que hagan referencia al mismo servicio, deberían tener el mismo alias. Esto facilitaría la comprensión del usuario. Por lo tanto este aspecto se mide mediante una fórmula que obtenga el porcentaje de alias iguales que hagan referencia mismo servicio.

$$\forall x \in \text{Re}\ ferenciaServicioYaExistente : \frac{\sum AliasIguales(x)}{\sum x} = CE2$$

    - Consistencia del Conjunto de Visualización: sobre una UI de Población o de Instancia se pueden definir varios Patrones Elementales del tipo Conjunto de Visualización. Este tipo de patrón indica los atributos visibles del conjunto de instancias. Aunque el conjunto de atributos sea distinto para dos Conjuntos de Visualización definidos sobre la misma clase, habrá muchos otros atributos que representarán al mismo atributo de la clase. En este caso, el alias de todos los Conjuntos de Visualización debe ser el mismo. De esta forma se asegura que el alias mostrado al usuario es el mismo en todos los Conjuntos de Visualización que hagan referencia al mismo atributo de la clase. La métrica para medir este aspecto obtiene el porcentaje de alias que son iguales para un mismo atributo de una clase referenciado.

$$\forall \text{Re}\ ferenciaAtributoyaExistenteEnConjuntoVis : \frac{\sum AliasIguales(x)}{\sum x} = CE3$$

    - Consistencia de las Variables de Filtro: las Variables de Filtro son campos de entrada que el usuario debe rellenar según los parámetros por los que desee filtrar el listado de instancias de una UI de Población. Estas Variables de Filtro se agrupan en Filtros. Cada una de las Variables de Filtro hacen referencia a un atributo de una de las clases del sistema o a una clase entera. A su vez, un mismo atributo o clase puede participar como Variable de Filtro en más de un Filtro. En este caso, se debe asegurar que el nombre de la Variable de Filtro es el mimo en todos los Filtros en los que se utilice esa variable.

      En OO-Method, cada Variable de Filtro lleva asociado un alias, que es el identificador que se muestra al usuario. En caso de que el analista no introduzca un alias, el alias es el nombre de la Variable de Filtro. Por lo tanto, para asegurar la consistencia entre Variables de Filtro que hacen referencia al mismo atributo o clase, solo hay que verificar si los alias de esas Variables del Filtro son idénticos.

$$\forall \text{Re } ferenciaYaExistenteEnVarFiltro : \frac{\sum AliasIguales(x)}{\sum x} = CE4$$

- **Prevención de errores:** este atributo mide la capacidad del sistema de adelantarse a futuros errores que pueda cometer el usuario. De esta forma se puede evitar que el usuario cometa el error facilitándole la información necesaria en cada momento. En la Ingeniería del Software son varios los mecanismos existentes para anticiparse a futuros errores que pueda cometer el usuario. Algunos de estos mecanismos son el uso de las máscaras, la definición de rangos de valores posibles, la definición de categorías, etc. De todos estos, OO-Method soporta dos en su Modelo de Presentación: la definición de máscaras (Patrón de Introducción) y la selección de un valor de entre un conjunto cerrado de posibles valores (Patrón de Selección Definida).

  Sin embargo, el uso de las primitivas de Introducción y de Selección Definida es difícil de justificar en el Compilador de Modelos de forma automática. Son primitivas que dependen de los requisitos de cada uno de los sistemas en construcción. Habrá sistemas que utilicen mucho estos mecanismos y habrá que los utilicen con mayor frecuencia y en ambos casos la usabilidad puede ser la misma.

  Una posible métrica para la Selección Definida, podría estar basada en la longitud de los argumentos de entrada y su tipo. Los argumentos de tipo string cuya longitud sea menor de 4 son candidatos a utilizar el patrón de Selección Definida, según las guías de usabilidad [24]. Normalmente, este tipo de argumentos solo puede recibir un pequeño conjunto de valores. Por ejemplo los valores: Si/No, V/M, Lun/Mar/Mie/Jue/Vie/Sab/Dom, etc. Por lo tanto la fórmula para medir este atributo obtendría la proporción de los argumentos de tipo string con tamaño menor de 4 que tienen Selección Definida asociada:

$$\forall x \in Long(ArgumentoTypoString) < 4 : \frac{\sum SelecciónDefinida(x)}{\sum x} = PDE$$

  Que se utilice en estos casos la Selección Definida, no implica que sólo se utilice en ellos. Este patrón tiene muchas más aplicaciones que escapan a la medición automática ya que depende exclusivamente de los requisitos del sistema, Por ejemplo, en una aplicación de alquiler de coches, un listado con los colores de los coches serían strings con más de 4 caracteres.

4.2.2. Indicadores

En la sección anterior se han mostrado las fórmulas para obtener un valor numérico por cada uno de los atributos. Incluso, algunos atributos tienen más de una fórmula matemática para medir su usabilidad, ya que dependen de varias primitivas de modelado. Por ejemplo el atributo "determinación de la acción" tiene la fórmula DA1 y DA2. Sin embargo, los números obtenidos mediante estas fórmulas no aportan ningún significado al valor de la usabilidad que representan. Hay que dotar a estos números con una semántica adecuada para interpretar la cantidad de usabilidad que representan.

No todas las fórmulas utilizadas devuelven valores numéricos que oscilan entre los mismos rangos. Hay fórmulas que van de 0 a 0.99 (por ejemplo CDD) y otras que van de 0 a infinito (por ejemplo APD). Por lo tanto, no todas las fórmulas utilizan las mismas unidades de medida. Sin embargo, para poder medir la usabilidad de los atributos, todas las fórmulas deberían ser medidas con las mismas unidades. Para igualar las unidades de medida de todas las fórmulas, se les debe asignar un valor cualitativo a cada uno de los números obtenidos. De esta forma se garantiza que todas las fórmulas obtendrán un valor basado en la misma unidad de medida, ya que los valores que pueden tomar pertenecen aun mismo conjunto para todas las fórmulas. Los valores cualitativos que se obtendrán de cada una de las fórmulas solo pueden ser cinco: Muy Bueno (MB), Bueno (B), Regular (R), Malo (M), Muy Malo (MM). Para asignar uno de estos valores cualitativos a las fórmulas, hay que definir unos rangos que en base al valor numérico obtenido en la fórmula, se le asigne uno de estos valores cualitativos al atributo.

Cada uno de los distintos rangos para definir los indicadores se han construido en base a un conjunto de guías de usabilidad (por ejemplo los de Leavit y Shneiderman [24]) y heurísticos (por ejemplo los de Nielsen [27]), que proporcionan una serie de buenas prácticas para que los sistemas sean usables. Los criterios seguidos para la definición de los rangos para cada una de las fórmulas están explicados en el punto anterior, en la sección donde se definen cada una de las fórmulas. Con estos indicadores, se puede saber entre qué valores de la fórmula el atributo tendrá un valor usable o no.

Los rangos utilizados como indicadores se pueden refinar mediante la experimentación empírica. Para ello se deben utilizar aplicaciones generadas con OO-Method en la que una serie de usuarios realizarán una serie de tareas concretas. Las tareas a realizar deben ser las mismas para todos los usuarios, de esta forma se garantiza que los criterios a evaluar serán los mismos en todos los usuarios. Una vez el usuario ha realizado las tareas, debe rellenar un cuestionario para evaluar la usabilidad del sistema. En este cuestionario se le formularán al usuario preguntas para registrar cuánto de usable ha percibido que es la aplicación. La validación de los indicadores consiste en medir a través del cuestionario, los mismos atributos que se han medido automáticamente en el Modelado Conceptual. Así, se pueden comparar los valores obtenidos automáticamente de los Modelos Conceptuales y la percepción de la usabilidad que tiene el usuario final. La experimentación empírica demostraría cuanto de acertados son los indicadores.

A continuación se presenta una tabla con los indicadores de cada una de las fórmulas descritas en la sección anterior. En la tabla “Medida” se almacenan las abreviaturas de cada una de las fórmulas utilizadas para la medición de los atributos. En el resto de columnas se muestran los rangos entre los cuales puede oscilar el valor de las fórmulas para asignarle a la fórmula un valor cualitativo: Muy Bueno (MB); Bueno (B); Regular (R); Malo (M); Muy Malo (MM).

Medida	MB	B	R	M	MM
CDD	CDD≥.95	.95>CDD≥.85	.85>CDD≥.75	.75>CDD≥.65	CDD <.65
DA1	DA1≥.95	.95> DA1≥.85	.85> DA1≥.75	.75> DA1≥.65	DA1<.65
DA2	DA2≥.95	.95> DA2≥.85	.85> DA2≥.75	.75> DA2≥.65	DA2<.65
APD	APD≤ 15	15< APD ≤20	20< APD ≤25	25< APD ≤30	APD >35
DI1	DI1≤ 15	15< DI1 ≤20	20< DI1 ≤25	25< DI1 ≤30	DI1 >35
DI2	DI2≤ 10	10< DI2 ≤13	13< DI2 ≤16	16< DI2 ≤19	DI2 >19
DI3	DI3≤ 7	7< DI3 ≤10	10< DI3 ≤13	13< DI3 ≤16	DI3 >16
DI4	DI4≤ 9	9< DI4 ≤12	12< DI4 ≤15	15< DI4 ≤18	DI4 >18
DI5	DA2≥.90	.90> DA2≥.80	.80> DA2≥.70	.70> DA2≥.60	DA2<.60
DI6	DI6≤ 3	3< DI6 ≤5	5< DI6 ≤7	7< DI6 ≤9	DI6 >9
DI7	DI7≥.85	.85> DI7≥.70	.70> DI7≥.55	.55> DI7≥.40	DI7<.40
SDE	SDE ≥.95	.95> SDE ≥.85	.85> SDE ≥.75	.75>SDE ≥.65	SDE <.65
CVI	CVI ≥.20	.20> CVI ≥.15	.15> CVI ≥.10	.10> CVI ≥.5	CVI <.5
CDM	CDM ≤ 15	15< CDM ≤25	25< CDM ≤35	35< CDM ≤45	CDM >45
NV	NV ≥.90	.90> NV ≥.80	.80> NV ≥.70	.70> NV ≥.60	NV <.60
CO1	CO1 ≥.95	.95> CO1 ≥.90	.90> CO1 ≥.85	.85>CO1 ≥.80	CO1 <.75
CO2	CO2 ≥.95	.95> CO2 ≥.90	.90> CO2 ≥.85	.85>CO2 ≥.80	CO2 <.75
CO3	CO3 ≥.95	.95> CO3 ≥.90	.90> CO3 ≥.85	.85>CO3 ≥.80	CO3 <.75
CE1	CE1 ≥.90	.90> CE1 ≥.80	.80> CE1 ≥.70	.70>CE1 ≥.60	CE1 <.60
CE2	CE2 ≥.90	.90> CE2 ≥.80	.80> CE2 ≥.70	.70>CE2 ≥.60	CE2 <.60
CE3	CE3 ≥.90	.90> CE3 ≥.80	.80> CE3 ≥.70	.70>CE3 ≥.60	CE3 <.60

CE4	CE4 ≥.90	.90> CE4 ≥.80	.80> CE4 ≥.70	.70>CE4 ≥.60	CE4 <.60
PDE	PDE ≥.90	.90> PDE ≥.00	.80> PDE ≥.70	.70>PDE ≥.60	PDE <.60

Tabla 4. Rango de indicadores

4.2.3. Agregación

Una vez se han definido indicadores que asignan un valor cualitativo a cada uno de los valores numéricos obtenidos mediante las fórmulas, el último paso para calcular la usabilidad del sistema es el de agregar los resultados obtenidos al aplicar estos indicadores. Tal y como se muestra en la Figura 10, para obtener el valor de usabilidad del sistema, hay que agregar los valores de usabilidad de las subcaracterísticas. A su vez, para obtener el valor de usabilidad de las subcaracterísticas, hay que agregar el valor de los atributos, que a su vez pueden depender de varias fórmulas que también hay que agregar.

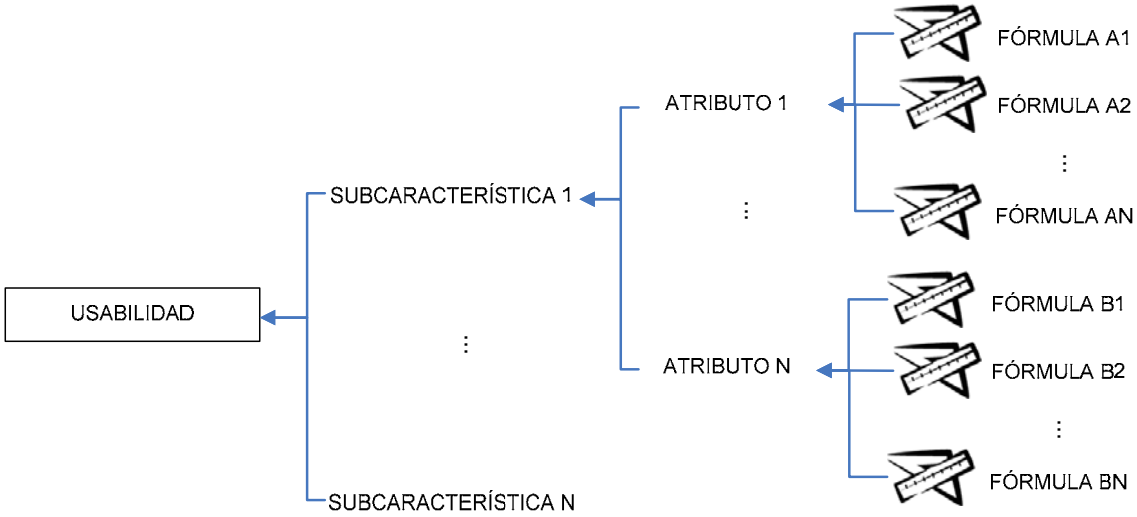

Figura 10. Proceso de agregación

En la literatura, hay trabajos relacionados con la medición de la usabilidad que utilizan como mecanismo de agregación la asignación de un peso a cada una de las ramas que forman el Modelo de Usabilidad. Estos trabajos están basados en los valores numéricos obtenidos de las fórmulas, no en sus valores cualitativos como plantea este trabajo. Un trabajo basado en la agregación por pesos es el desarrollado por Olsina [30]. En el trabajo de Olsina se distinguen dos tipos de agregaciones:

- La basada en modelos de puntuación aditiva lineal [16].
- La basada en modelos de puntuación multi-criterio y no lineales [10].

En ambos casos, la importancia de los indicadores se representa por medio de los pesos.

Supongamos por ejemplo que aplicamos la técnica de agregación por pesos al Modelo de Usabilidad propuesto. Para medir la subcaracterística SB que depende de los atributos A1, A2 y A3 que utilizan respectivamente las fórmulas F1, F2 y F3, la expresión de la agregación sería la siguiente:

A1 tiene un peso de 0,3 (P1)
A2 tiene un peso de 0,2 (P2)
A3 tiene un peso de 0,5 (P3)

$$\sum pesos = 1$$

F1xP1+F2xP2+F3xP3=SB

Una vez hechas las agregaciones para obtener todas las subcaracterísticas, debe utilizarse la misma técnica para agregar las subcaracterísticas y obtener un valor para la usabilidad. Por lo tanto, habría que definir un peso para cada una de las subcaracterísticas que componen la usabilidad de tal forma que la suma de estos pesos sea 1. La asignación de estos pesos dependería de la importancia que se quiera dar a cada una de las subcaracterísticas con respecto al resto.

Este caso de ejemplo solo es válido si los atributos A1, A2 y A3 dependen de una única fórmula (F1, F2 y F3 respectivamente). En el caso de que estos atributos dependan de más de una fórmula, se deben definir pesos para cada una de las fórmulas que definen el atributo (dependiendo de la importancia que se quiera dar a cada fórmula con respecto al atributo que miden), de forma que su suma sea 1. Con estos pesos, las fórmulas se agregarían para obtener el valor de los atributos y posteriormente se agregarían estos atributos con sus pesos asociados para obtener las subcaracterísticas.

Como puede apreciarse, este método de agregación es muy pesado y depende fuertemente de los pesos. Los pesos dependen de la importancia que se quiera dar a cada uno de los elementos que forman la agregación. En todo momento la suma de los pesos de la agregación debe sumar 1. Habría que definir un peso para cada uno de estos elementos:

- Por cada fórmula que comparta la medición de un atributo con más fórmulas
- Por cada atributo del Modelo de Usabilidad
- Por cada subcaracterística del Modelo de Usabilidad

Además, el valor de estos pesos puede cambiar de la evaluación de una aplicación a otra distinta dependiendo de la valoración que haga el usuario de los aspectos de la usabilidad. También, estos pesos se pueden refinar con experimentaciones futuras. Por tanto los valores de estos pesos cambian a lo largo del tiempo y su mantenimiento es costoso, ya que la cantidad de estos pesos es muy elevada en el Modelo de Usabilidad propuesto.

Debido al gran trabajo que representaría el mantener un repositorio de pesos, se ha optado por seguir una agregación basada en valores cualitativos. La agregación cualitativa se aplica en todos los conjuntos de la misma forma, ya se agrupen fórmulas, atributos o subcaracterísticas. No hay fórmulas con más peso que otras a la hora de realizar el cálculo de la usabilidad, ya que los criterios de agregación son los mismos para todas ellas. Por lo tanto, lo único que se debe refinar mediante la validación empírica al aplicar este método de valoración de la usabilidad son los indicadores utilizados en cada una de las fórmulas. Obteniendo indicadores precisos para las fórmulas se puede calcular la usabilidad de forma correcta sin la necesidad de utilizar pesos. A continuación se detalla cómo se realiza la agregación utilizada en este proyecto.

4.2.3.1. Agregación basada en valores cualitativos

La agregación utilizada en este trabajo está basada en la agregación de valores cualitativos. De esta forma se pueden agregar los valores cualitativos obtenidos mediante los indicadores. Las reglas utilizadas en este trabajo para agregar términos están basadas en el trabajo existente de Quispe y Condori [35] relacionado con la definición de reglas de reescritura de términos. Estas reglas de reescritura se utilizaban para semi-automatizar el proceso de evaluación de impacto de las decisiones en los requisitos no funcionales de un sistema. A su vez, su trabajo está basado en el trabajo de Requisitos No Funcionales (RNF) llevado a cabo por Chung et al [6]. El trabajo realizado por Chung permite tratar a los RNFs como metas difusas y las interacciones que existen entre este tipo de metas (interdependencias). Básicamente la noción de satisfacción de meta es el término sobre el cual se construye dicho marco. La evaluación de metas difusas e interdependencias determinan el impacto de decisiones en el logro de los RNFs.

El trabajo de Quispe y Condori amplía las reglas de reescritura de términos definidas en el trabajo de Chung et al. De todas las reglas de reescritura definidas en el trabajo de Quispe y

Condori, este trabajo solo utiliza funciones de aridad binaria, ya que el objetivo es agrupar los elementos tomando los componentes de 2 en 2. A continuación se detallan las reglas de agrupación derivadas del trabajo de Quispe y Condori:

- Las combinaciones entre términos que presentan en alguno de sus parámetros la constante Regular (R), genera como consecuencia el valor del otro término distinto a R. En caso que ambos valores sean R, el resultado de la combinación será R.

  $\forall x \in Indicadores : Combinación(x, R) \rightarrow x$

  Por ejemplo:

  $Combinación(MB, R) \rightarrow MB$

  $Combinación(M, R) \rightarrow M$

- Las combinaciones entre términos de agregación opuestos generan como resultado la constante Regular (R).

  $\forall x, \forall y \in Indicadores \mid x = \neg y : Combinación(x, y) \rightarrow R$

  Es decir, las reglas de agrupación serían:

  $Combinación(MB, MM) \rightarrow R$

  $Combinación(B, M) \rightarrow R$

- Las combinaciones restantes se definirán de la siguiente manera:

  $Combinación(MM, MM) \rightarrow MM$
  $Combinación(MM, M) \rightarrow MM$
  $Combinación(MM, B) \rightarrow MM$
  $Combinación(M, M) \rightarrow MM$
  $Combinación(B, B) \rightarrow MB$
  $Combinación(MB, M) \rightarrow MB$
  $Combinación(MB, B) \rightarrow MB$
  $Combinación(MB, MB) \rightarrow MB$

  Tal y como se puede apreciar en estas reglas de reescritura de términos, las combinaciones se hacen de 2 en 2. Por lo tanto, para aplicar la agregación se debe construir un árbol con nodos intermedios, donde cada uno de los nodos que forman el árbol solo tengan 2 hijos accesibles mediante ramas de longitud 1. Es decir, las fórmulas se agruparán de 2 en 2 hasta llegar a los nodos padre donde se definen los atributos. Los atributos a su vez se agruparán de 2 en 2 para definir las subcaracterísticas que se encontrarán como nodos padre de los atributos. Por último, las subcaracterísticas también se deben agrupar de 2 en 2 para obtener el valor de la usabilidad que es el nodo raíz de todo el árbol.

  La Figura 11 muestra el árbol de agregaciones construido para agregar el Modelo de Usabilidad representado en la Figura 10. En dicha figura se puede apreciar como se crean nodos auxiliares (los nodos sin nombre) utilizados para agrupar nodos hijos de 2 en 2 para poder aplicar las reglas de agrupación. Se deben crear tantos nodos intermedios sean necesarios para agrupar a los hijos en grupos de 2 elemento. Todo nodo intermedio solo puede tener dos hijos.

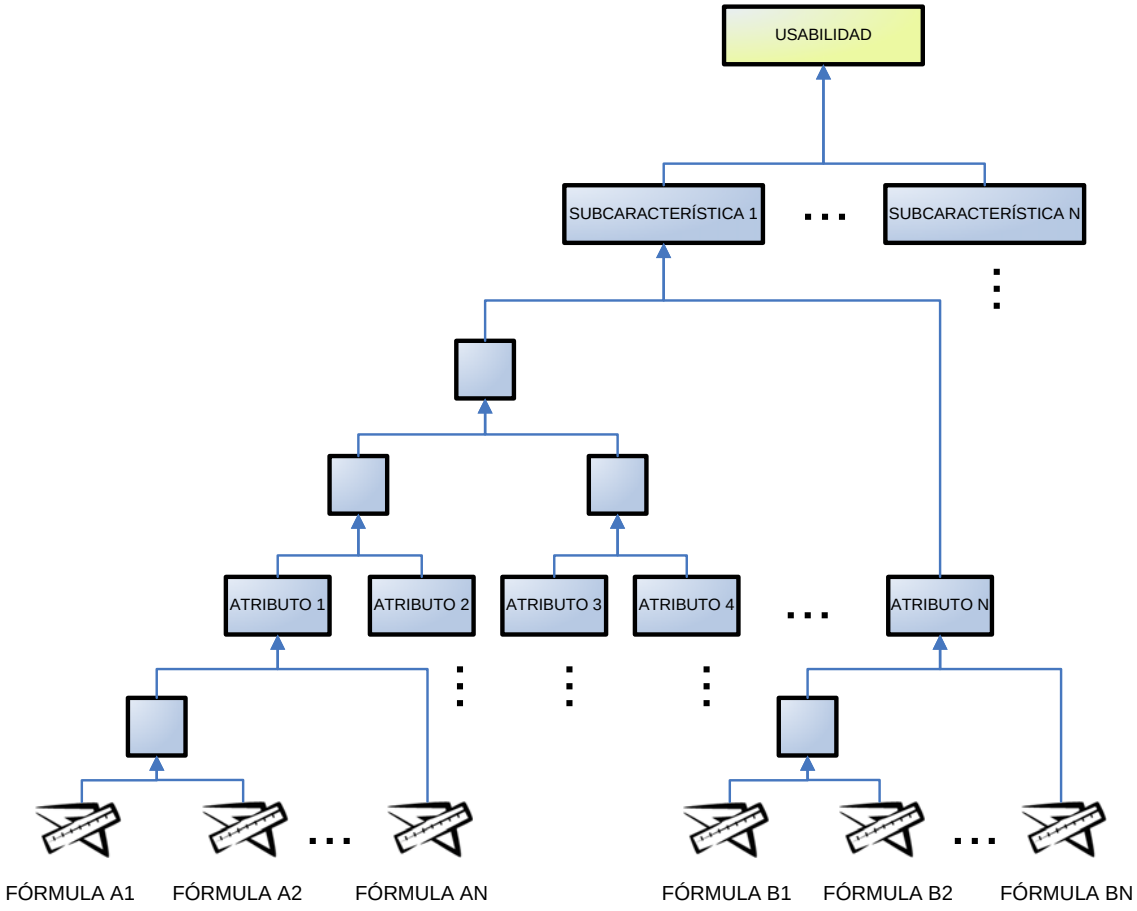

Figura 11. Árbol de agregaciones

5. VALIDACIÓN DE LAS FÓRMULAS

Tal y como se ha comentado en capítulos anteriores, el método propuesto para la medición de la usabilidad depende fuertemente de dos aspectos. Por un lado, de las fórmulas que utilizan las primitivas de modelado para obtener un valor numérico con el cual interpretar la usabilidad del sistema. Por otro lado, de los rangos de los indicadores obtenidos a partir de guías y heurísticos de usabilidad, que son los que dotan de significado al valor numérico obtenido mediante las fórmulas.

El valor de los indicadores se puede validar empíricamente a través de la experimentación con aplicaciones generadas a través de OO-Method, tal y como se ha comentado anteriormente. Estas experimentaciones también harían posible el ajustar los valores de los rangos de los indicadores.

Sin embargo, la validación que se plantea en este trabajo para las fórmulas es una validación teórica basada en el método de Poels llamado DISTANCE [34]. DISTANCE es una aproximación axiomática para la medición del software que define un conjunto de propiedades medibles que son a su vez necesarias y suficientes. En el campo de las matemáticas, un gran número de conceptos se definen usando este tipo de axiomas necesarios y suficientes. Según DISTANCE, cada función que satisface los axiomas de una métrica es por definición una medida válida de distancia si se asume una estructura de proximidad. La aproximación de Poels basada en la distancia para la medición del software describe los atributos del software por medio de estructuras de proximidad.

En la propuesta de Poels, cada atributo software se define como la distancia desde el producto software que se debe medir (o su abstracción) hasta alguna referencia al producto software (o abstracción). Ahora bien, si los atributos software se definen como distancias, entonces las métricas se pueden usar para medir los atributos del software. De ahí que las medidas del software tengan una definición axiomática, es decir, deben satisfacer los axiomas de la métrica.

La estructura de DISTANCE proporciona una serie de procedimientos para modelar atributos del software y definir las correspondientes medidas para ellos. Los diferentes pasos a llevar a cabo en los procedimientos se insertan en un modelo de proceso para la medición del software que:

1. Detalla para cada tarea las entradas necesarias, ocultando las suposiciones y los resultados esperados.
2. Recomienda el orden de ejecución, proporcionando ciclos de retroalimentación iterativa.
3. Incrusta los procedimientos de medida dentro de una aproximación de medición orientada a objetivos.

El trabajar con distancias conceptuales ofrece muchas ventajas, por ejemplo el concepto de similitud y no-similitud son bastante conocidos y fáciles de imaginar y utilizar en el día a adía. Además, las distancias son conceptos flexibles, ya que requieren un contexto explícito para la evaluación, es decir, las entidades son o no similares con respecto a ciertas características específicas. Por último, el concepto de no-similitud ha sido definido formalmente en la teoría de la medición por medio de estructuras de proximidad. Midiendo la distancia o la no-similitud por lo tanto se llega a la construcción de una representación de la proximidad.

Al aplicar este método al Modelo de Usabilidad propuesto, cada una de las fórmulas definidas para obtener un valor de la usabilidad, se modela como la distancia entre las entidades software que caracterizan y otras entidades software que sirven como puntos de referencia o normas de medida. Estas distancias se miden mediante funciones matemáticas que satisfacen el conjunto de axiomas del espacio de medida.

Las principales razones por las cuales se ha utilizado el método DISTANCE para la validación de las fórmulas que obtienen un valor para calcular la usabilidad son:

- Proporciona un marco que utiliza el concepto de distancia y disimilaridad basado en estructuras de proximidad [43].
- Provee un marco sistemático a través de un modelo de proceso a seguir para la construcción de una medida.
- Además, este marco ha sido aplicado en la validación teórica de métricas orientadas a objetos tales como:[34][15][7].

La Figura 12 muestra un diagrama de actividad con todos los procesos que participan en la utilización del proceso de medición de DISTANCE. Cada uno de los procesos que componen este diagrama se explican en detalle a continuación. El hecho de que las fórmulas definidas para el Modelo de Usabilidad propuesto se puedan representar mediante las fórmulas propuestas por DISTANCE garantiza que son fórmulas válidas, ya que se puede garantizar formalmente que se ha construido una representación de proximidad. Aquellas fórmulas que incumplan alguno de los procesos mostrados en la Figura 12 serán fórmulas no válidas y por lo tanto habría que sustituirlas por otras que sí cumplieran con todos estos procesos. A continuación se detallan todos los procesos de DISTANCE para la fórmula CDD (Completitud De la Documentación).

Es importante remarcar que el método de validación de métricas propuesto por Poels está pensado para validar únicamente métricas directas, es decir, métricas que no dependan de otras. Las fórmulas utilizadas para la medición de la usabilidad, todas están basadas en porcentajes y en medias, y por lo tanto dependen de 2 valores para obtener dichos porcentajes y medias: dependen de un numerador y de un denominador. El numerador hace el cálculo característico de la fórmula y el denominador es el encargado de comparar el valor del numerador con el resto del sistema (con el fin de proporcionar el porcentaje o la media). Solo se ha utilizado el método de Poels para validar los numeradores, ya que los denominadores son fácilmente deducibles a partir de los numeradores.

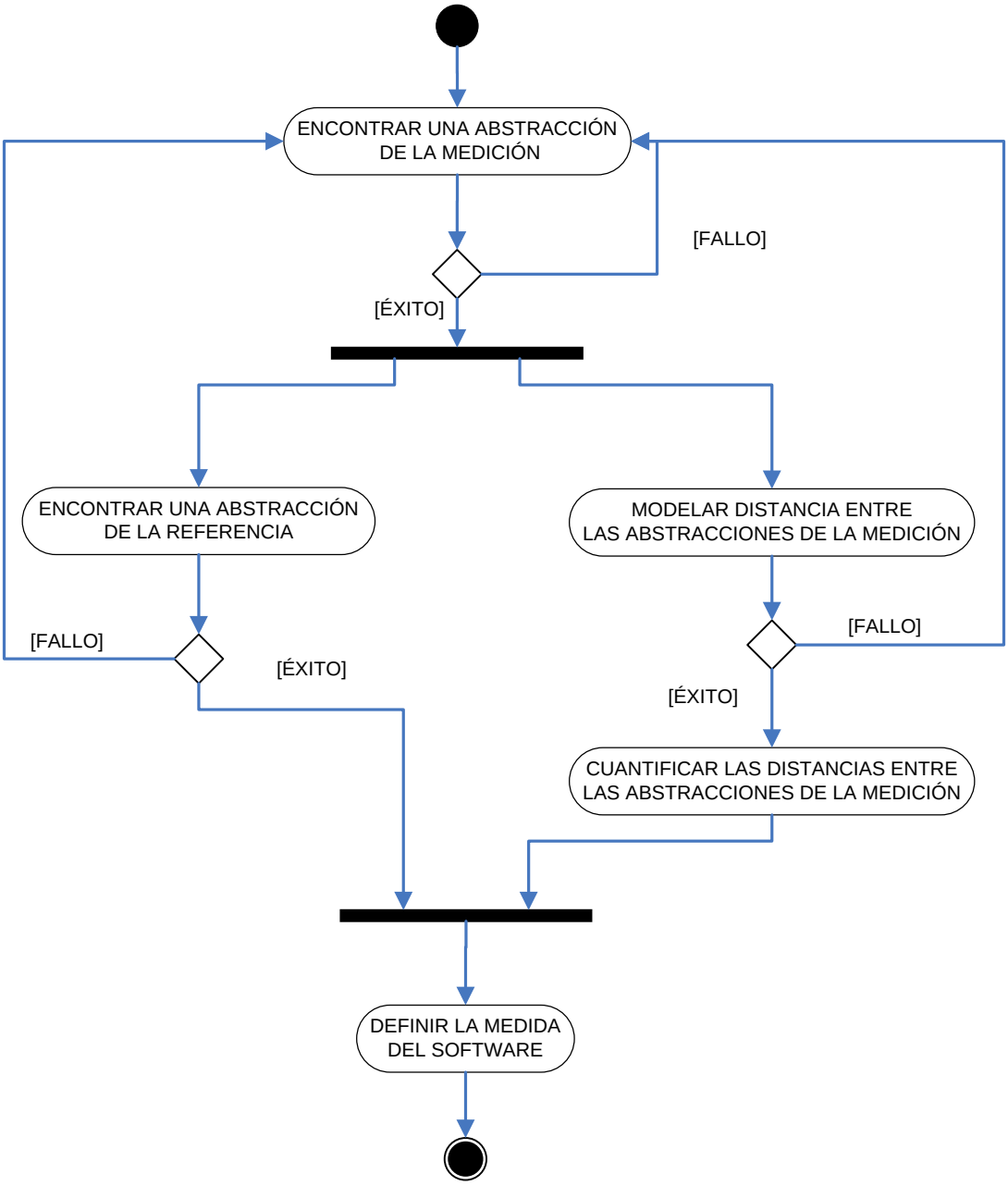

Figura 12. Modelo de proceso para la medición software basada en las distancias

5.1. Encontrar una abstracción de la medición

Del conjunto de entidades del software P que se caracterizan mediante el atributo *attr*, se selecciona un conjunto de entidades software M que se pueden usar como abstracciones de la medición para enfatizar *attr*, y definir un mapeo *abs*: P → M

Un buen modelo de una entidad software $p \in P$ es la que refleja la extensión del *attr* que caracteriza p. Esto significa que si p se caracterizara de forma diferente por *attr* de lo que actualmente lo hace, entonces p se mapearía en otro elemento de M distinto a su abstracción de medición actualmente utilizada $abs(p) \in M$. De ahí que el supuesto técnico subyacente a este paso sea que una abstracción de la medición adecuada para *attr* se puede hallar y que los mapeos de los elementos de P dentro de sus modelos en M se pueden describir de una forma no ambigua.

De acuerdo al Modelo de Usabilidad propuesto, la fórmula CDD (Completitud De la Documentación) calcula el porcentaje de primitivas conceptuales que llevan asociado un mensaje de ayuda. Se centra en las siguientes primitivas conceptuales: Atributos, Servicios, Argumentos, Patrón de Introducción, Selección Definida, Unidad de Interacción de Instancia, Unidad de Interacción de Población, Unidad de Interacción de Servicio, Maestro Detalle, Acciones, Navegaciones, Criterio de Ordenación, Filtro. A continuación se describe la notación utilizada en la definición de la función de abstracción de medición para la fórmula CDD:

- Sea UMCOO el Universo de los Modelos Conceptuales de OO-Method.
- Sea UPCA el Universo de las Primitivas Conceptuales con mensajes de Ayuda utilizadas en la fórmula CDD (solo las primitivas anteriormente citadas).
- Sea $\wp$(UPCA) el conjunto de conjuntos de UPCA.
- La operación de proyección PCA retorna para un MCOO (Modelo Conceptual de OO-Method) que pertenece a un UMCOO, el conjunto de primitivas conceptuales que tengan un mensaje de ayuda definido, el cual es un elemento de $\wp$(UPCA).

La abstracción de medición es definida por:

abs_{CDD}:UMCOO → $\wp$(UPCA) : MCOO → PCA(MCOO)

5.2. Modelar distancias entre abstracciones de medición

Se define un conjunto T_e de tipos de transformación elementales para M de forma que sea constructivamente completo e inversa constructivamente completo.

En este paso se modelan las distancias entre los elementos de M. Para este fin se utiliza el concepto de *transformaciones elementales*. Una transformación elemental de un elemento M convierte este elemento en otro elemento de M que es una modificación del primero. Esta modificación debe ser una modificación atómica, es decir, no se puede dividir en varios cambios elementales. Intuitivamente, el número de transformaciones elementales necesarias para transformar un elemento de M en otro elemento, representa la distancia entre ambos elementos.

Es un requisito que el conjunto T_e de tipos de transformaciones elementales para M sea constructivamente completo. Es decir, que todos los elementos de M se puedan alcanzar aplicando una secuencia finita de transformaciones elementales partiendo de un elemento de M inicial. Además, el conjunto T_e debe ser inversa constructivamente completo, es decir, el elemento inicial de M es alcanzable desde cualquier otro elemento de M aplicando una secuencia finita de transformaciones elementales. Mediante las propiedades de constructivamente completo y de inversa constructivamente completo, se garantiza que todo elemento de M se pueda convertir en cualquier otro elemento de M, directamente o mediante el elemento inicial.

Aplicando esta definición a nuestra fórmula de ejemplo, el objetivo sería modelar las distancias entre los elementos del conjunto de $\wp$(UPCA). En este paso se definen una serie de funciones homogéneas (T_e) sobre $\wp$(UPCA) de forma que para un porcentaje de primitivas con mensajes de ayuda definidos, se pueda alcanzar otro distinto. Para transformar un porcentaje de CDD en otro distinto solo habría que añadir o eliminar mensajes de ayuda a las primitivas conceptúales que forman parte de la fórmula CDD. Por lo tanto solamente harían falta dos funciones homogéneas: una que añada mensajes de ayuda a las primitivas y otras que elimine los mensajes de ayuda.

En base a estas dos funciones homogéneas una estructura de proximidad aditiva segmentada puede ser construida para definir una noción de distancia entre conjuntos de primitivas con mensajes de ayuda asociados.

Sea $\wp$(UPCA) un conjunto de Universo de Primitivas Conceptuales y T_e un conjunto de funciones homogéneas sobre $\wp$(UPCA). Por otro lado, s es un conjunto que pertenece a $\wp$(UPCA) y que puede ser transformado en otro por medio de las funciones homogéneas que perteneces a T_e.

$T_{e\text{-}CDD}$= {$t_{0\text{-}CDD}$, $t_{1\text{-}CDD}$} donde $t_{0\text{-}CDD}$ y $t_{1\text{-}CDD}$ son definidas :

$T_{0\text{-}CDD}$= $\wp$(UPCA) → $\wp$(UPCA) : s → s $\cup$ {e} con e $\in$ UPCA

$T_{1\text{-}CDD}$= $\wp$(UPCA) → $\wp$(UPCA) : s → s - {e} con e $\in$ UPCA

5.3. Cuantificar las distancias entre abstracciones de medición

Usando el procedimiento que se detalla a continuación, se define una métrica δ : M x M → $\Re$ tal que (M, δ) es un espacio de métrica.

En este paso se cuantifican las distancias en M que se han definido usando T_e. Esto se consigue a través de un espacio de métricas. Para ello se utilizan los siguientes procedimientos.

1. Sea c un número real positivo. Se define la función φ de la siguiente forma:

   φ : T → $\Re$: $T_{m,m'}$ → k.c donde k es la longitud de $T_{m,m'}$

2. Se define la métrica δ como:

   δ : M x M → $\Re$: (m,m') → φ ($T_{m,m'}$), donde $T_{m,m'} \in S T_{m,m'}$

La función φ devuelve para una secuencia de transformaciones elementales el producto de la longitud de esa secuencia (k), y un número real positivo (c). La función δ devuelve para cada par de elementos (m,m') de M, el valor que φ devuelve para la secuencia más corta de transformaciones elementales de m a m'. Por lo tanto, la distancia entre m y m' se mide contando las transformaciones elementales más cortas para pasar de m a m', y multiplicando este número por una constante positiva.

Aplicando estos procedimientos a nuestro ejemplo con la fórmula CDD, se debería definir el espacio de métricas ($\wp$(UPCA), δ) de la siguiente forma:

1. Sea c un número real positivo, la función φ_{CDD} sería:

   φ_{CDD}: T → $\Re$: $T_{s,s'}$ → k.c donde k es la longitud de $T_{s,s'}$

2. La métrica δ_{CDD} se define:

   δ_{CDD}: $\wp$(UPCA) x $\wp$(UPCA) → $\Re$: (s, s') → φ ($T_{s,s'}$), donde $T_{s,s'} \in S T_{s,s'}$

La definición de $\wp$(UPCA) x $\wp$(UPCA) → $\Re$ se puede formular de otra manera distinta. Si suponemos que tenemos que transformar s $\in$ $\wp$(UPCA) en s' $\in$ $\wp$(UPCA), la longitud más corta de transformaciones elementales de s a s' es igual a la cardinalidad de las diferencias simétricas entre s y s'. Por lo tanto la definición de δ_{CDD} se puede cambiar a:

δ_{CDD}: $\wp$(UPCA) x $\wp$(UPCA) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)

5.4. Encontrar una abstracción de referencia

Se define una función *ref*: P → M tal que para todo p ∈ P, *ref*(p)=*abs*(p) si y solo si p tiene el valor teóricamente mas pequeño de *attr*.

En este paso se debe pensar qué *abs*(p) existiría en el caso que p se caracterice por el valor teóricamente más bajo. Esta representación imaginaria de p se llama *abstracción de referencia* de p para *attr*. Se puede afirmar que cuanto más cerca está *abs(*p) a su abstracción de referencia, más bajo es el valor del atributo. Análogamente, cuanto más lejos esté *abs*(p) de la abstracción de referencia, más alto es el valor del atributo. Por lo tanto, la distancia conceptual entre la verdadera abstracción de medición y la abstracción de referencia, se utiliza como un modelo del atributo de interés.

Formalmente, la abstracción de referencia se obtiene a partir de la función *ref*: P → M. Como *ref* es una función, hay exactamente una abstracción de referencia asociada con la entidad p ∈ P para el atributo de interés *attr*. En las métricas definidas para sistemas orientados a objetos, es común que *ref* mapee todos los elementos de P en la misma abstracción de referencia, que es normalmente el valor inicial de M.

Aplicando este procedimiento a nuestro caso de ejemplo, CDD, habría que identificar la abstracción de medición para los Modelos Conceptuales de OO-Method con el tamaño funcional teóricamente más bajo. Como la abstracción de medición viene definida por el conjunto de primitivas conceptuales con mensajes de ayuda asociados, el valor teórico más bajo sería el conjunto sin ninguna primitiva conceptual con mensaje de ayuda.

ref_{CDD}: UMCOO → $\wp$(UPCA): MCOO → ϕ

5.5. Definir la medida del software

Se define una función μ : P → $\Re$ tal que para todo p ∈ P, μ(p)= δ(*abs*(p), *ref*(p))

Los resultados de las actividades previas se combinan para producir una definición formal del atributo de interés *attr* y su medida. Formalmente, *attr* de p ∈ P es la distancia entre *abs*(p) y *ref*(p), que se modela por la secuencia más corta de transformaciones elementales para pasar de *abs*(p) a *ref*(p). (Es decir se modela por $T_{abs(p),ref(p)} \in ST_{abs(p),ref(p)}$). El valor para esta distancia es δ(*abs*(p), *ref*(p)). Por lo tanto, si μ : P → $\Re$ se define como μ(p)= δ(*abs*(p), *ref*(p)), entonces μ se puede utilizar como una medida para *attr*.

Al aplicar el marco de DISTANCE a nuestro caso de ejemplo, el tamaño funcional del modelo conceptual de OO-Method: MCOO ∈ UMCOO es definido de la siguiente forma:

μ_{CDD}= δ(*abs*$_{CDD}$(MCOO), *ref*$_{CDD}$(MCOO))

μ_{CDD}=| *abs*$_{CDD}$(MCOO) - *ref*$_{CDD}$(MCOO)| + | *ref*$_{CDD}$(MCOO) - *abs*$_{CDD}$(MCOO) |

μ_{CDD}= | PCA(MCOO) - ϕ | + | ϕ - PCA(MCOO)|

μ_{CDD}= PCA(MCOO)

Por lo tanto se concluye que la fórmula CDD está correctamente definida como el número de primitivas conceptuales con mensajes de ayuda asociados identificados en todo el modelado

conceptual de OO-Method. Esta conclusión se fundamenta en el hecho de que la fórmula se puede representar mediante el método DISTANCE. En la Tabla 5 se muestra un resumen con todos los pasos seguidos en la validación de la fórmula.

Nombre de la fórmula	CDD (Completitud De Documentación)	
Aspecto a medir (*attr)*	Porcentaje de primitivas conceptuales con mensaje de ayuda definido	
Entidad de Software (p∈ P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CDD}$:UMCOO → $\wp$(UPCA) : MCOO → PCA(MCOO)	UPCA es el Universo de las Primitivas Conceptuales con mensajes de Ayuda. PCA es la operación que devuelve el conjunto de primitivas conceptuales que tengan un mensaje de ayuda definido.
T_e	$T_{0\text{-}CDD}$= $\wp$(UPCA) → $\wp$(UPCA) : s → s ∪ {e} con e ∈ UPCA $T_{1\text{-}CDD}$= $\wp$(UPCA) → $\wp$(UPCA) : s → s - {e} con e ∈ UPCA	Las funciones homogéneas (T_e) agregan o eliminan primitivas conceptuales con mensajes de ayuda.
δ	δ_{CDD}: $\wp$(UPCA) x $\wp$(UPCA) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de primitivas con mensaje de ayuda a otro.
ref	ref_{CDD}: UMCOO → $\wp$(UPCA): MCOO → ϕ	Sería el conjunto formado por todas las primitivas sin mensaje de ayuda.
μ	μ_{CDD}= δ (abs_{CDD}(MCOO),ref_{CDD}(MCOO)) μ_{CDD}= |PCA(MCOO)|	La medida sería el número de primitivas conceptuales con mensajes de ayuda que existen en el conjunto PCA(MCOO). A partir de este número se podría obtener el porcentaje de primitivas con mensajes de ayuda de entre todas las primitivas existentes.

Tabla 5. Definición basada en DISTANCE de la fórmula CDD

A continuación se muestra una tabla por cada una de las fórmulas utilizadas para medir la usabilidad de los atributos del Modelo de Usabilidad propuesto. Con estas tablas se demuestra que todas las fórmulas definidas son válidas.

Nombre de la fórmula	DA1 (Determinación de la acción 1)	
Aspecto a medir (*attr)*	Porcentaje de Servicios, Filtros y Criterios de Ordenación con alias definido	
Entidad de Software (p∈ P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{DA1}$:UMCOO → $\wp$(UPCL) : MCOO → PCL(MCOO)	UPCL es el Universo de las Primitivas Conceptuales con alias definido. PCL es la operación que devuelve el conjunto de primitivas conceptuales que tengan un alias definido.
T_e	$T_{0\text{-}DA1}$= $\wp$(UPCL) → $\wp$(UPCL) : s → s ∪ {e} con e ∈ UPCL	Las funciones homogéneas (T_e) agregan o eliminan primitivas conceptuales con alias definido.

	$T_{1\text{-}DA1}=\wp(UPCL) \rightarrow \wp(UPCL) : s \rightarrow s - \{e\}$ con $e \in UPCL$	
δ	$\delta_{DA1}: \wp(UPCL) \times \wp(UPCL) \rightarrow \Re : (s, s') \rightarrow c (\lvert s - s' \rvert + \lvert s' - s \rvert)$	Número de transformaciones elementales necesarias para pasar de un número de primitivas con alias definido a otro.
ref	ref_{DA1}: UMCOO $\rightarrow \wp$(UPCL): MCOO $\rightarrow \phi$	Sería el conjunto formado por todas las primitivas sin ningún alias definido.
μ	$\mu_{DA1}=\delta(abs_{DA1}(MCOO), ref_{DA1}(MCOO))$ $\mu_{DA1}= \lvert PCL(MCOO) \rvert$	La medida sería el número de primitivas conceptuales con alias definido que existen en el conjunto PCL(MCOO). A partir de esta medida se puede obtener el porcentaje de primitivas conceptuales con alias definido.

Tabla 6. Definición basada en DISTANCE de la fórmula DA1

Nombre de la fórmula	DA2 (Determinación de la acción 2)	
Aspecto a medir (*attr)*	Porcentaje de argumentos objeto valuados con Información Suplementaria definida	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{DA2}:UMCOO $\rightarrow \wp$ (UAOVIS): MCOO $\rightarrow$ AOVIS (MCOO)	UAOVIS es el Universo de Argumentos Objeto Valuados con Información Suplementaria asociada. AOVIS es la operación que devuelve el conjunto de argumentos objeto valuados que tengan una Información Suplementaria asociada.
T_e	$T_{0\text{-}DA2}=\wp(UAOVIS) \rightarrow \wp(UAOVIS) : s \rightarrow s \cup \{e\}$ con $e \in UAOVIS$ $T_{1\text{-}DA2}=\wp(UAOVIS) \rightarrow \wp(UAOVIS): s \rightarrow s - \{e\}$ con $e \in UAOVIS$	Las funciones homogéneas (T_e) agregan o eliminan argumentos objeto valuados con Información Suplementaria.
δ	$\delta_{DA2}: \wp(UAOVIS) \times \wp(UAOVIS) \rightarrow \Re : (s, s') \rightarrow c (\lvert s - s' \rvert + \lvert s' - s \rvert)$	Número de transformaciones elementales necesarias para pasar de un número de argumentos objeto valuados con Información Suplementaria a otro.
ref	Ref_{DA2}: UMCOO $\rightarrow \wp$ (UAOVIS): MCOO $\rightarrow \phi$	Sería el conjunto formado por todos los argumentos objeto valuados sin Información Suplementaria.
μ	$\mu_{DA2}=\delta(abs_{DA2}(MCOO), ref_{DA2}(MCOO))$ $\mu_{DA2}= \lvert AOVIS (MCOO) \rvert$	La medida sería el número argumentos objeto valuados con Información Suplementaria que existen en AOVIS (MCOO). A partir de esta medida se puede obtener el porcentaje de argumentos objeto valuados con Información Suplementaria.

Tabla 7. Definición basada en DISTANCE de la fórmula DA2

Nombre de la fórmula	APD (Agrupación Por Disposición) y DI1 (Densidad de la Información1)	
Aspecto a medir (*attr)*	Media de argumentos por Grupo de Argumentos definido	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{APD}:UMCOO $\rightarrow \wp$ (UA):	UA es el Universo de Argumentos.

	MCOO → A (MCOO)	A es la operación que devuelve el conjunto de argumentos.
T_e	$T_{0\text{- APD}} = \wp(UA) \rightarrow \wp(UA)$: s → s ∪ {e} con e ∈ UA $T_{1\text{- APD}} = \wp(UA) \rightarrow \wp(UA)$: s → s - {e} con e ∈ UA	Las funciones homogéneas (T_e) agregan o eliminan argumentos al Modelo Conceptual.
δ	δ_{APD}: $\wp(UA) \times \wp(UA)$ → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de argumentos a otro.
ref	Ref_{APD}: UMCOO → $\wp$ (UA): MCOO → ϕ	Sería el conjunto formado por primitivas conceptuales sin ningún argumento.
μ	$\mu_{APD} = \delta(abs_{APD}(MCOO), ref_{APD}(MCOO))$ μ_{APD} = |A (MCOO)|	La medida sería el número argumentos que existen en el conjunto A(MCOO). A partir de esta media se puede obtener la media de argumentos por Conjunto de Agrupación.

Tabla 8. Definición basada en DISTANCE de la fórmula APD y DI1

Nombre de la fórmula	DI2 (Densidad de la Información2)	
Aspecto a medir (*attr)*	Media de Elementos de Acción por UIP e UII.	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{DI2}:UMCOO → $\wp$ (UEA) : MCOO → EA(MCOO)	UEA es el Universo de los Elementos de Acción. EA es la operación que devuelve el conjunto de Elementos de Acción definidos en el Modelo Conceptual.
T_e	$T_{0\text{- DI2}} = \wp(UEA) \rightarrow \wp(UEA)$: s → s ∪ {e} con e ∈ UEA $T_{1\text{- DI2}} = \wp(UEA) \rightarrow \wp(UEA)$: s → s - {e} con e ∈ UEA	Las funciones homogéneas (T_e) agregan o eliminan Elementos de Acción.
δ	δ_{DI2}: $\wp(UEA) \times \wp(UEA)$ → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de Elementos de Acción a otro.
ref	ref_{DI2}: UMCOO → $\wp$ (UEA): MCOO → ϕ	Sería el conjunto formado por todas las primitivas sin ningún Elemento de Acción definido.
μ	$\mu_{DI2} = \delta(abs_{DI2}(MCOO), ref_{DI2}(MCOO))$ μ_{DI2}= |EA(MCOO)|	La medida sería el número de Elementos de Acción que existen en el conjunto EA(MCOO). A partir de esta medida se puede obtener la media de Elementos de Acción por UIP e UII.

Tabla 9. Definición basada en DISTANCE de la fórmula DI2

Nombre de la fórmula	DI3 (Densidad de la Información 3)	
Aspecto a medir (*attr)*	Media de atributos por Conjunto de Visualización en UIP E UII	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{DI3}:UMCOO → $\wp$ (UACV) : MCOO → ACV(MCOO)	UACV es el Universo de los Atributos del sistema. ACV es la operación que devuelve el conjunto de atributos que forman parte de los conjuntos de

		visualización definidos en el Modelo Conceptual.
T_e	$T_{0\text{-}DI3}= \wp(UACV) \rightarrow \wp(UACV) : s \rightarrow s \cup \{e\}$ con $e \in UACV$ $T_{1\text{-}DI3}= \wp(UACV) \rightarrow \wp(UACV) : s \rightarrow s - \{e\}$ con $e \in UACV$	Las funciones homogéneas (T_e) agregan o eliminan atributos a un Conjunto de Visualización.
δ	$\delta_{DI3}: \wp(UACV) \times \wp(UACV) \rightarrow \Re : (s, s') \rightarrow c(|s - s'| + |s' - s|)$	Número de transformaciones elementales necesarias para pasar de un número de atributos de un Conjunto de Visualización a otro.
ref	ref_{DI3}: UMCOO $\rightarrow \wp(UACV)$: MCOO $\rightarrow \phi$	Sería el conjunto formado por todos los Conjuntos de Visualización sin ningún atributo definido.
μ	$\mu_{DI3}= \delta(abs_{DI3}(MCOO), ref_{DI3}(MCOO))$ $\mu_{DI3}= \|ACV(MCOO)\|$	La medida sería el número de atributos del Conjunto de Visualización que existen en el conjunto ACV(MCOO). A partir de esta medida se puede obtener la media de atributos entre todos lo Conjuntos de Visualización del sistema.

Tabla 10. Definición basada en DISTANCE de la fórmula DI3

Nombre de la fórmula	DI4 (Densidad de la Información 4)	
Aspecto a medir (*attr)*	Media de Elementos de Navegación por UI e UII.	
Entidad de Software (p∈P)	MCOO $\in$ UMCOO	
Resultados de DISTANCE		
abs	abs_{DI4}:UMCOO $\rightarrow \wp(UEN)$: MCOO $\rightarrow$ EN(MCOO)	UEN es el Universo de los Elementos de Navegación. EN es la operación que devuelve el conjunto de Elementos de Navegación definidos en el Modelo Conceptual.
T_e	$T_{0\text{-}DI4}= \wp(UEN) \rightarrow \wp(UEN) : s \rightarrow s \cup \{e\}$ con $e \in UEN$ $T_{1\text{-}DI4}= \wp(UEN) \rightarrow \wp(UEN) : s \rightarrow s - \{e\}$ con $e \in UEN$	Las funciones homogéneas (T_e) agregan o eliminan Elementos de Navegación.
δ	$\delta_{DI4}: \wp(UEN) \times \wp(UEN) \rightarrow \Re : (s, s') \rightarrow c(|s - s'| + |s' - s|)$	Número de transformaciones elementales necesarias para pasar de un número de Elementos de Navegación a otro.
ref	ref_{DI4}: UMCOO $\rightarrow \wp(UEN)$: MCOO $\rightarrow \phi$	Sería el conjunto formado por todas las primitivas sin ningún Elemento de Nevegación definido.
μ	$\mu_{DI4}= \delta(abs_{DI4}(MCOO), ref_{DI4}(MCOO))$ $\mu_{DI4}= \|EN(MCOO)\|$	La medida sería el número de Elementos de Navegación que existen en el conjunto EN(MCOO). A partir de esta medida se puede obtener la media de Elementos de Navegación por UIP y UII.

Tabla 11. Definición basada en DISTANCE de la fórmula DI4

Nombre de la fórmula	DI5 (Densidad de la Información 5)	
Aspecto a medir (*attr)*	Promedio de UIP y UII con al menos un filtro definido.	
Entidad de Software (p∈P)	MCOO $\in$ UMCOO	
Resultados de DISTANCE		
abs	abs_{DI5}:UMCOO $\rightarrow \wp(UUIP)$:	UUIP es el Universo de las UIP UIPF es la operación que devuelve el conjunto de

	MCOO → UIPF(MCOO)	UIP con al menos un Filtro definido.
T_e	$T_{0\text{- DI5}}=\wp$ (UUIP) → $\wp$ (UUIP) : s → s $\cup$ {e} con e $\in$ UUIP $T_{1\text{- DI5}}=\wp$ (UUIP) → $\wp$ (UUIP) : s → s - {e} con e $\in$ UUIP	Las funciones homogéneas (T_e) agregan o eliminan Filtros a las UIP.
δ	$\delta_{\text{ DI5}}$: $\wp$ (UUIP) x $\wp$ (UUIP) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de una UIP con Filtros a otra con Filtros o viceversa.
ref	ref_{DI5}: UMCOO → $\wp$ (UUIP): MCOO → ϕ	Sería el conjunto formado por todas las primitivas UIP sin ningún Filtro definido.
μ	$\mu_{\text{ DI5}}=\delta$ (abs_{DI5}(MCOO),ref_{DI5} (MCOO)) $\mu_{\text{ DI5}}$= |UIPF(MCOO)|	La medida sería el número de UIP que existen en el conjunto UIPF(MCOO) (Esta función devuelve solo las UIP con al menos un Filtro). A partir de esta medida se puede obtener el porcentaje de UIP con Filtro contando cuantas UIP hay en todo el sistema.

Tabla 12. Definición basada en DISTANCE de la fórmula DI5

Nombre de la fórmula	DI6 (Densidad de la Información 6)	
Aspecto a medir (*attr)*	Media de Variables De Filtro por cada Filtro definido en una UIP.	
Entidad de Software (p$\in$P)	MCOO $\in$ UMCOO	
Resultados de DISTANCE		
abs	abs $_{\text{DI6}}$:UMCOO → $\wp$ (UVF) : MCOO → VF(MCOO)	UVF es el Universo de las Variables de Filtro. VF es la operación que devuelve el conjunto de las Variables de Filtro definido en el Modelo Conceptual.
T_e	$T_{0\text{- DI6}}=\wp$ (UVF) → $\wp$ (UVF) : s → s $\cup$ {e} con e $\in$ UVF $T_{1\text{- DI6}}=\wp$ (UVF) → $\wp$ (UVF) : s → s - {e} con e $\in$ UVF	Las funciones homogéneas (T_e) agregan o eliminan Variables de Filtro.
δ	$\delta_{\text{ DI6}}$: $\wp$ (UVF) x $\wp$ (UVF) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de Variables de Filtro a otro.
ref	ref_{DI6}: UMCOO → $\wp$ (UVF): MCOO → ϕ	Sería el conjunto formado por todas las primitivas sin ninguna Variable de Filtro definida.
μ	$\mu_{\text{ DI6}}=\delta$ (abs_{DI6}(MCOO),ref_{DI6} (MCOO)) $\mu_{\text{ DI6}}$= |VF(MCOO)|	La medida sería el número de Variables de Filtro que existen en el conjunto VF(MCOO). A partir de esta medida se puede obtener la media de Variables de Filtro por cada Filtro definido en las UIPs.

Tabla 13. Definición basada en DISTANCE de la fórmula DI6

Nombre de la fórmula	DI7 (Densidad de la Información 7)
Aspecto a medir (*attr)*	Porcentaje de UIP con al menos un Criterio de Ordenación definido.
Entidad de Software (p$\in$P)	MCOO $\in$ UMCOO
Resultados de DISTANCE	

abs	abs $_{DI7}$:UMCOO → $\wp$(UUIP) : MCOO → UIPCO(MCOO)	UUIP es el Universo de las UIP UIPCO es la operación que devuelve el conjunto de UIP con al menos un Criterio de Ordenación definido.
T_e	$T_{0\text{- }DI7}$= $\wp$(UUIP) → $\wp$(UUIP) : s → s ∪ {e} con e ∈ UUIP $T_{1\text{- }DI7}$= $\wp$(UUIP) → $\wp$(UUIP) : s → s - {e} con e ∈ UUIP	Las funciones homogéneas (T_e) agregan o eliminan Criterios de Ordenación a las UIP.
δ	δ_{DI7}: $\wp$(UUIP) x $\wp$(UUIP) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de una UIP con Criterios de Ordenación a otra que no los tenga y viceversa.
ref	*ref* $_{DI7}$: UMCOO → $\wp$(UUIP): MCOO → ϕ	Sería el conjunto formado por todas las primitivas UIP sin ningún Criterio de Ordenación definido.
μ	μ_{DI7}= δ (*abs*$_{DI7}$(MCOO),*ref*$_{DI7}$ (MCOO)) μ_{DI7}= |UIPCO(MCOO)|	La medida sería el número de UIP que existen en el conjunto UIPCO(MCOO) (Esta función devuelve solo las UIP con al menos un Criterio de Ordenación). A partir de esta medida se puede obtener el porcentaje de UIP con Criterio de Ordenación contando cuantas UIP hay en todo el sistema.

Tabla 14. Definición basada en DISTANCE de la fórmula DI7

Nombre de la fórmula	SDE (Significación Del Etiquetado)	
Aspecto a medir (*attr)*	Porcentaje de atributos con alias definido	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{SDE}$:UMCOO → $\wp$(UAAD) : MCOO → AAD(MCOO)	UAAD es el Universo de los Atributos con un Alias Definido. AAD es la operación que devuelve el conjunto de los Atributos con un Alias Definido.
T_e	$T_{0\text{- }SDE}$ = $\wp$(UAAD) → $\wp$(UAAD) : s → s ∪ {e} con e ∈ UAAD $T_{1\text{- }SDE}$ = $\wp$(UAAD) → $\wp$(UAAD) : s → s - {e} con e ∈ UAAD	Las funciones homogéneas (T_e) agregan o eliminan alias a los atributos.
δ	δ_{SDE}: $\wp$(UAAD) x $\wp$(UAAD) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de atributos con alias a otro.
ref	*ref* $_{SDE}$: UMCOO → $\wp$(UAAD): MCOO → ϕ	Sería el conjunto formado por todos los atributos sin ningún alias.
μ	μ_{SDE} = δ (*abs* $_{SDE}$ (MCOO),*ref* $_{SDE}$ (MCOO)) μ_{SDE} = |AAD(MCOO)|	La medida sería el número de atributos con alias que existen en el conjunto AAD(MCOO). A partir de esta medida se puede obtener el porcentaje de atributos con alias en todo el sistema contando los atributos que existen en total.

Tabla 15. Definición basada en DISTANCE de la fórmula SDE

Nombre de la fórmula	CVI (Completitud de Valores Iniciales)
Aspecto a medir (*attr)*	Porcentaje de argumentos con valor inicial

Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CVI}$:UMCOO → $\wp$ (UAVI) : MCOO → AVI(MCOO)	UAVI es el Universo de los Argumentos con Valor Inicial definido. AVI es la operación que devuelve el conjunto de Argumentos con Valor Inicia definido.
T_e	$T_{0\text{- CVI}} = \wp$ (UAVI) → $\wp$ (UAVI) : s → s ∪ {e} con e ∈ UAVI $T_{1\text{- CVI}} = \wp$ (UAVI) → $\wp$ (UAVI) : s → s - {e} con e ∈ UAVI	Las funciones homogéneas (T_e) agregan o eliminan valores iniciales a los argumentos.
δ	δ_{CVI}: $\wp$ (UAVI) x $\wp$ (UAVI) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de argumentos con valor inicial a otro.
ref	*ref* $_{CVI}$: UMCOO → $\wp$ (UAVI): MCOO → ϕ	Sería el conjunto formado por todos los argumentos sin ningún valor inicial definido.
μ	$\mu_{CVI} = \delta$ (*abs* $_{CVI}$ (MCOO),*ref* $_{CVI}$ (MCOO)) μ_{CVI} = |AVI(MCOO)|	La medida sería el número de argumentos con valor inicial definido que existen en el conjunto AVI(MCOO). A partir de esta medida se puede obtener el porcentaje de argumentos con valor inicial definido contando los argumentos existentes en todo el sistema.

Tabla 16. Definición basada en DISTANCE de la fórmula CVI

Nombre de la fórmula	CDM (Calidad De los Mensajes)	
Aspecto a medir (*attr)*	Media de la longitud de los mensajes de error mostrados al usuario	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CDM}$:UMCOO → $\wp$ (UME) : MCOO → ME(UME)	UME es el Universo de las longitudes de los Mensajes de Error. ME es la operación que devuelve el conjunto de las longitudes de los mensajes de error.
T_e	$T_{0\text{- CDM}} = \wp$ (UME) → $\wp$ (UME) : s → s ∪ {e} con e ∈ UME $T_{1\text{- CDM}} = \wp$ (UME) → $\wp$ (UME) : s → s - {e} con e ∈ UME	Las funciones homogéneas (T_e) agregan o eliminan caracteres a un mensaje de error.
δ	δ_{CDM}: $\wp$ (UME) x $\wp$ (UME) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de una longitud del mensaje de error a otra.
ref	*ref* $_{CDM}$: UMCOO → $\wp$ (UME): MCOO → ϕ	Sería el conjunto formado por todos los mensajes de error sin caracteres.
μ	$\mu_{CDM} = \delta$ (*abs* $_{CDM}$ (MCOO),*ref* $_{CDM}$ (MCOO)) μ_{CDM} = |ME(MCOO)|	La medida sería la suma de las longitudes de los mensajes de error que existen en el conjunto ME(MCOO). A partir de esta medida se puede obtener la media de las longitudes de los mensajes de error contando todos los mensajes de error definidos en el sistema.

Tabla 17. Definición basada en DISTANCE de la fórmula CDM

Nombre de la fórmula	NV (Navegaciones Definidas)	
Aspecto a medir (*attr)*	Media de navegaciones definidas entre todas las UIP y UII	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{NV}:UMCOO → $\wp$(UN) : MCOO → N(MCOO)	UN es el Universo de las Navegaciones definidas en las UIP y UII. N es la operación que devuelve el conjunto de Navegaciones definidas.
T_e	$T_{0\text{-}NV} = \wp(UN) \rightarrow \wp(UN) : s \rightarrow s \cup \{e\}$ con e ∈ UN $T_{1\text{-}NV} = \wp(UN) \rightarrow \wp(UN) : s \rightarrow s - \{e\}$ con e ∈ UN	Las funciones homogéneas (T_e) agregan o eliminan navegaciones en las UIP y UII.
δ	δ_{NV}: $\wp(UN) \times \wp(UN) \rightarrow$ $\Re : (s, s') \rightarrow c(\lvert s - s'\rvert + \lvert s' - s\rvert)$	Número de transformaciones elementales necesarias para pasar de un número de navegaciones a otro.
ref	ref_{NV}: UMCOO → $\wp$(UN): MCOO → ϕ	Sería el conjunto formado por todas las UIP y UII sin navegaciones definidas.
μ	$\mu_{NV} = \delta(abs_{NV}(MCOO), ref_{NV}(MCOO))$ $\mu_{NV} = \lvert N(MCOO)\rvert$	La medida sería el número de navegaciones que existen en el conjunto N(MCOO). A partir de esta medida se puede obtener la media de las navegaciones por UI contando el número de UIPs y UIIs definidas en el sistema.

Tabla 18. Definición basada en DISTANCE de la fórmula NV

Nombre de la fórmula	CO1 (Consistencia de Orden 1)	
Aspecto a medir (*attr)*	Porcentaje de clases con el mismo orden en sus argumentos de los servicios de crear y modificar	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{CO1}:UMCOO → $\wp$(UCMOA) : MCOO → CMOA(MCOO)	UCMOA es el Universo de las Clases con el Mismo Orden de sus Argumentos en los servcios de crear y modificar. CMOA es la operación que devuelve el conjunto Clases con el mismo Orden de Argumentos en los servicios crear y modificar.
T_e	$T_{0\text{-}CO1} = \wp(UCMOA) \rightarrow \wp(UCMOA) : s \rightarrow s \cup \{e\}$ con e ∈ UCMOA $T_{1\text{-}CO1} = \wp(UCMOA) \rightarrow \wp(UCMOA) : s \rightarrow s - \{e\}$ con e ∈ UCMOA	Las funciones homogéneas (T_e) agregan o eliminan clases con el mismo orden en sus argumentos de los servicios de creación y modificación.
δ	δ_{CO1}: $\wp(UCMOA) \times \wp(UCMOA) \rightarrow$ $\Re : (s, s') \rightarrow c(\lvert s - s'\rvert + \lvert s' - s\rvert)$	Número de transformaciones elementales necesarias para pasar de un número de clases con el mismo orden en los argumentos de los servicios de crear y modificar a otro número.
ref	ref_{CO1}: UMCOO → $\wp$(UCMOA): MCOO → ϕ	Sería el conjunto formado por todas las clases cuyos argumentos de los servicios crear y modificar están todos en distinto orden.
μ	$\mu_{CO1} = \delta(abs_{CO1}(MCOO), ref_{CO1}(MCOO))$	La medida sería el número de clases con el mismo orden de sus argumentos en los servicios crear y modificar que existen en el conjunto CMO(MCOO). A partir de esta medida se puede

	μ_{CO1} = |CMOA(MCOO)|	obtener el porcentaje de clases con el mismo orden de los argumentos en sus servicios de creación y modificación contando todas las clases definidas en el sistema que dispongan de servicios de creación y modificación.

Tabla 19. Definición basada en DISTANCE de la fórmula CO1

Nombre de la fórmula	CO2 (Consistencia de Orden 2)	
Aspecto a medir (*attr)*	Porcentaje de clases con el mismo orden en sus servicios de creación, modificación y destrucción	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CO2}$:UMCOO → $\wp$(UCMO) : MCOO → CMO(MCOO)	UCMO es el Universo de las Clases con el Mismo Orden de aparición en sus servicios de creación, modificación y destruir. CMO es la operación que devuelve el conjunto Clases con el mismo Orden de sus servicios crear , modificar y destruir.
T_e	$T_{0\text{-}CO2} = \wp$(UCMO) → $\wp$(UCMO) : s → s ∪ {e} con e ∈ UCMO $T_{1\text{-}CO2} = \wp$(UCMO) → $\wp$(UCMO) : s → s - {e} con e ∈ UCMO	Las funciones homogéneas (T_e) agregan o eliminan clases con el mismo orden en sus servicios de creación, modificación y destrucción.
δ	δ_{CO2}: $\wp$(UCMO) x $\wp$(UCMO) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de clases con el mismo orden en sus servicios de crear, modificar y destruir a otro.
ref	*ref* $_{CO2}$: UMCOO → $\wp$(UCMO): MCOO → ϕ	Sería el conjunto formado por todas las clases que difieren en el orden de aparición de sus servicios de crear, modificar y destruir.
μ	$\mu_{CO2} = \delta$ (*abs* $_{CO2}$ (MCOO),*ref* $_{CO2}$ (MCOO)) μ_{CO2} = |CMO(MCOO)|	La medida sería el número de clases con el mismo orden de sus servicios crear, modificar y destruir que existen en el conjunto CMO(MCOO). A partir de esta medida se puede obtener el porcentaje de clases con el mismo orden en sus servicios de creación y destrucción contando todas las clases definidas en el sistema que dispongan de los tres servicios.

Tabla 20. Definición basada en DISTANCE de la fórmula CO2

Nombre de la fórmula	CO3 (Consistencia de Orden 3)	
Aspecto a medir (*attr)*	Porcentaje de clases cuyos atributos se muestran de forma ordenada al usuario (mostrando primero los atributos utilizados como identificadores)	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CO3}$:UMCOO → $\wp$(UCMOA) : MCOO → CMOA(MCOO)	UCMOA es el Universo de las Clases con el Mismo Orden de aparición de los atributos. CMOA es la operación que devuelve el conjunto de Clases con el mismo Orden de sus Atributos.
T_e	$T_{0\text{-}CO3} = \wp$(UCMOA)→ $\wp$(UCMOA) : s → s ∪ {e} con e ∈ UCMOA	Las funciones homogéneas (T_e) agregan o eliminan clases con el mismo orden en sus atributos.

	$T_{1\text{-}CO3} = \wp(UCMOA) \rightarrow \wp(UCMOA) : s \rightarrow s - \{e\}$ con $e \in UCMOA$	
δ	$\delta_{CO3}: \wp(UCMOA) \times \wp(UCMOA) \rightarrow \Re : (s, s') \rightarrow c\ (|s - s'| + |s' - s|)$	Número de transformaciones elementales necesarias para pasar de un número de clases con el mismo orden en sus atributos a otro.
ref	$ref_{CO3}: UMCOO \rightarrow \wp(UCMOA): MCOO \rightarrow \phi$	Sería el conjunto formado por todas las clases que no están ordenadas (no muestran primero sus atributos identificadores)
μ	$\mu_{CO3} = \delta(abs_{CO3}(MCOO), ref_{CO3}(MCOO))$ $\mu_{CO3} = \|CMOA(MCOO)\|$	La medida sería el número de clases cuyos atributos estén ordenados que existen en el conjunto CMOA(MCOO). A partir de esta medida se puede obtener el porcentaje de clases con el mismo orden en sus servicios de creación y destrucción contando todas las clases definidas en el sistema que dispongan de los tres servicios.

Tabla 21. Definición basada en DISTANCE de la fórmula CO3

Nombre de la fórmula	CE1 (Consistencia de Etiquetado 1)	
Aspecto a medir (*attr)*	Porcentaje de navegaciones hacia un destino ya referenciado cuyo alias sea el mismo que el de la referencia ya existente.	
Entidad de Software ($p \in P$)	$MCOO \in UMCOO$	
Resultados de DISTANCE		
abs	$abs_{CE1}: UMCOO \rightarrow \wp(UNMA) : MCOO \rightarrow NMA(MCOO)$	UNMA es el Universo de las Navegaciones con el Mismo Alias hacia una referencia ya existente. NMA es la operación que devuelve el conjunto de Navegaciones con el Mismo Alias hacia destinos ya existentes.
T_e	$T_{0\text{-}CE1} = \wp(UNMA) \rightarrow \wp(UNMA) : s \rightarrow s \cup \{e\}$ con $e \in UNMA$ $T_{1\text{-}CE1} = \wp(UNMA) \rightarrow \wp(UNMA) : s \rightarrow s - \{e\}$ con $e \in UNMA$	Las funciones homogéneas (T_e) agregan o eliminan alias ya existentes para las navegaciones hacia los mismos destinos.
δ	$\delta_{CE1}: \wp(UNMA) \times \wp(UNMA) \rightarrow \Re : (s, s') \rightarrow c\ (|s - s'| + |s' - s|)$	Número de transformaciones elementales necesarias para pasar de un número de navegaciones hacia el mismo destino con el mismo alias a otro.
ref	$ref_{CE1}: UMCOO \rightarrow \wp(UNMA): MCOO \rightarrow \phi$	Sería el conjunto formado por todas las navegaciones hacia el mismo destino cuyos alias fueran todos distintos entre sí.
μ	$\mu_{CE1} = \delta(abs_{CE1}(MCOO), ref_{CE1}(MCOO))$ $\mu_{CE1} = \|NMA(MCOO)\|$	La medida sería el número de navegaciones hacia destinos ya referenciados que tengan el mismo alias que existen en el conjunto NMA(MCOO). A partir de esta medida se puede obtener el porcentaje de navegaciones con el mismo alias hacia los mismos destinos de entre todas las navegaciones hacia destinos ya existentes del sistema.

Tabla 22. Definición basada en DISTANCE de la fórmula CE1

Nombre de la fórmula	CE2 (Consistencia de Etiquetado 2)
Aspecto a medir (*attr)*	Porcentaje de alias iguales de Elementos de Acción que referencia a un servicio que ya es accesible desde otra parte del programa.

Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CE2}$:UMCOO → $\wp$(UNMA) : MCOO → NMA(MCOO)	UAEAM es el Universo de los Alias de Elementos de Acción que referencian un Mismo servicio desde distintas partes de una misma aplicación. AEAM es la operación que devuelve el conjunto de Alias de Elementos de Acción que referencian un Mismo servicio en una misma aplicación.
T_e	$T_{0\text{- }CE2}$ = $\wp$(UNMA)→ $\wp$(UNMA) : s → s ∪ {e} con e ∈ UNMA $T_{1\text{- }CE2}$ = $\wp$(UNMA)→ $\wp$(UNMA) : s → s - {e} con e ∈ UNMA	Las funciones homogéneas (T_e) agregan o eliminan alias ya existentes para los Elementos de Acción que referencian los mismos servicios.
δ	δ_{CE2}: $\wp$(UNMA)x $\wp$(UNMA) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número Elementos de Acción que referencian al mismo servicio y tienen el mismo alias, a otro distinto.
ref	*ref* $_{CE2}$: UMCOO → $\wp$(UNMA): MCOO → ϕ	Sería el conjunto formado por todos los alias de Elementos de Acción que apuntan a servicios ya referenciados cuyos alias sean todos distintos entre sí.
μ	μ_{CE2} = δ(*abs* $_{CE2}$ (MCOO),*ref* $_{CE2}$ (MCOO)) μ_{CE2} = |NMA (MCOO)|	La medida sería el número de alias iguales de Elementos de Acción que apuntan a servicios ya referenciados en el mismo sistema que existen en el conjunto ASM(MCOO). A partir de esta medida se puede obtener el porcentaje de alias iguales que referencian al mismo servicio. Para ello hay que contar en todo el sistema el número total de Elementos de Acción que referencian un servicio desde varias partes de la aplicación.

Tabla 23. Definición basada en DISTANCE de la fórmula CE2

Nombre de la fórmula	CE3 (Consistencia de Etiquetado 3)	
Aspecto a medir (*attr)*	Porcentaje de alias iguales de Elementos del Conjunto de Visualización que referencian a un atributo que ya es visible desde otra parte del programa.	
Entidad de Software (p∈P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{CE3}$:UMCOO → $\wp$(UAECV) : MCOO → AECV (MCOO)	UAECV es el Universo de los Alias de Elementos del Conjunto de Visualización que referencian un mismo atributo desde distintas partes de una misma aplicación. AECV es la operación que devuelve el conjunto de Alias de Elementos del Conjunto de Visualización que referencian un mismo atributo en una misma aplicación.
T_e	$T_{0\text{- }CE3}$ = $\wp$(UAECV)→ $\wp$(UAECV) : s → s ∪ {e} con e ∈ UAECV $T_{1\text{- }CE3}$ = $\wp$(UAECV)→ $\wp$(UAECV) : s → s - {e} con e ∈ UAECV	Las funciones homogéneas (T_e) agregan o eliminan alias ya existentes para los Elementos del Conjunto de Visualización que referencian los mismos atributos.
δ	δ_{CE3}: $\wp$(UAECV)x $\wp$(UAECV) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de Elementos de Conjunto de Visualización que

		referencian al mismo atributo y tienen el mismo alias, a otro distinto.
ref	ref_{CE3}: UMCOO → $\wp$ (UAECV): MCOO → ϕ	Sería el conjunto formado por todos los alias de Elementos de Conjunto de Visualización que apuntan a atributos ya referenciados cuyos alias sean todos distintos entre sí.
μ	$\mu_{CE3} = \delta$ (abs_{CE3} (MCOO),ref_{CE3} (MCOO)) μ_{CE3} = |AECV (MCOO)|	La medida sería el número de alias iguales de Elementos de Conjunto de Visualización que apuntan a atributos ya referenciados en el mismo sistema que existen en el conjunto AECV(MCOO). A partir de esta medida se puede obtener el porcentaje de alias iguales que referencian al mismo atributo. Para ello hay que contar en todo el sistema el número total de Elementos de Conjunto de Visualización que referencian un mismo atributo desde varias partes de la aplicación.

Tabla 24. Definición basada en DISTANCE de la fórmula CE3

Nombre de la fórmula	CE4 (Consistencia de Etiquetado 4)	
Aspecto a medir (*attr)*	Porcentaje de alias iguales de Variables de Filtro que referencian a un atributo que ya es visible desde otra Variable de Filtro del programa.	
Entidad de Software (p∈ P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs_{CE4}:UMCOO → $\wp$(UAVF) : MCOO → AVF (MCOO)	UAVF es el Universo de los Alias de las Variables de Filtro que referencian un mismo atributo desde distintas partes de una misma aplicación. AVF es la operación que devuelve el conjunto de Alias de Variables de Filtro que referencian un mismo atributo en una misma aplicación.
T_e	$T_{0\text{-}CE4}$ = $\wp$ (UAVF)→ $\wp$ (UAVF) : s → s ∪ {e} con e ∈ UAVF $T_{1\text{-}CE4}$ = $\wp$ (UAVF)→ $\wp$ (UAVF) : s → s - {e} con e ∈ UAVF	Las funciones homogéneas (T_e) agregan o eliminan alias ya existentes para las Variables de Filtro que referencian los mismos atributos.
δ	δ_{CE4}: $\wp$ (UAVF)x $\wp$ (UAVF) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de Variables de Filtro referencian al mismo atributo y tienen el mismo alias, a otro distinto.
ref	ref_{CE4}: UMCOO → $\wp$ (UAVF): MCOO → ϕ	Sería el conjunto formado por todos los alias de Variables de Filtro que apuntan a atributos ya referenciados cuyos alias sean todos distintos entre sí.
μ	$\mu_{CE4} = \delta$ (abs_{CE4} (MCOO),ref_{CE4} (MCOO)) μ_{CE4} = |AVF (MCOO)|	La medida sería el número de alias iguales de Variables de Filtro que apuntan a atributos ya referenciados en el mismo sistema que existen en el conjunto AVF(MCOO). A partir de esta medida se puede obtener el porcentaje de alias iguales que referencian al mismo atributo. Para ello hay que contar en todo el sistema el número total de Variables de Filtro que referencian un mismo atributo desde varias partes de la aplicación.

Tabla 25. Definición basada en DISTANCE de la fórmula CE4

Nombre de la fórmula		PDE (Prevención De Errores)
Aspecto a medir (*attr)*		Porcentaje de uso del patrón Selección Definida en la visualización de atributos de tipo string cuya longitud es menor a 4 caracteres.
Entidad de Software (p∈P)		MCOO ∈ UMCOO
Resultados de DISTANCE		
abs	abs_{PDE}:UMCOO → $\wp$(UASD) : MCOO → ASD (MCOO)	UASD es el Universo de los Atributos de tipo string cuya longitud es menor de 4 caracteres y que utilizan el patrón de Selección Definida. ASD es la operación que devuelve el conjunto de Atributos de longitud menor de 4 de tipo string que utilizan el patrón de Selección Definida en su visualización.
T_e	$T_{0\text{- PDE}} = \wp(UASD) \rightarrow \wp(UASD) : s \rightarrow s \cup \{e\}$ con $e \in UASD$ $T_{1\text{- PDE}} = \wp(UASD) \rightarrow \wp(UASD) : s \rightarrow s - \{e\}$ con $e \in UASD$	Las funciones homogéneas (T_e) agregan o eliminan el patrón de Selección Definida a los atributos de tipo string cuya longitud es menor que 4.
δ	$\delta_{PDE}: \wp(UASD) \times \wp(UASD) \rightarrow \Re : (s, s') \rightarrow c(\lvert s - s'\rvert + \lvert s' - s\rvert)$	Número de transformaciones elementales necesarias para pasar de un número de atributos de tipo string y longitud menor que 4 que utilizan el patrón Selección Definida, a otro distinto.
ref	ref_{PDE}: UMCOO → $\wp$(UASD): MCOO → ϕ	Sería el conjunto formado por todos los atributos de tipos string y longitud menor que 4 que no utilizan el patrón de Selección Definida.
μ	$\mu_{PDE} = \delta(abs_{PDE}(MCOO), ref_{PDE}(MCOO))$ $\mu_{PDE} = \lvert ASD(MCOO)\rvert$	La medida sería el número de atributos de tipo string y de longitud menor que 4 que utilizan el patrón de Selección Definida que existen en el conjunto ASD(MCOO). A partir de esta medida se puede obtener el porcentaje de uso del patrón Selección Definida entre todos los atributos que sean de tipo string y longitud menor que 4. Para ello solo hay que contar el número de atributos de todo el sistema que cumplen ambas condiciones.

Tabla 26. Definición basada en DISTANCE de la fórmula PDE

6. MEJORAS EN EL MODELO DE USABILIDAD

El Modelo de Usabilidad presentado en la sección 1 fue elaborado utilizando criterios ergonómicos, heurísticos y guías de usabilidad. Sin embargo, este modelo todavía puede ser enriquecido con más atributos aun no incorporados. Para detectar nuevos atributos, este trabajo se ha centrado en el estudio de patrones de usabilidad, la misma técnica que han utilizado Radeke et al en su herramienta PIM, tal y como se ha explicado en el estado del arte [37]. Los patrones de Radeke tienen como inconveniente que están ligados a la herramienta PIM, y no están lo suficientemente detallados como para exportarlos a otras herramientas o metodologías. Este hecho ha favorecido que hayamos buscado entre otros autores que hayan trabajado con patrones de usabilidad. Finalmente, la decisión fue elegir los patrones de usabilidad definidos en el proyecto STATUS *(SofTware Arquitectures That support USability)* [42]. La selección de los patrones de STATUS estuvo motivada por las siguientes ventajas:

- El mecanismo de usabilidad de estos patrones está definido de forma abstracta desde una perspectiva orientada a objetos. Por lo tanto la aplicación de estos patrones a OO-Method se puede realizar fácilmente.
- Las soluciones propuestas en el proyecto STATUS no son específicas para un método de producción de software en particular ni para una plataforma software concreta.
- Estos patrones están descritos para incorporarse a la arquitectura de los sistemas desde las primeras fases de construcción del sistema.

El propósito de STATUS fue determinar la conexión entre una arquitectura software y la usabilidad del sistema software resultante [12][22]. Como resultado de este proyecto, se definieron un conjunto de patrones de usabilidad que deben ser considerados durante el proceso de generación de la aplicación. El analista debe capturar los requisitos de estos patrones preguntando al usuario de la aplicación. Estas preguntas son formuladas usando plantillas que contienen la información necesaria para configurar el patrón correctamente. Los requisitos sobre usabilidad capturados con estas plantillas, son modelados en la fase de análisis, diseñados en la fase de diseño y convertidos a código en la fase de implementación. Por lo tanto, estos patrones se tienen en cuenta a lo largo de todo el proceso de desarrollo del sistema.

En el proyecto STATUS se definen los patrones de usabilidad como la técnica o el mecanismo que se puede aplicar al diseño de la arquitectura de un sistema para dirigir una necesidad identificada a través de una de las *propiedades de la usabilidad*. Por *propiedades de usabilidad* se refiere a aquellos requisitos de usabilidad de un sistema que están más directamente relacionados con el dominio de la solución y que tienen una relación directa con las decisiones de diseño software. Estas propiedades expresan los heurísticos y principios de diseño que los investigadores del ámbito de la usabilidad han encontrado para tener una influencia directa sobre la usabilidad del sistema. Además de las propiedades de usabilidad, el concepto de *atributos de usabilidad* también está relacionado con los patrones de usabilidad. Un *atributo de usabilidad* es un componente preciso y medible del concepto abstracto que es la usabilidad. La relación entre patrones de usabilidad, propiedades de usabilidad y atributos de usabilidad es la que se muestra en la Figura 13.

Tal y como muestra la Figura 13, las propiedades de usabilidad forman parte de los requisitos que dirigen el proceso de diseño. Esto se lleva a cabo seleccionando patrones de diseño apropiados para satisfacer las propiedades requeridas. Se debe validar la arquitectura con respecto a estas propiedades y esta validación puede informar de la necesidad de volver a rediseñar algunos aspectos. Una vez los componentes han sido diseñados e implementados, el sistema resultante se puede testear con respecto a sus atributos de usabilidad. Si el resultado de este test no es satisfactorio, los componentes y la arquitectura del sistema deben ser rediseñados.

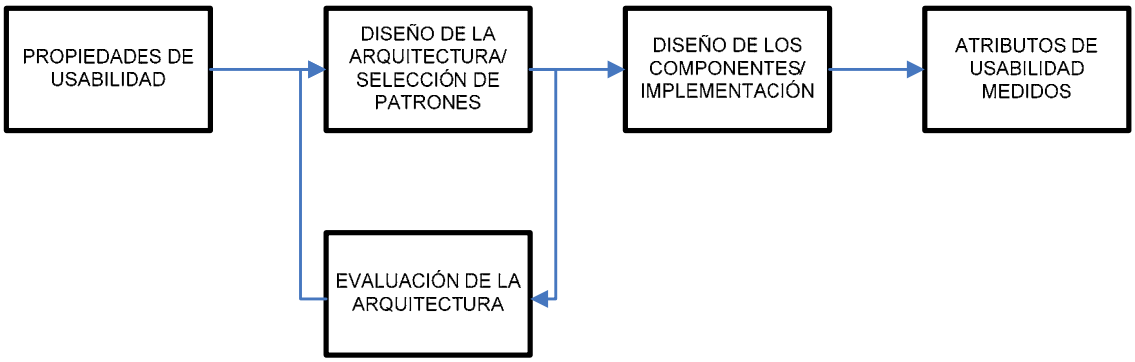

Figura 13. Proceso de diseño con los patrones de usabilidad

En la Figura 14 se muestra un ejemplo de las relaciones existentes entre atributos de usabilidad, propiedades de usabilidad y patrones de usabilidad. Los atributos de usabilidad se encuentran al nivel más abstracto posible, mientras que los patrones de usabilidad se encuentran en el menos abstracto. Por ejemplo, el patrón *asistente* utiliza el concepto de *consejo* para guiar al usuario durante la ejecución de una tarea compleja. La propiedad *consejo* mejora el atributo *facilidad de aprendizaje*.

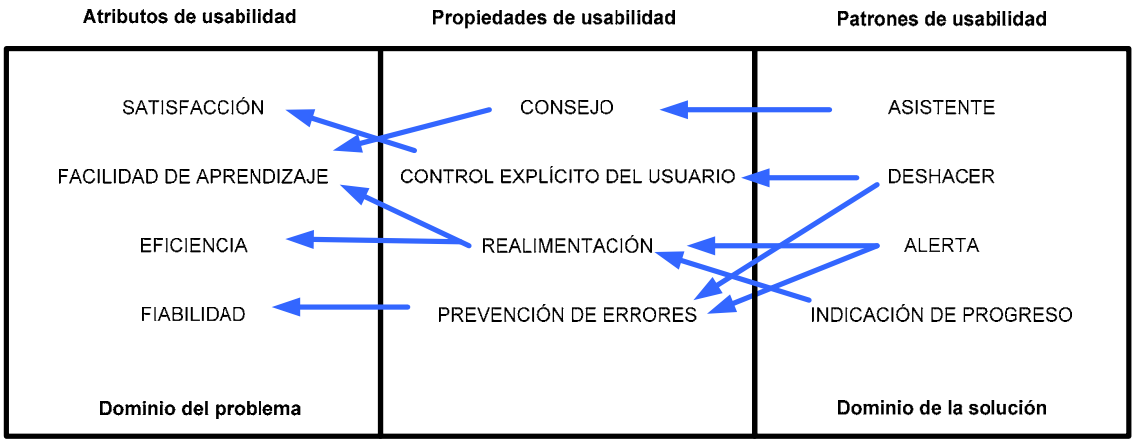

Figura 14. Ejemplo de relación entre atributos, propiedades y patrones de usabilidad

A partir de este proceso, el término *patrón de usabilidad* se refiere a la técnica o mecanismo que se puede aplicar al diseño de la arquitectura de un sistema software para dirigir una necesidad identificada por una propiedad de usabilidad dentro de la fase de captura de requisitos. Un patrón de usabilidad no es lo mismo que un patrón de diseño, ya que los patrones de usabilidad no especifican detalles de implementación en términos de clases, sino a un nivel mayor de abstracción. Para implementar la solución aportada por cada uno de los patrones de usabilidad se pueden implementar diversos métodos, no hay una opción única. Uno de los aspectos que comparten los patrones de usabilidad con los de diseño es que ambos tienen el objetivo de capturar la experiencia en el diseño de forma que pueda ser reutilizada por los diseñadores de software en problemas similares. Este hecho facilita la tarea de los diseñadores a la hora de resolver problemas concretos, ya que no parten de cero a la hora de abordarlos.

En este apartado se presenta cómo afecta la funcionalidad de los patrones de usabilidad al Modelo de Usabilidad propuesto. Los nuevos atributos detectados al estudiar los patrones de usabilidad de STATUS deben llevar asociados tanto fórmulas como indicadores para dar significado al valor numérico obtenido mediante las fórmulas.

Una vez modificado el Modelo de Usabilidad para soportar los nuevos atributos detectados, deben integrarse los patrones de usabilidad de STATUS dentro de un entorno de transformación de modelos como es OO-Method. Esta integración es necesaria debido a que la funcionalidad de estos patrones si no está reflejada en el Modelo de Usabilidad, menos aun está implementada actualmente en OO-Method. Lo primero que hay que decidir es dónde modelar la usabilidad en OO-Method. Algunos compiladores de modelos tienen un modelo que captura la interacción del usuario con el sistema, como Wisdom [29] o USIXML [44], al igual que OO-Method, que tiene un Modelo de Interacción. La usabilidad está fuertemente ligada a la interacción con el usuario. Es por ello razonable que el modelado de la usabilidad se haga desde este Modelo de Interacción.

Una vez está definido dónde se modelan estos patrones, hay que estudiar en qué aspectos puede mejorar la usabilidad de los sistemas generados. Para llevar a cabo esta labor, se van viendo las propiedades de cada uno de los patrones y en qué manera pueden mejorar la interacción con el usuario. Además muchos de los patrones son configurables, es decir, el usuario puede elegir cómo desea que se apliquen; por ejemplo: en caso de que el usuario deba recibir un mensaje de aviso a través de una ventana emergente, decidir si la ventana de aviso es o no bloqueante; o seleccionar si la ventana en la que se muestra el mensaje al usuario es de tipo “alerta” o “error”. Se debe valorar también en qué casos esta posibilidad de configuración debe o no ser tenida en cuenta. Por ejemplo, la posibilidad de mostrar una barra de progreso mientras se está ejecutando una acción no será configurable, ya que se que la barra de progreso se debería mostrar siempre.

La funcionalidad de los patrones de usabilidad, aunque sea representada en el Modelo de Interacción dentro del Modelado Conceptual, acaba convirtiéndose en código que implementa dicha funcionalidad. Por lo tanto, hay que estudiar el motor de generación de código que tiene el Compilador de Modelos. Estudiando el código de las clases que genera este compilador, se averigua dónde y cómo se pueden implementar los distintos patrones de usabilidad. Habrá patrones que podrán ser implementados añadiendo nuevos servicios a clases ya existentes y habrá otros que deban ser representados con nuevas clases creadas desde cero. La representación abstracta de estos cambios se hace en base a Diagramas de Clase y Diagramas de Secuencia, en donde participan las clases generadas por el Compilador de Modelos.

La Figura 15 muestra el proceso de generación de código usando un compilador de modelos. Se parte de un Modelado Conceptual formado por varios modelos que representa el espacio del problema. Esta fase es equivalente a la fase PIM (*Platform Independent Model)* de MDA. Cada uno de los modelos conceptuales representa varios aspectos del sistema, como persistencia, comportamiento e interacción, por ejemplo. Una vez construidos todos estos modelos, se aplica el Compilador de Modelos para generar el código de la aplicación software que formará lo que se llama el espacio de la solución. Este Compilador de Modelos es el que contiene el motor de transformación con la estrategia de generación de código. Esta fase es equivalente a la llamada PSM (*Platform Specific Model)* de MDA. El compilador de modelos genera un *Código Específico* para cada sistema software particular. Esta fase es equivalente al *Code Model* definido en MDA.

Aunque cada sistema tenga una implementación y un código específico distinto, la arquitectura de todos los sistemas generados será similar en todos ellos siempre que se trabaje sobre un mismo lenguaje de programación. Es decir, las clases que implementan las funciones del sistema y las relaciones existentes entre ellas, serán similares para todos los sistemas desarrollados. Solo cambiarán algunos de sus atributos y la lógica de sus servicios, la arquitectura del sistema, las relaciones entre clases y el resto de atributos y servicios se mantendrán constantes. Este código que será el mismo para todos los sistemas construidos, es lo que se llama *Código Genérico*. Este es el código utilizado para representar de forma abstracta, mediante Diagramas de Clase y Diagramas de Secuencia, los cambios a incorporar en el Compilador de Modelos.

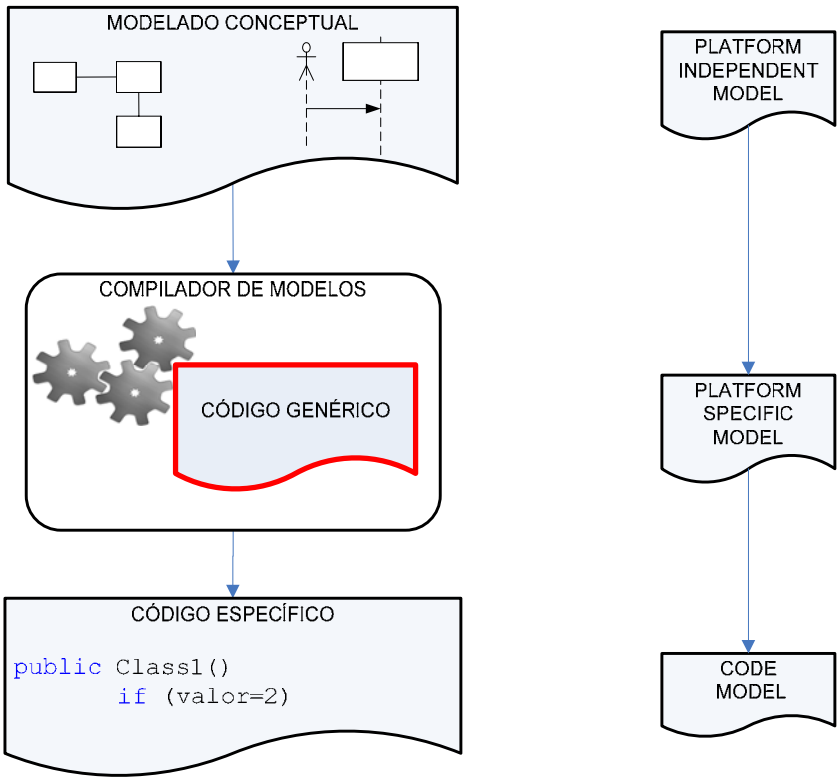

Figura 15. Proceso de generación de código en OO-Method

Es importante resaltar, que el término “clase” utilizado en el proceso de generación automática de código, es distinto dependiendo de la fase en la que nos encontremos dentro del proceso de generación de código. En la fase de *Modelado Conceptual*, el término “clase” significará una clase del modelo de objetos que se esté modelando. Con estas clases se representarán los atributos y los servicios de los objetos que formarán la lógica del sistema. Sin embargo, en el *Código Genérico*, este término indica una clase en el lenguaje de programación seleccionado. Esta clase será igual en todos los sistemas que se hayan generado sobre el mismo lenguaje de programación. Estas clases no contiene lógica del negocio, solo interesa las relaciones que mantiene con el resto de clases y cual es su función dentro del conjunto de clases del sistema. Por último, en la fase de *Código Específico*, el término “clase” representa una clase en el lenguaje de programación seleccionado que es específica de un determinado sistema. Estas clases son específicas de cada sistema construido, ya que contienen los métodos para resolver un caso práctico concreto.

Todo el proceso de incorporación de los patrones de STATUS en OO-Method está representado gráficamente en la Figura 16. Partiendo de una especificación del patrón tal y como aparece en el proyecto STATUS, se estudia la funcionalidad del patrón y se ven las ventajas que aportaría a los sistemas generados. En caso que la funcionalidad del patrón no esté soportada por OO-Method, se propone cómo incorporarlo en el proceso de generación de software. Esta incorporación implica cambios en los tres niveles de abstracción de OO-Method: *Modelado Conceptual, Compilador de Modelos* y *Código Generado*. Para el caso de los cambios a aplicar en el Compilador de Modelos, se ha utilizado el Código Genérico representado mediante Diagramas de Clase y Diagramas de Secuencia.

Sobre el Diagrama de Clases se pueden representar las nuevas clases y servicios que contienen la funcionalidad de los patrones de usabilidad. Para cada clase modificada por la incorporación de los patrones, se debe explicar el significado de los nuevos métodos y atributos. Sin embargo, solo con este diagrama no se puede expresar cual será la secuencia de acciones que se va a realizar al utilizar los patrones. Para ello necesitamos mayor detalle que puede ser alcanzado utilizando Diagramas de Secuencia. Usando una combinación de

Diagramas de Secuencia junto con Diagramas de Clase, se consigue la suficiente expresividad como para representar la forma en la que se pueden incorporar los patrones de usabilidad al Compilador de Modelos.

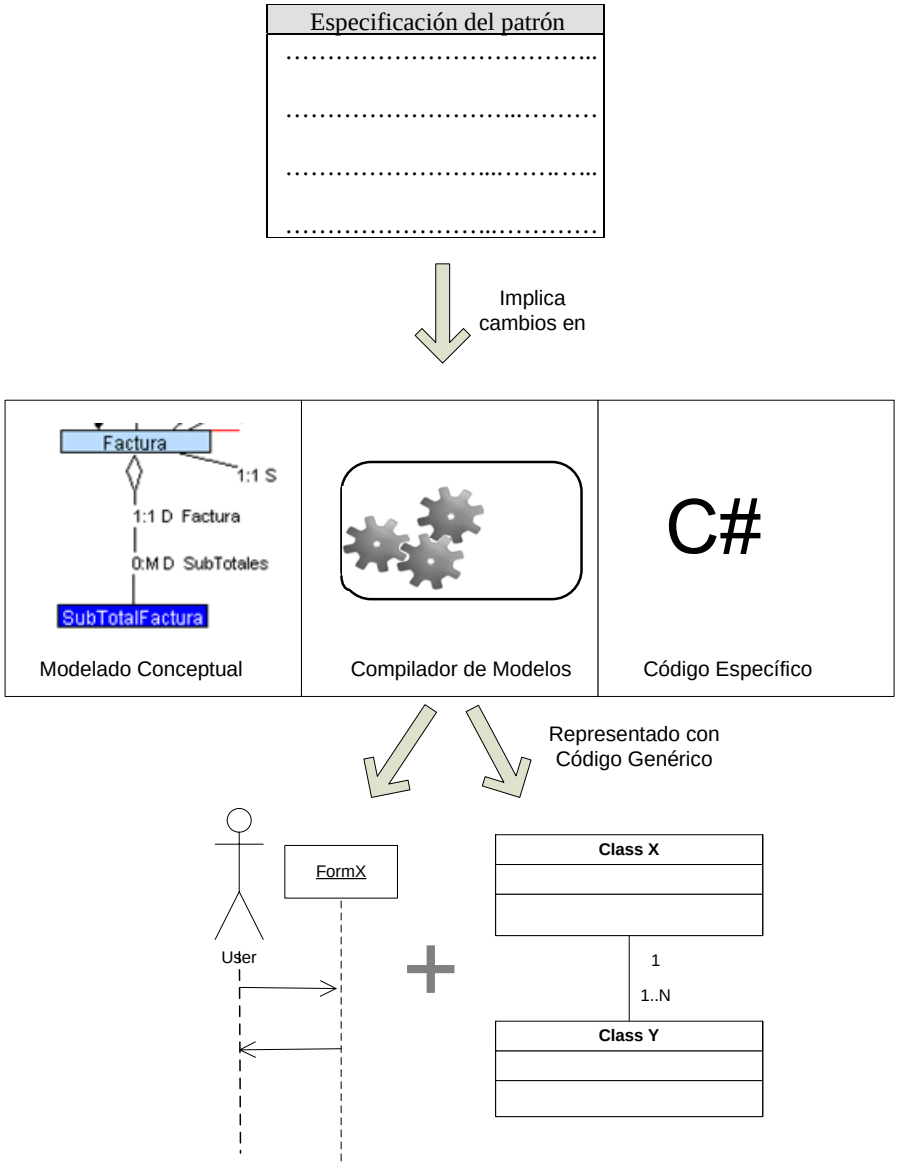

Figura 16. Proceso de incorporación de los patrones de STATUS en OO-Method

Por último, hay que definir una frontera clara entre los patrones que se aplicarán automáticamente y los que no. Es decir, hay que ver por un lado, qué patrones de usabilidad deben ser modelados por el analista dentro del Modelado Conceptual. Éstos deben ir incluidos en algunos de los modelos abstractos de OO-Method. Por otro lado, otros serán incluidos únicamente en el Compilador de Modelos. Estos últimos serán incorporados al código de manera transparente para el analista, siguiendo la generación automática de código y no son configurables por el analista:

- Por un lado, en el Modelado Conceptual se modelarán aquellos patrones de usabilidad que requieran cierta configuración por parte del analista, bien porque no siempre deben ser aplicados o bien porque su aplicación puede darse de distintas maneras.
- Por otro lado, se incluirán solo en el Compilador de Modelos aquellos patrones que se apliquen en cualquier caso y que además lo hagan siempre de la misma forma, es decir, no tengan varias posibilidades de configuración.

6.1. Patrones de STATUS

En esta sección se presentan los patrones de STATUS que se tendrán en cuenta a la hora de mejorar el Modelo de Usabilidad y el Modelado Conceptual de OO-Method. El conjunto de estos patrones de STATUS está agrupado por medio de familias. Una familia es un conjunto de patrones que tienen el mismo objetivo, donde cada miembro de la familia intenta alcanzar dicho objetivo de distinta forma. Este documento se ha centrado solo en una familia para abordar todo el proceso de incorporación al Modelo de Usabilidad y su implantación en el proceso de generación de software OO-Method. Para ello se ha buscado una familia que tuviera varios patrones aun no incorporados en el Modelo de Usabilidad y por lo tanto, aun no soportados por el Modelado Conceptual de OO-Method.

La familia que mejor se adapta a estas características es la familia de patrones llamada *Feedback* (retroalimentación). Esta familia agrupa todos los patrones que informan al usuario en todo momento de las acciones que está realizando el sistema. El objetivo de esta familia es que el usuario disponga en todo momento de información sobre el estado actual del sistema, si las peticiones del usuario están siendo atendidas y sobre el éxito o el fracaso de cada petición. A continuación se presentan los patrones que forman esta familia y como acomete cada uno de ellos estos objetivos.

Para la presentación de cada uno de los patrones de la familia Feedback, la estructura a seguir será la siguiente:

- **Identificación:** contiene un nombre, la familia a la que pertenece y posibles alias.
- **Problema**: lo que se intenta resolver con el patrón de usabilidad descrito.
- **Contexto:** situación en la que el patrón de usabilidad debería ser aplicado.
- **Solución:** está compuesta por:
  - **Guía de Elicitación de los Mecanismos de Usabilidad:** para cada recomendación de la comunidad HCI (*Human Computer Interaction*), han sido creadas una serie de preguntas que deben ser formuladas a los usuarios del sistema para capturar los requisitos de usabilidad.
  - **Guía de Especificación de Mecanismos de Usabilidad:** muestra la información que se debe rellenar para un sistema software concreto en la fase de captura de requisitos.

La familia Feedback está compuesta por cuatro patrones: *System Status Feedback, Interaction Feedback, Progress Feedback* y *Warning*.

6.1.1. System Status Feedback

IDENTIFICADOR
Nombre: System Status Feedback
Familia: Feedback
Alias: Status Display Modelling Feedback Area
PROBLEMA
Qué información se debe obtener y especificar para que la aplicación proporcione a los usuarios información del estado del sistema.
CONTEXTO
Cuando ocurren algunos cambios en el estado del sistema, el usuario debería ser avisado, especialmente cuando el cambio de estado afecta a la información de un estado que varía a lo largo del tiempo (Por ejemplo, la barra de estado en las aplicaciones de Windows, horarios de trenes, Displays VCR…) Cuando ocurren cambios que son importantes para el usuario o

Cuando ocurren fallos que son importantes para el usuario:

2. Durante la ejecución de una tarea
3. Porque no hay suficientes recursos en el sistema
4. Porque los recursos externos no están trabajando correctamente

SOLUCIÓN

Guía de Obtención de los Mecanismos de Usabilidad:

Recomendación HCI	Temas a discutir con los stakeholders
1. Los expertos HCI sostienen que el usuario quiere ser informado cuando ocurre un cambio de estado.	Los cambios en el estado del sistema pueden ser provocados por peticiones del usuario o por otras acciones o cuando haya un problema con un recurso externo u otro recurso del sistema. 1.1 ¿El usuario quiere que el sistema le avise de los estados del sistema?, en ese caso, ¿de cuales? 1.2 ¿El usuario quiere que el sistema le avise de fallos del sistema (cualquier operación que el sistema no es capaz de completar, pero no son fallos provocados por entradas incorrectas del usuario)? En ese caso ¿Cuáles? 1.3 ¿El usuario quiere que el sistema avise si no hay suficientes recursos para ejecutar las órdenes en curso? En ese caso, ¿qué recursos? 1.4 ¿El usuario quiere que el sistema le avise si hay problemas con los recursos externos o dispositivos con los que el sistema interactúa? En ese caso, ¿Cuáles?
2. Se deben elegir pantallas bien diseñadas para la información a mostrar. Éstas deben ser no bloqueante si la información no es importante, pero bloqueante si algo importante ocurre. Las pantallas se deben colocar de manera que enfaticen las cosas importantes y dé menos importancia a las triviales y prevenga la confusión entre distintas partes de la información mostrada. La atención se debe centrar en la información importante con colores brillantes, parpadeo, sonido o los 3.	Por cada situación identificada en el punto 1, discutir con el usuario: 2.1 ¿Qué información se debe mostrar al usuario? 2.2 ¿Qué información se tendrá que mostrar de manera bloqueante porque pertenece a una situación crítica? Representada por una ventana emergente en la pantalla principal que impide al usuario seguir hasta que no la cierre. 2.3 ¿Cuál de esta información tendrá que ser resaltada porque es importante pero no crítica? Usando diferentes colores, sonidos, tamaños, etc. 2.4 ¿Cuál de esta información será mostrada en el área del estado? Mediante algún tipo de ventana emergente en área del estado del sistema.

	Para cada componente que muestre el estado del sistema, de acuerdo a su importancia, el las posibilidades de visualización irá desde los componentes bloqueantes (por ejemplo, una ventana en el área principal que impide al usuario continuar hasta que sean cerradas), pasando por resaltar ciertas zonas (con diferentes colores, sonidos, movimientos o tamaños) hasta las percepciones del ojo humano (como la identificación de un icono ubicado en el área del estado del sistema). Se puede observar que durante el proceso de elicitación de requisitos, el debate sobre el tipo exacto de respuesta se puede dejar hasta la fase de diseño de la interfaz. La importancia de las diferentes situaciones sobre qué información sobre el estado se mostrará y, además, el tipo general de la ventana (bloqueante, resaltada o estándar) que se mostrará no se debe discutir en esta fase.
3. Respecto a la localización del indicador de *feedback*, la literatura HCI menciona que los usuarios prefieren un lugar donde sepan que pueden encontrar fácilmente su información sobre el estado. A parte del espacio en la pantalla donde los usuarios trabajan, ellos prefieren ver el *feedback* en el centro o en la parte superior de la pantalla, y es menos agradable para ellos que se encuentre al pie de la pantalla. La práctica habitual de colocar la información sobre cambios del estado en una línea que se muestra en el pie de la pantalla es desaconsejable, especialmente si tiene un trazo fino sobre un fondo gris. Se debe recordar que las personas nacidas en la cultura europea o americana están acostumbrados a leer de izquierda a derecha y de arriba abajo, y algo colocado en la esquina superior izquierda se verá fácilmente.	3.1 ¿Las personas que usan el sistema son de diferentes culturas? En ese caso, el sistema debe presentar el estado del sistema en la forma adecuada (de acuerdo a la cultura del usuario). Así que se debe preguntar al usuario sus costumbres y cultura. 3.2 ¿Cuál es el mejor lugar para ubicar la información de *feedback* de cada situación?
Guía de Especificación de Mecanismos de Usabilidad:	
La siguiente información debería ser instanciada en el documento de captura de requisitos: - El estado del sistema que se debería relatar es X. La información a mostrar en el área de estado es… La información a resaltar es …. La información bloqueante es ….. - El sistema software necesitará suministrar *feedback* sobre los fallos I, II, III ocurridos en las tareas A, B, C respectivamente. La información relacionada con los fallos I, II, etc. se debe mostrar en el área de estado… La información relacionada con los fallos III, IV, etc. se debe mostrar en formato resaltado. La información relacionada con los fallos V, VI, etc. se debería mostrar en formato bloqueante. - El sistema software proporciona *feedback* sobre los recursos D, E, F cuando falla IV, I y VI, respectivamente. La información a presentar sobre esos recursos es O, P, Q. La información	

que está relacionada con los fallos I, II, etc. se debe mostrar en el área de estado. La información relacionada con los fallos III, IV, etc. se debe mostrar en formato resaltado. La información relacionada con los fallos V, VI, etc. se debe mostrar en formato bloqueante. - El sistema software necesitará suministrar *feedback* a los recursos externos G, J, K, cuando los fallos VII, VIII y IX ocurran respectivamente. La información a presentar sobre esos recursos es R, S, T. La información relacionada con los fallos I, II, etc.. se debe mostrar en el área de estado. La información relacionada con los fallos con los fallos III, IV, etc. se debe mostrar con formato resaltado. La información relacionada con los fallos V, VI, etc. se debe mostrar de manera bloqueante.

Tabla 27. Descripción de System Status Feedback

6.1.2. Interaction Feedback

IDENTIFICADOR	
Nombre: Interaction Feedback	
Familia: Feedback	
Alias: Interaction Feedback Modelling Feedback Area	
PROBLEMA	
Qué información debe ser obtenida y especificada para mostrar al usuario que el sistema ha escuchado su petición.	
CONTEXTO	
Cuando el usuario realiza un evento de interacción, como un clic de ratón, movimiento de ratón, apretar una tecla, etc. el sistema debe informar al usuario de que la interacción se ha aceptado.	
SOLUCIÓN	
Guía de Obtención de los Mecanismos de Usabilidad:	
Recomendación HCI	Temas a discutir con los stakeholders
1. Ofrecer un *feedback* proporcional a la escala del evento de interacción y su importancia, solo para confirmar que el sistema se ha percatado del evento. Esta funcionalidad se puede presentar: marcando un botón, sombreando una palabra, cambiando la forma del cursor de los objetos involucrados, etc. Se debe proporcionar siempre un *feedback* visual y permitir al usuario habilitar *feedbacks* con sonido para todos los eventos de interacción.	1.1. Para cada acción que el usuario solicite al sistema se debe decidir, qué tipo de acuse de recibo mostrará el sistema (marcando un botón, sombreando una palabra, cambiando la forma del cursor de los objetos involucrados, otra señal visual, audio, etc). Este tema se puede decidir en las tareas de Diseño de la Interfaz. 1.2. Se debe verificar que la aplicación proporciona el feedback en 0.1 milisegundos después de cada vez que se pulse una tecla, se mueva el ratón, o el usuario haga cualquier otra actividad sobre el sistema.
Guía de Especificación de Mecanismos de Usabilidad:	
La siguiente información debe ser instanciada en el documento de requisitos: El sistema ha de responder a los eventos de interacción A, B, C. C tiene una importancia alta. La respuesta del sistema se hará en 0.1 milisegundos después de la interacción del usuario.	

Tabla 28. Descripción de Interaction Feedback

6.1.3. Progress Feedback

<table>
<tr><td colspan="2">IDENTIFICACIÓN</td></tr>
<tr><td colspan="2">Nombre: Progress feedback</td></tr>
<tr><td colspan="2">Familia: Feedback</td></tr>
<tr><td colspan="2">Alias: Progress Indicator
Progress
Show Computer is Thinking, Time to Do Something Else
Modelling Feedback Area</td></tr>
<tr><td colspan="2">PROBLEMA</td></tr>
<tr><td colspan="2">Qué información debe ser obtenida y especificada para proporcionar a los usuarios la información que está relacionada con la evolución de las tareas solicitadas.</td></tr>
<tr><td colspan="2">CONTEXTO</td></tr>
<tr><td colspan="2">Cuando un proceso interrumpe la interfaz de usuario durante dos segundos o más.</td></tr>
<tr><td colspan="2">SOLUCIÓN</td></tr>
<tr><td colspan="2">Guía de Obtención de los Mecanismos de Usabilidad:</td></tr>
<tr><td>Recomendación HCI</td><td>Temas a discutir con los stakeholders</td></tr>
<tr><td>1. Para procesos que se estén ejecutando para 2 o más segundos: si el proceso es crítico, los usuarios no deberían poder hacer otra cosa hasta que esta tarea se completara. Si la tarea no es crítica y dura más de 5 segundos, los usuarios deberían poder ejecutar otra operación si lo desean. Se debería avisar al usuario cuando las operaciones que se estén ejecutando finalicen.</td><td>1.1 Qué tareas es probable que duren más de 2 segundos y cuales de ellas son críticas.
1.2 Cómo se informará al usuario cuando el proceso termine.</td></tr>
<tr><td>2. Mostrar un indicador animado que indique cuánto progreso se ha hecho. Además se debería informar al usuario textual o gráficamente de lo siguiente:
o Qué se está ejecutando actualmente.
o Qué proporción de la operación se ha realizado.
o Cuánto tiempo falta para terminar.
o Cómo parar la operación o cancelarla si el tiempo que falta para terminar es mayor de 10 segundos.
Sobre el tiempo restante: si el tiempo se puede calcular, se debe usar un Indicador de Tiempo de Progreso, bien mediante una figura o gráficamente. Se debe mostrar un indicador del tiempo que falta o un indicador de la porción que falta por completarse. Si el tiempo no se puede estimar, pero el proceso tiene fases identificables, se deben indicar las fases completadas, y las fases que restan. Se debe usar un Checklist de Progreso si no existe ninguna de estas posibilidades, después al menos se debe indicar el número de unidades procesadas. Si no se puede cuantificar cuánto tiempo llevará el proceso entonces</td><td>2.1. Cómo se informará al usuario sobre el progreso de las diferentes tareas y qué información se desea para cada una.
2.2. Se debe verificar que la aplicación no consume más de 1 minuto para mostrar el indicador de progreso y actualizar el feedback a una velocidad que dé la impresión al usuario de que la operación aun se sigue realizando, por ejemplo cada 2 segundos.</td></tr>
</table>

simplemente se debería mostrar un indicador como que el proceso sigue en marcha, usando para ello un Indicador de Progreso Indeterminado.	
Guía de Especificación de Mecanismos de Usabilidad:	
La siguiente información se debe instanciar en los documentos de requisitos: Las tareas U, V, Z necesitan más de 2 segundos, por lo tanto necesitarán un *feedback* de progreso. Para ellos, la información a mostrar será R, S, T en la parte A, B, C, respectivamente. La información debe aparecer en menos de 1 segundo y se debe refrescar cada 2 segundos.	

Tabla 29. Descripción de Progress Feedback

6.1.4. Warning

IDENTIFICACION	
Nombre: Warning	
Familia: Feedback	
Alias: Warning, Think Twice	
PROBLEMA	
Qué información debe ser obtenida y especificada para solicitar al usuario la confirmación de la acción solicitada en caso de que ésta tenga consecuencias irreversibles.	
CONTEXTO	
Cuando una acción que tiene serias consecuencias es solicitada por el usuario.	
SOLUCIÓN	
Guía de Obtención de los Mecanismos de Usabilidad:	
Recomendación HCI	Temas a discutir con los stakeholders
1. Para cada acción que el usuario puede realizar hay que considerar lo siguiente: si la acción es reversible, la proporción de acciones reversibles que el sistema soporta, la frecuencia con la que la acción se hace, el grado de impacto que puede causar, y la inmediatez del feedback. Con todo esto se debe considerar que sería más apropiado, si un aviso, una confirmación o una autorización.	1.1. Discutir con el usuario todas las tareas que se deben realizar y sus consecuencias (considerar la frecuencia de estas acciones y su impacto) y cuales requieren confirmación (tener en cuenta no sobrecargar al usuario con avisos).

2. Puede que los usuarios no entiendan las consecuencias de sus acciones. Por eso, el aviso debería contener los siguientes elementos: - Un resumen del problema y la condición que ha disparado el aviso. - Una pregunta para que el usuario indique si desea continuar con la acción o realizar otras acciones. Deben haber dos posibles elecciones para el usuario, seguir con la acción o abandonarla. - También puede incluir una descripción más detallada de la situación para ayudar al usuario a tomar la decisión. Las opciones deberían incluir un verbo que se refiera a la acción solicitada. - En algunos casos pueden haber más de dos opciones. El aumento del número de opciones se puede aceptar en algunos casos, pero se debe tender a minimizar el número de opciones.	2.1. ¿Qué información se mostrará para cada una de las tareas a confirmar? Se debe recordar que ha de proporcionar las consecuencias de la acción y las alternativas que se le darán al usuario.
Guía de Especificación de Mecanismos de Usabilidad:	
La siguiente información deberá ser instanciada en el documento de requisitos: Las tareas U, V, Z necesitarán aviso. Para ellas la información a mostrar será R, S, T respectivamente.	

Tabla 30. Descripción de Warning

6.2. Cambios en el Modelo de Usabilidad

Una vez descritos los patrones que forman la familia Feedback, hay que decidir qué nuevos atributos generan dentro del Modelo de Usabilidad. La descripción de la familia coincide con un atributo ya existente: *Realimentación Informativa.* Por lo tanto parece lógico agrupar todos los patrones dentro de este atributo del Modelo de Usabilidad. De esta forma el atributo Realimentación Informativa pasaría a ser un agrupador de atributos que agruparía los atributos:

- Realimentación del estado del sistema: este atributo mide la capacidad del sistema de informar al usuario sobre el éxito o el error en la ejecución de servicios.
- Realimentación de interacción: este atributo mide la capacidad del sistema de indicar al usuario que su petición está siendo atendida.
- Realimentación de progreso: este atributo indica si el sistema es capaz de mostrar al usuario que su petición se está ejecutando y cuánto tiempo le resta para su finalización. Este atributo solo tiene sentido para tareas largas. Según la comunidad HCI, se entiende por tarea larga aquella que dura más de dos segundos.
- Aviso: este atributo indica la capacidad del sistema de mostrar mensajes de aviso o advertencia al usuario cada vez que se cumpla una situación “extraña” dentro de la lógica de negocio. La definición de qué es una situación extraña es proporcionada por el analista, que la define con el formato de una condición. Con este mensaje de aviso, el usuario debe ser capaz de confirmar la ejecución de la acción o cancelarla. Cada vez que la condición definida por el analista sea cierta, se debe mostrar el mensaje de aviso para que el usuario confirme que realmente quiere ejecutar esa acción.

La Tabla 31 muestra el Modelo de Usabilidad con los nuevos atributos detectados. Los nuevos atributos se han resaltado en negrita.

1. **Facilidad de aprendizaje** 1.1. Facilidades de Ayuda 1.1.1. Completitud de Documentación 1.1.2. Documentación Multiusuario 1.1.2.1. Distinción por Capacitación 1.1.2.2. Distinción por Rol 1.2. Predecibilidad 1.2.1. Grado de Significación de Iconos 1.2.2. Título de Iconos o Enlaces 1.2.3. Determinación de la Acción **1.3. Realimentación Informativa** **1.3.1Realimentación del Estado del Sistema** **1.3.2Realimentación de Interacción** **1.3.3Realimentación de Progreso** **1.3.4 Aviso** 1.4. Facilidad de Memorización 2. **Comprensibilidad** 2.1. Legibilidad 2.1.1. Tamaño de Fuente 2.1.2. Contraste 2.1.3. Disposición 2.2. Facilidad de Lectura 2.2.1. Agrupación Cohesiva de la Información 2.2.1.1. Agrupación por Disposición 2.2.1.2. Agrupación por Formato 2.2.2. Densidad de la Información 2.3. Familiaridad de conceptos 2.3.1. Significación del Etiquetado 2.3.2. Internacionalización 2.4. Reducción de la Carga de Trabajo 2.4.1. Brevedad 2.4.1.1. Inicialización de Valores 2.4.1.1.1. Completitud de Valores Iniciales 2.4.1.1.2. Modificabilidad de Valores Iniciales 2.4.1.2. Minimización de acciones 2.4.2. Autodescripción 2.5. Orientación al usuario 2.5.1. Calidad de los Mensajes 2.5.2. Navegabilidad 3. **Operabilidad** 3.1. Capacidades de Instalación	3.1.1. Facilidad de Instalación 3.1.2. Multiplicidad de Instalación 3.1.3. Capacidad de Actualización 3.1.4. Transparencia de Actualización 3.2. Validación de datos 3.3. Capacidad de Control 3.3.1. Aplazamiento de Edición 3.3.2. Soporte de Cancelación 3.3.3. Ejecución Explícita 3.3.4. Soporte de Interrupción 3.3.5. Soporte de Deshacimiento 3.3.6. Soporte de Rehacimiento 3.4. Capacidad de Adaptación 3.4.1. Adaptabilidad 3.4.2. Adaptatividad 3.5. Consistencia 3.5.1. Comportamiento Constante de Controles 3.5.2. Permanencia de Controles 3.5.3. Estabilidad de los Controles 3.5.4. Consistencia de Orden 3.5.5. Consistencia de Etiquetado 3.6. Gestión de Errores 3.6.1. Prevención de Errores 3.6.2. Recuperación de Errores 3.7. Monitorización del Estado del Sistema 4. **Grado de atracción** 4.1. Metáfora 4.2. Uniformidad del Color del Fondo 4.3. Uniformidad del Color de la Fuente 4.4. Uniformidad del Estilo de la Fuente 4.5. Uniformidad del Tamaño de la Fuente 4.6. Uniformidad en Disposición de Elementos 4.7. Atracción Subjetiva 5. **Conformidad** 5.1. Cumplimiento con ISO/IEC 9126 (partes 1 y 3) 5.2. Cumplimiento con ISO 9241-10 5.3. Cumplimiento con Guía de Estilo de Microsoft 5.4. Cumplimiento con Guía de Estilo de Java

Tabla 31. Modelo de Usabilidad con los atributos detectados por el patrón Feedback

Una vez incorporados los nuevos atributos en el Modelo de Usabilidad, hay que definir fórmulas para medirlos e indicadores para interpretar los valores de usabilidad obtenidos con las fórmulas. Para definir las fórmulas, es necesario incorporar las primitivas conceptuales en OO-Method para soportar la funcionalidad de los patrones de usabilidad, ya que las fórmulas detectarán el porcentaje de uso de estas primitivas. Por ello, previamente a la definición de las fórmulas, se van a presentar los cambios necesarios en OO-Method para soportar los patrones de la familia Feedback a lo largo de todo el proceso de desarrollo.

7. CAMBIOS EN EL MODELADO CONCEPTUAL

En esta sección se describen las primitivas conceptuales que hay que incorporar a OO-Method para soportar la funcionalidad de los patrones de usabilidad. Aunque los patrones de usabilidad están muy ligados a la interacción del usuario con el sistema, no es solo el Modelo de Interacción el que se ve modificado por dichos patrones, sino todos los modelos que conforman el Modelado Conceptual de OO-Method: Modelo de Objetos, Modelo Dinámico, Modelo Funcional, y como no, Modelo de Interacción. Esto es debido a que los patrones de usabilidad no solo describen la forma en la que el sistema interactúa con el usuario, sino que también afectan a la forma en la que el sistema ejecuta los servicios.

La Figura 17 muestra el proceso de generación de OO-Method resaltando la fase en la que se va a centrar este apartado para la incorporación de los patrones de usabilidad: los Modelos Conceptuales.

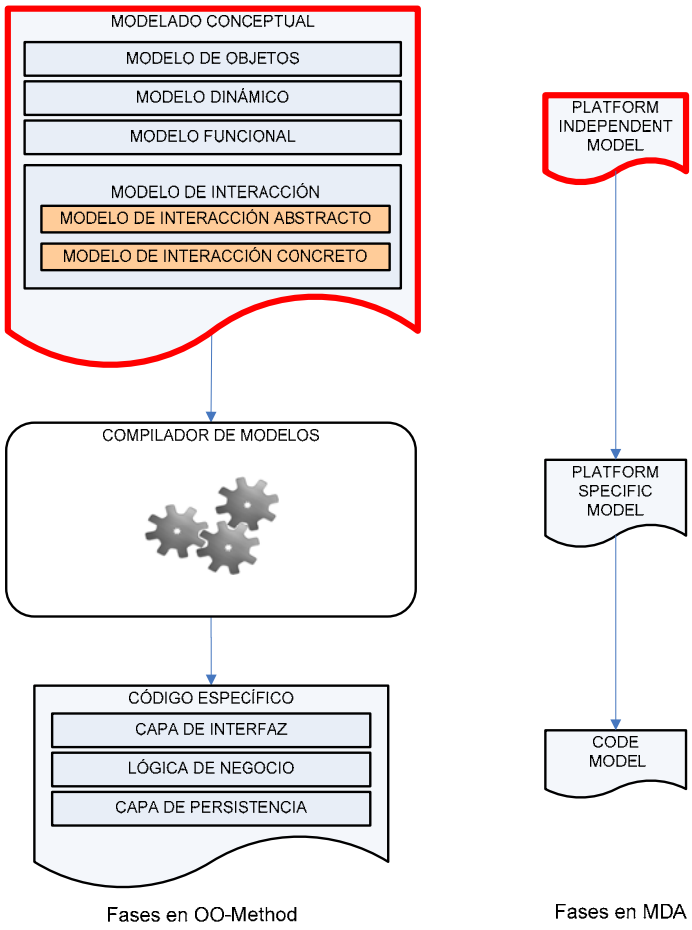

Figura 17. Fases de OO-Method y MDA en las que se centrará el modelado conceptual de los patrones

Las primitivas conceptuales definidas deben asegurar que se consigue, para cada patrón, toda la funcionalidad definida en el proyecto STATUS. Habrá funcionalidad que será generada por el Compilador de Modelos sin la ayuda del analista y habrá funcionalidad que será derivada de las primitivas conceptuales creadas por el analista. Es importante hacer esta distinción entre las funcionalidades de los patrones porque solo las funcionalidades del segundo tipo afectan al Modelado Conceptual. Sin embargo, ambos tipos de funcionalidad afectan al Compilador de Modelos.

Estos cambios en el Modelado Conceptual de OO-Method implican también cambios en la herramienta que soporta este método, OlivaNOVA. Estos cambios posibilitan el modelado de las nuevas primitivas como una característica más de OlivaNOVA. Junto con la descripción de las nuevas primitivas, se presenta un prototipo de cómo sería la herramienta OlivaNOVA si implementara el modelado de las nuevas primitivas generadas por los patrones de la familia Feedback.

7.1. Definición de nuevas primitivas de modelado

Para cada uno de los patrones de la familia Feedback, se describen las mejoras que puede aportar a las aplicaciones generadas con OO-Method. Una vez detectadas estas mejoras, se presenta la expresividad necesaria a nivel de Modelado Conceptual que sería necesaria para representar la funcionalidad de los patrones. Tal y como se ha comentado anteriormente, un mismo patrón de usabilidad, puede afectar a más de un Modelo Conceptual de OO-Method, como se puede apreciar a continuación.

7.1.1. System Status Feedback

Después de ejecutar una acción, el sistema debería informar al usuario si la acción se ha ejecutado correctamente o no. Actualmente, los sistemas generados con OO-Method ya informan al usuario de los servicios que no se ejecutan satisfactoriamente, debido a que no se satisfacen precondiciones, restricciones de integridad o que la ejecución del servicio implica una transición entre estados no válida.

Aunque el sistema ya muestra un mensaje de error cuando ocurre una de estas situaciones, hay ciertos mensajes de error que no son configurables por el analista, por ejemplo aquellos relacionados con una transición entre estados del objeto no definida en el Modelo Dinámico (transición entre estados no válida). El analista debería poder seleccionar el mensaje de error y en aquellos casos en que no haya introducido ninguno, el sistema mostrará un mensaje de error por defecto, genérico para cualquier sistema. Este mensaje por defecto es generado de forma automática por el Compilador de Modelos.

Actualmente, si el servicio se ejecuta correctamente, el usuario no recibe ningún tipo de información. Si el usuario quiere comprobar que ha tenido éxito la acción, debe comprobarlo él mismo a través de consultas. Estas consultas no tienen importancia si la información es visible desde la misma interfaz en la que se ha lanzado el servicio, pero si para comprobar que el servicio se ha ejecutado correctamente hay que hacer navegaciones entre interfaces, el precio que paga el usuario para comprobar el éxito del servicio es muy caro. El sistema debe proporcionar un mecanismo para que el analista decida qué servicios necesitan mostrar un mensaje de que se han ejecutado correctamente y cuales no, dependiendo de los requisitos de cada usuario.

Hay un aspecto de esta funcionalidad del patrón que sí puede añadirse automáticamente en el Compilador de Modelos sin que el analista tenga que modelar nada. Es la parte de añadir a todos los servicios modelados un mensaje de aviso que se mostrará al usuario cuando la acción se ejecute correctamente y otro cuando falle por culpa de una transición no definida en el Modelo Dinámico para ese servicio. Ambas funcionalidades del patrón de usabilidad se aplicarán automáticamente a todos los servicios existentes en el sistema. Para el caso de un mensaje de éxito en la ejecución del servicio, el mensaje que se mostraría por defecto cuando la ejecución sea correcta podría ser “Acción realizada correctamente”. En cambios, cuando se produzca un error por transición entre estados no válida, el mensaje por defecto podría ser “El servicio X no se puede ejecutar en el actual estado del objeto Y”. Donde X es el nombre del servicio que se ejecuta e Y el objeto que implementa el servicio. Sin embargo, aunque esta sea la funcionalidad por defecto, se debe dar la posibilidad al analista de cambiar los textos de los mensajes por otros distintos para un servicio determinado en ambos casos e incluso ofrecer la posibilidad de omitir los mensajes.

El modelado de este patrón de usabilidad afecta de la siguiente forma a estos dos modelos:

- Modelo de Objetos: en este modelo se inserta el texto para el caso en que se produzca un error provocado por no estar definida la transición entre estados del objeto dentro del Modelo Dinámico. También se inserta el texto para el caso en que la acción se ejecute correctamente:
    - El analista puede seleccionar qué servicios definidos en el Modelo de Objetos no desea que muestren el mensaje de error por defecto cada vez que el usuario lance su ejecución en un estado del objeto desde el cual no está permitida su ejecución. Para cada servicio que seleccione, el analista debe introducir el texto que se mostrará al usuario en caso de error provocado por el Modelo Dinámico.
    - Se decide también el texto que se mostrará cuando la acción se ejecute correctamente. Si no se introduce ningún texto, la aplicación mostrará uno genérico creado por el propio Compilador de Modelos de OO-Method.
- Modelo de Interacción Concreto: el analista decide la forma en la que mostrar el mensaje al usuario. Esta elección se hace en base a parámetros a los que el analista debe dar valor. El analista deberá configurar las siguientes características para dos casos, en el caso de mensajes que se mostrarán si la ejecución ha sido correcta y los que se mostrarán en caso de error. El modelado de los mensajes de error en este modelo incluye tanto los provocados por incumplir una precondición, como por incumplir una restricción de integridad, como el provocado por ejecutar un servicio en un estado del objeto desde el cual no está permita su ejecución. Los parámetros a configurar tanto para los mensajes de error, como para los de éxito en la ejecución son los siguientes:
    - Mostrar mensaje mediante texto:
        - En una nueva ventana:
            - Bloqueante (modal) o no boqueante
            - Tipo de Mensaje: aviso, error o alerta
            - Fuente del texto
            - Tamaño
            - Color
            - Alineación
        - Texto en alguna parte de la ventana principal:
            - Ubicación en la pantalla principal
            - Fuente del texto
            - Tamaño
            - Color
            - Alineación
    - Mostrar mensaje mediante iconos:
        - Icono a mostrar, por ejemplo  para cuando la acción se ejecute correctamente y  cuando haya algún fallo
        - Dónde mostrar el icono dentro de la ventana principal
    - No mostrar ningún tipo de mensaje.

Por defecto, los mensajes que avisen del éxito en la ejecución de un servicio se mostrarán mediante el icono  en la esquina inferior derecha de la ventana principal. En cuanto a los mensajes de error, se mostrarán en una ventana modal de tipo error con fuente Arial, tamaño 10, color negro y alineación centrada.

7.1.2. Interaction Feedback

Actualmente los sistemas generados con OO-Method poseen ya parte de la funcionalidad de este patrón incorporada. Por ejemplo, cuando el usuario lanza una acción, el puntero del ratón se transforma en un reloj de arena, indicando que la acción se está ejecutando. Sin embargo, hay un aspecto que no se contempla, y es la posibilidad de deshabilitar los botones y los campos editables de la ventana mientras dura la ejecución del servicio solicitado. Por un lado, sería una forma más de avisar al usuario de que su petición se está atendiendo y por otra facilita el trabajo al usuario, ya que impide que lance dos veces la misma acción por equivocación. Esta última funcionalidad pertenece a otro patrón llamado “Prevención de errores”.

Con esta funcionalidad se evitaría que el usuario haga doble clic sobre el botón en lugar de uno, bien por equivocación o bien por creer que su petición no ha sido aceptada. En cuanto el usuario lance la ejecución de un servicio, se deberían deshabilitar todos los botones de la ventana desde la cual se haya lanzado la acción, excepto el botón para cancelar la acción en curso.

Esta funcionalidad, llevada a los patrones de presentación de OO-Method solo afectará a las Unidades de Interacción de Servicio. Cuando el usuario pulse el botón *OK* para lanzar la acción asociada a esa Unidad de Interacción de Servicio, los campos editables se deshabilitarán, así como el botón *OK*. El botón *Cancel*, por el contrario, permanecerá habilitado por si el usuario quiere cancelar la acción en el transcurso de la ejecución.

La funcionalidad de este patrón es interesante para todas las Unidades de Interacción de Servicio que se definan en el Modelo de Interacción, ya que por un lado informan al usuario de que se ha atendido su petición y por otro previene de posibles errores por parte del usuario. Además, esta funcionalidad no presenta posibilidades de configuración por parte del analista, su funcionalidad es totalmente cerrada. Por ello, debería estar implícito en el código generado a partir de todas las Unidades de Interacción de Servicio. Por lo tanto, el Compilador de Modelos añadirá el código necesario al sistema generado para soportar la funcionalidad de este patrón de manera automática sin que el analista tenga que modelar nada en la fase de Modelado Conceptual. Así que la funcionalidad de este patrón solo afectará al Compilador de Modelos, no al Modelado Conceptual.

7.1.3. Progress Feedback

Las aplicaciones generadas con OO-Method actualmente, ya disponen de esta funcionalidad en algunos casos. Cuando se seleccionan varias instancias de un objeto y se lanza una acción sobre ellas, aparece una barra de progreso indicando el tiempo que resta para que la acción se ejecute en todas ellas. Sin embargo, esta funcionalidad no está disponible en las transacciones. Las transacciones son un candidato ideal para usar la funcionalidad de este patrón porque engloban un conjunto de acciones complejas, y por tanto, el tiempo que tarden en ejecutarse puede ser considerable. Se podría añadir la funcionalidad de una barra de progreso en los dos tipos de transacciones existentes en OO-Method:

- Para la ejecución de las transacciones globales, donde se ejecuta el conjunto de servicios que forman la transacción para todas las instancias implicadas. Una transacción global engloba un conjunto de actividades que pueden ser servicios y transacciones locales. En una transacción global se deben ejecutar todos los servicios que la componen, si alguno de los servicios falla, toda la transacción global falla y ningún servicio de la transacción se ejecuta, es lo que se conoce como “todo o nada”. Además, las transacciones globales siguen el principio de atomicidad, es decir, no se pueden observar estados intermedios de la transacción, sino solo el resultado final. Se debería mostrar una barra de progreso que vaya avanzando cada vez que se ejecute la transacción global sobre una instancia.
- Esta funcionalidad también se puede aplicar para el caso de las transacciones locales, que son transacciones, que al igual que las globales, están formadas por un

conjunto de servicios y otras transacciones. También siguen el principio de atomicidad y de todo o nada. Se diferencian de las globales en que las locales solo afectan a una instancia de un objeto. Para este tipo de transacciones puede ser interesante que el usuario vea en todo momento qué porción de los servicios que forman la transacción se ha realizando en un instante de tiempo determinado.

Esta funcionalidad del patrón aplicada a ambos tipos de transacciones se generará automáticamente por el Compilador de Modelos, tal y como ya se hace actualmente cuando se ejecuta una acción para un conjunto de instancias seleccionadas. Sin embargo, una barra de progreso se puede representar de muy distintas formas, por lo tanto también se debería enriquecer el Modelo de Interacción Concreto con más primitivas con las que modelar el aspecto de las barras de progreso, tanto las que se mostrarán con las transacciones, como las que se mostrarán al lanzar un servicio sobre un conjunto de instancias seleccionadas. Este modelado se hace para cada una de las transacciones tanto locales como globales y para cada uno de los servicios, que mostrarán la barra de progreso cuando se lance sobre un conjunto de instancias ese mismo servicio (selección múltiple de instancias). Para cada transacción o servicio seleccionado, en el Modelo de Interacción Concreto se decide si mostrar la barra de progreso de forma gráfica, textual u ocultarla. Una vez elegido el tipo de visualización hay más detalles de configuración, tal y como se muestra a continuación:

- Mostrar barra de progreso gráficamente
  - Ubicación de la barra de progreso:
    - En algún margen de la pantalla principal
    - En un nuevo formulario
  - Mostrar porcentaje numérico de lo que ya se ha ejecutado
  - Desplazarse de izquierda a derecha o viceversa
- Mostrar barra de progreso indicando textualmente los servicios ya realizados
  - Texto de las lista de servicios:
    - Ubicación en la pantalla principal
    - Fuente del texto
    - Tamaño
    - Color
    - Alineación
  - El listado de las servicios que forman la transacción local se puede hacer de una de estas tres formas:
    - Se muestran todos los servicios y se resalta el que se esté ejecutando en el mismo momento
    - Se muestra la lista de servicios conforme se vayan ejecutando
    - Se muestra solo el servicio que se ejecute sin mostrar el resto de servicios que forman la transacción local
- Dar la posibilidad de ocultar la barra de progreso

Por defecto, si el analista no lo modela de otra forma, la barra de progreso se mostrará siempre gráficamente, en el margen de la pantalla principal, sin mostrar el porcentaje y desplazándose de izquierda a derecha.

7.1.4. Warning

En OO-Method, cada vez que un usuario quiere ejecutar un servicio (como un borrado, una modificación o cualquier otra acción sobre una instancia seleccionada en una UI de Población) se muestra una UI de Servicio. Esta UI de Servicio puede ser que se utilice para solicitar argumentos al usuario, pero otras veces no se solicita ningún argumento, porque su argumento es el propio objeto que invoca al servicio (THIS). En este último caso, las UI de Servicio se

pueden ver como un mecanismo de confirmación antes de ejecutar el servicio. Aun así sigue faltando una confirmación de ejecución si el servicio requiere la entrada de argumentos.

Además de los mensajes de confirmación antes de ejecutar un servicio, hay otros avisos que aun no soporta OO-Method. Son mensajes de aviso que se muestran al usuario dependiendo de cierta condición lógica sujeta a los requisitos funcionales del sistema. Algunas reglas de negocio recomiendan avisar al usuario antes de la ejecución de una acción cuando se satisface cierta condición. Este aviso tiene sentido cuando las acciones son irreversibles. Por ejemplo, en un sistema de facturación, el sistema debería avisar al usuario antes de emitir una factura con un valor de más de 10000 €, porque es una situación poco común y por tanto es posible que sea un error. La acción de emitir factura es irreversible, ya que se envía automáticamente por correo electrónico al cliente en cuanto se emite.

Mediante el patrón de Warning se consiguen solucionar los dos problemas. En el primer caso, si queremos que el mensaje se muestre siempre (confirmar la ejecución de un servicio), el analista solo debe poner una condición al mensaje que siempre sea cierta. En el segundo caso, el analista deberá definir la condición basándose en la lógica de negocio del sistema.

La mejora que aporta este patrón de usabilidad no puede ser incorporada directamente en el proceso de compilación de modelos. El analista debe configurar este patrón tanto en el Modelo de Objetos, como en el Modelo de Interacción Concreto:

- Modelo de Objetos: el analista debe elegir los siguientes campos:
  - El servicio sobre el que se aplica el patrón de usabilidad.
  - La condición que se debería satisfacer para mostrar el mensaje de aviso.
  - El texto que se mostrará al usuario cuando la condición se cumpla.

  Tal y como se puede apreciar, son los mismos campos que se necesitan para modelar una precondición en el Modelado Conceptual actual. El patrón de Warning puede verse como una precondición en la que se permite la ejecución del servicio.
- Modelo de Interacción Concreto: el analista podrá configurar los siguientes parámetros:
  - Si la ventana es bloqueante (modal) o no. Por ventana bloqueante se entiende aquella ventana que no permite hacer ninguna otra operación con el sistema hasta que no se cierre dicha ventana. La ventana no bloqueante es aquella que bloquea la ventana que la ha lanzado, pero el usuario puede trabajar con el resto de ventanas del sistema. Por defecto la ventana será no bloqueante.
  - El tipo de ventana, es decir, si es una ventana de alerta, información, error, etc. Por defecto será un mensaje de alerta.
  - Fuente del texto.
  - Tamaño.
  - Color.
  - Alineación.

Por defecto, el aviso se mostrará en una ventana modal de tipo alerta, con fuente Arial, tamaño 10, color negro y alineación centrada para los mensajes de aviso.

7.2. Modelado de las nuevas primitivas

Una vez definidas estas primitivas, hay que proporcionar a la herramienta OlivaNOVA de los mecanismos necesarios para poder incorporarlas a los Modelos Conceptuales. Esta sección muestra un prototipo de cómo OlivaNOVA implementaría el modelado de las nuevas primitivas creadas para incorporar la funcionalidad de los patrones de usabilidad. Es importante remarcar que estos patrones de usabilidad no solo afectan al Modelo de Interacción, sino también al Modelo de Objetos, por lo tanto habrá que hacer cambios en ambos.

7.2.1. System Status Feedback

Tal y como se ha comentado en la sección anterior, el modelado de este patrón afecta al Modelo de Objetos y al Modelo de Interacción Concreto.

En el Modelo de Objetos se definen los mensajes tanto de error como de confirmación de que se ha ejecutado correctamente el servicio. Aquellos servicios en los que no se defina ningún mensaje de este tipo se mostrará texto por defecto, creado por el Compilador de Modelos.

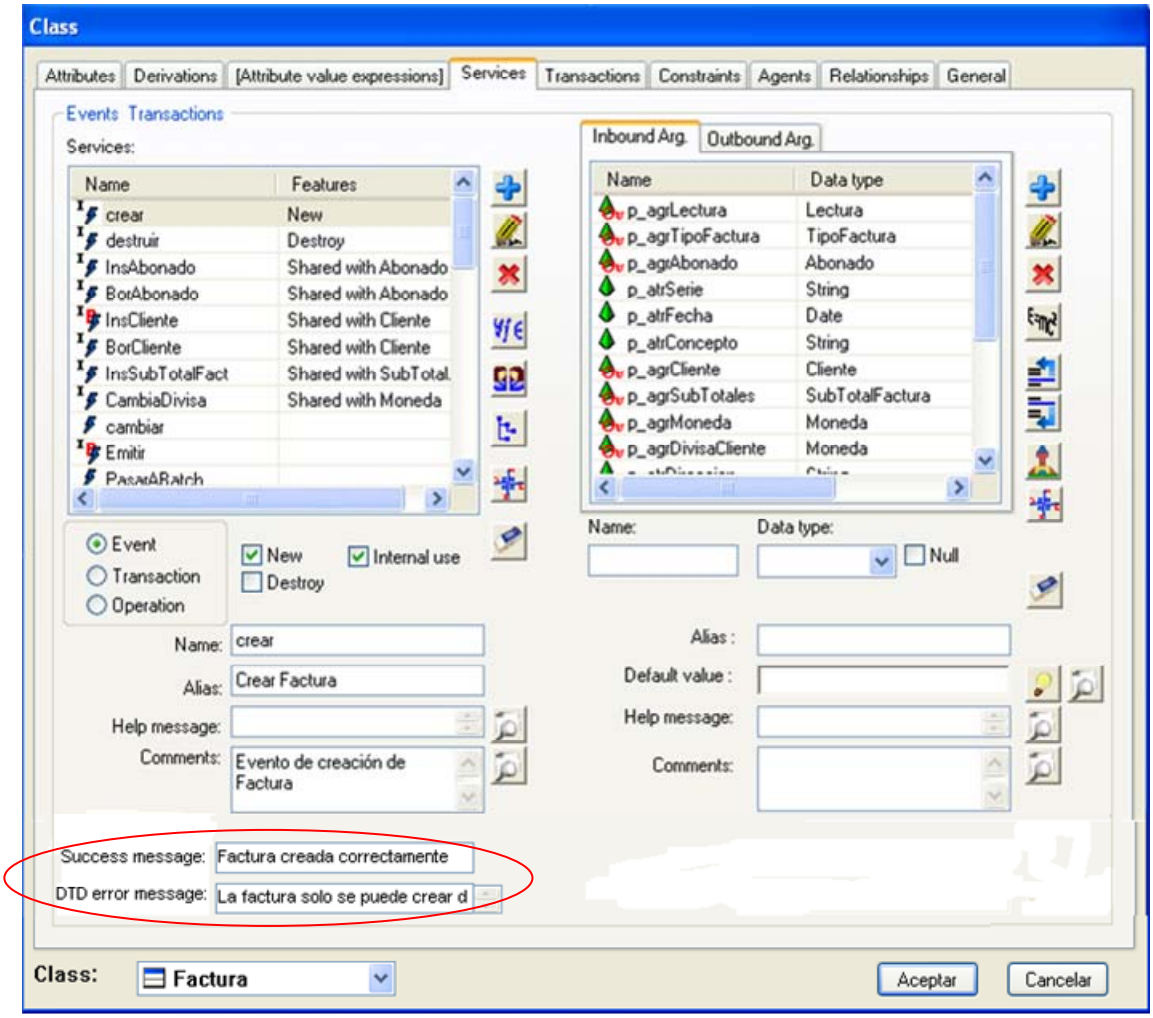

Figura 18. Prototipo de System Status Feedback en el Modelo de Objetos

La Figura 18 muestra la ventana desde la cual se modelan los servicios de la clase *Factura*. Resaltado en rojo se muestran los dos campos editables desde los cuales el analista puede introducir el texto que se mostrará al usuario. Este es el único cambio en el Modelo de Objetos que se debe aplicar al modelador. En el campo *Success message* se define el mensaje que se mostrará al usuario cuando el servicio se ejecute correctamente. En el campo *DTD error message* se define el mensaje de error que se mostrará al usuario cuando se solicite ejecutar

el servicio desde un estado del objeto en el cual no hay una transición definida para ese servicio.

En el Modelo de Objetos se especifica el mensaje que se mostrará en cada caso, en cambio es en el Modelo de Interacción Concreto a donde se define la forma en la que se mostrarán estos mensajes. En el Modelo de Interacción Concreto hay que hacer dos especificaciones: para los mensajes de éxito en la ejecución y para mensajes de error. Por lo tanto habrá dos ventanas distintas en el modelador, una para cada tipo de mensaje.

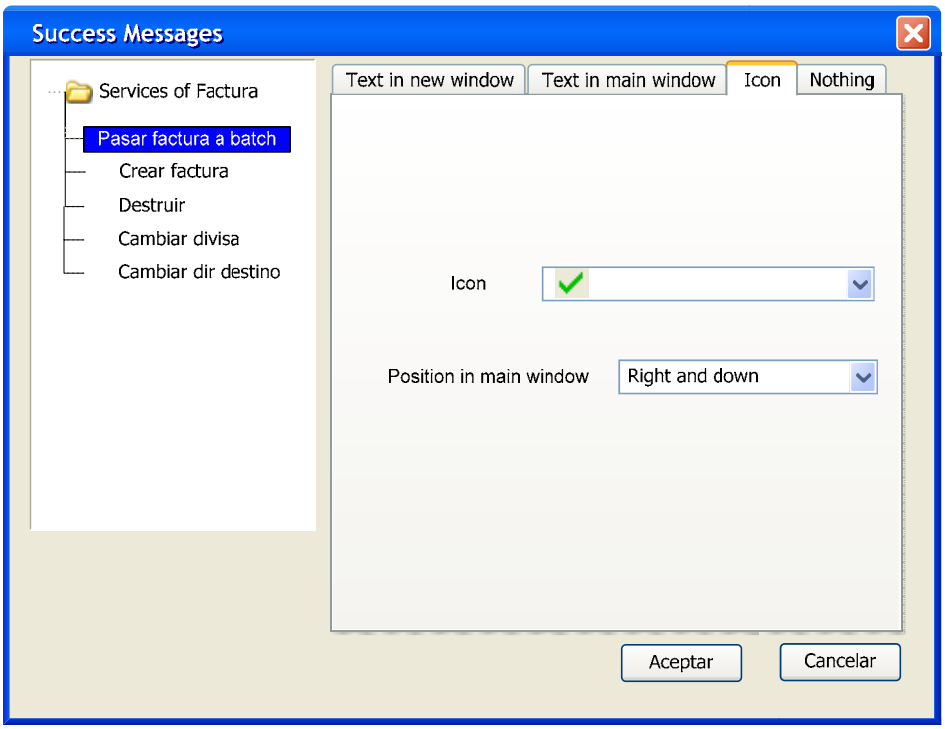

Figura 19. Prototipo de System Status Feedback en el Modelo de Interacción Concreto para servicios ejecutados correctamente

La Figura 19 muestra la ventana del modelador desde la cual se especifica cómo se mostrarán los mensajes de éxito en la ejecución de un servicio. Esta ventana se divide en cuatro secciones, según los cuatro tipos que existen para mostrar los mensajes: mostrar el texto mediante mensaje en una nueva ventana, mostrar el texto mediante mensaje en la ventana principal, utilizar iconos, o no mostrar ningún mensaje. Estos cuatro tipos son excluyentes entre sí, el analista solo puede seleccionar uno. En el ejemplo de la Figura 19, el analista ha decidido mostrar el mensaje mediante iconos en el margen derecho inferior de la pantalla principal.

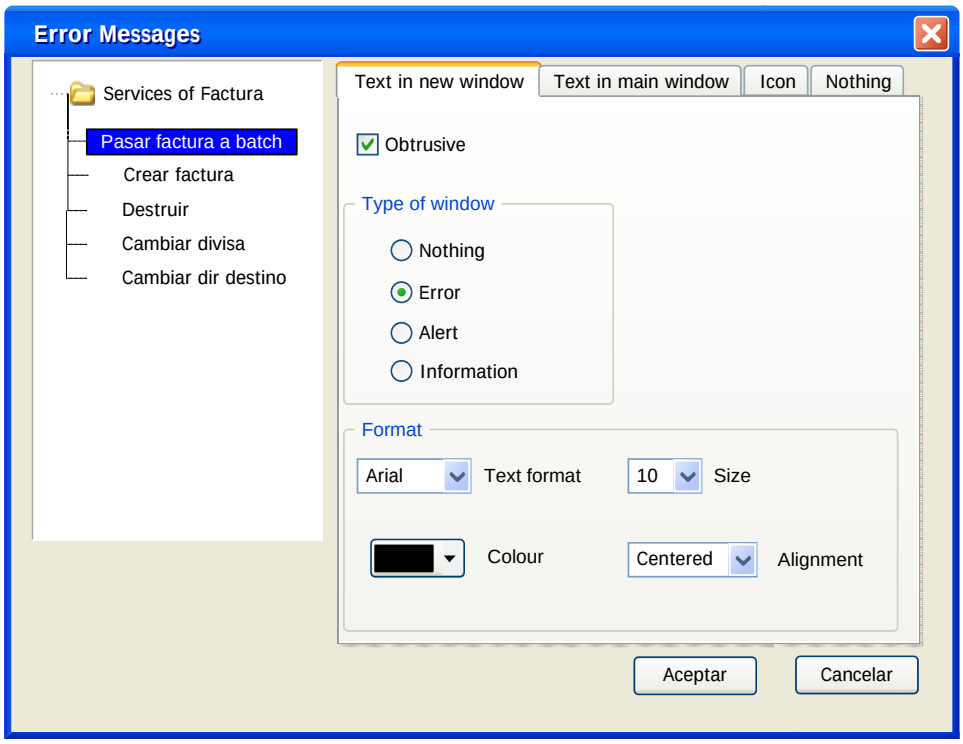

Figura 20. Prototipo de System Status Feedback en el Modelo de Interacción Concreto para un error en la ejecución del servicio

La Figura 20 muestra la pantalla del modelador desde la cual se define cómo se visualizarán los mensajes de error producidos en la ejecución de un servicio. La sintaxis de esta ventana es exactamente igual que la anterior, solo cambia la semántica (esta define la visualización de los mensajes de error). Así como en el Modelo de Objetos solo se especificaba el texto del mensaje de error producido por una transición no válida entre estados del objeto, en este Modelo de Interacción Concreto se especifica la forma en la que se mostrará cualquier tipo de mensaje de error: por incumplimiento de una precondición, de una restricción de integridad o por no existir una transición válida en el DTD. En los tres casos se mostrará el mensaje tal y como se defina en esta ventana. El texto en los tres casos sí que será diferente y vendrá dado por el Modelo de Objetos o el Compilador de Modelos.

En el ejemplo de la Figura 20 se indica que el mensaje se mostrará mediante texto en una nueva ventana, la ventana será modal, de tipo error, con fuente Arial, tamaño 10, color negro y alineación centrada.

7.2.2. Interaction Feedback

La incorporación de este patrón a OlivaNOVA no afecta para nada al modelador, ya que no lleva consigo primitivas de modelado asociadas. Todos los cambios provocados por este patrón solamente afectarán al Compilador de Modelos y al código generado, por lo tanto este patrón no introduce ningún cambio en OlivaNOVA.

7.2.3. Progress Feedback

Las primitivas de este patrón solo se modelan desde el Modelo de Interacción Concreto, por lo tanto OlivaNOVA solo debe modificar este modelo para incorporar la funcionalidad de dicho patrón. En el Modelo de Interacción Concreto especifica para cada transacción definida en el sistema, cómo se visualizará la barra de progreso.

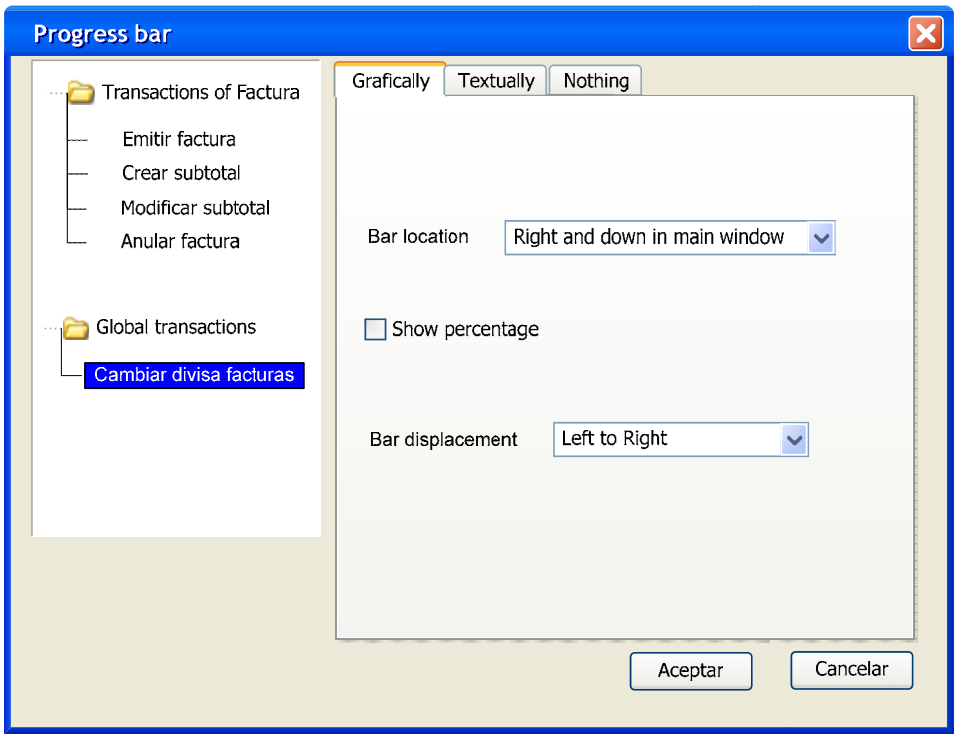

Figura 21. Prototipo de Progress Feedback en el Modelo de Interacción Concreto

La Figura 21 presenta la interfaz desde la que se modelará este patrón. En el árbol de la izquierda se pueden ver todas las transacciones del sistema agrupadas por la clase a la que pertenecen, además de las transacciones globales (no pertenecen a ninguna clase en concreto). Una vez definida la transacción sobre la que se define la forma de visualización, se elige cómo mostrar la barra de progreso: de forma gráfica, mediante texto o si no se muestra ninguna barra. Estas tres opciones son excluyentes entre sí.

En el ejemplo de la Figura 21 se ha definido una forma de visualización para la transacción global llamada *cambiar divisa facturas*. Esta barra de progreso será de forma gráfica, no mostrará el porcentaje de tiempo que resta para finalizar y el desplazamiento de la barra será de izquierda a derecha.

7.2.4. Warning

Este patrón afecta tanto al Modelo de Objetos como al Modelo de Interacción Concreto. En el Modelo de Objetos se define sobre qué servicio se va a definir el mensaje de aviso y la condición que se debe satisfacer para mostrarlo. Mientras que en el Modelo de Interacción Concreto se especifica cómo se mostrará este mensaje de aviso al usuario.

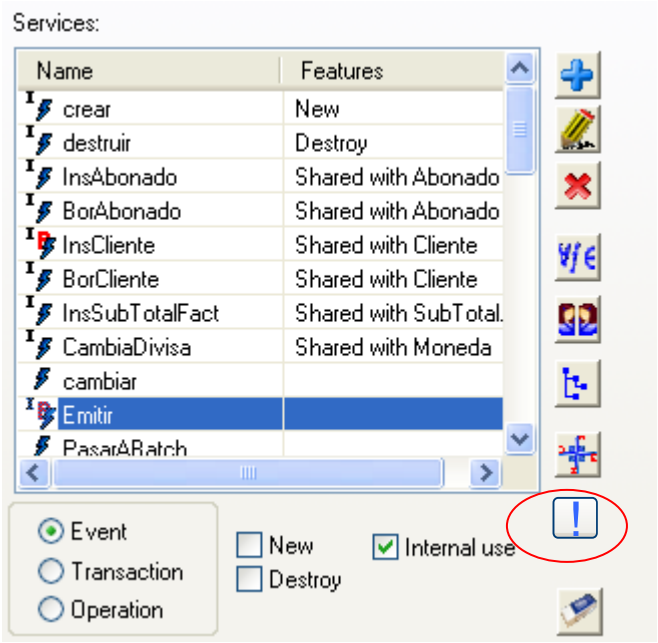

Figura 22. Prototipo de Warning en el Modelo de Objetos (1)

La Figura 22 muestra la interfaz de modelado de los servicios de las clases en OlivaNOVA. Para aplicar el patrón de Warning, el analista debe primero seleccionar de entre todos los servicios en cual va a definir el mensaje de aviso. Una vez seleccionado el servicio, solo debería pulsar el botón marcado en color rojo y se le mostraría una ventana desde la cual definir la condición para mostrar el aviso y el texto del mensaje, tal y como se muestra en la Figura 23.

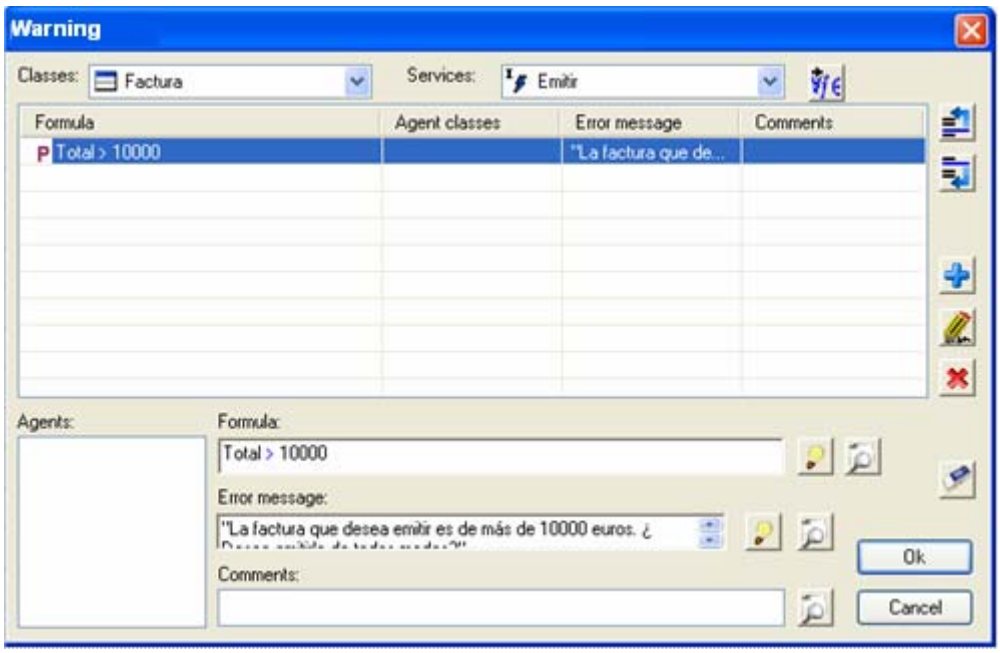

Figura 23. Prototipo de Warning en el Modelo de Objetos (2)

En el campo *Mensaje de error* de la Figura 23, el analista especifica el texto que se mostrará al usuario cuando la condición del aviso sea cierta. Esta condición se define en el campo *Fórmula*. Los componentes del modelo que pueden formar parte de la fórmula de la condición son:

- Atributos de las clases.
- Argumentos del servicio o transacción que forman la acción.
- Funciones de usuario definidas por el analista.
- Funciones estándar ya incluidas en ONME para trabajar con booleanos, numéricos, cadenas de caracteres y fechas.
- Operadores para trabajar con distintos tipos de datos como booleanos, números, cadenas de caracteres y fechas.
- Servicios visibles de clases.

Para construir esta fórmula se puede utilizar un asistente como el que aparece en la Figura 24. Este asiste evita los errores del analista en la definición de la fórmula.

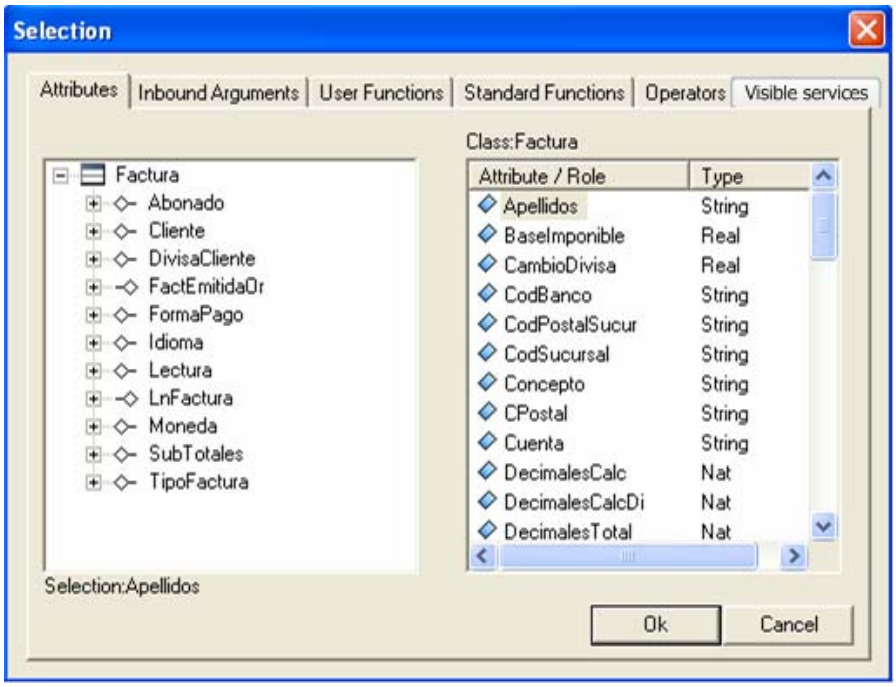

Figura 24. Asistente para la definición de la fórmula del patrón Warning

El otro modelo que se ve modificado por la incorporación del patrón Warning es el Modelo de Interacción Concreto. En este modelo se detalla la apariencia visual del mensaje de texto.

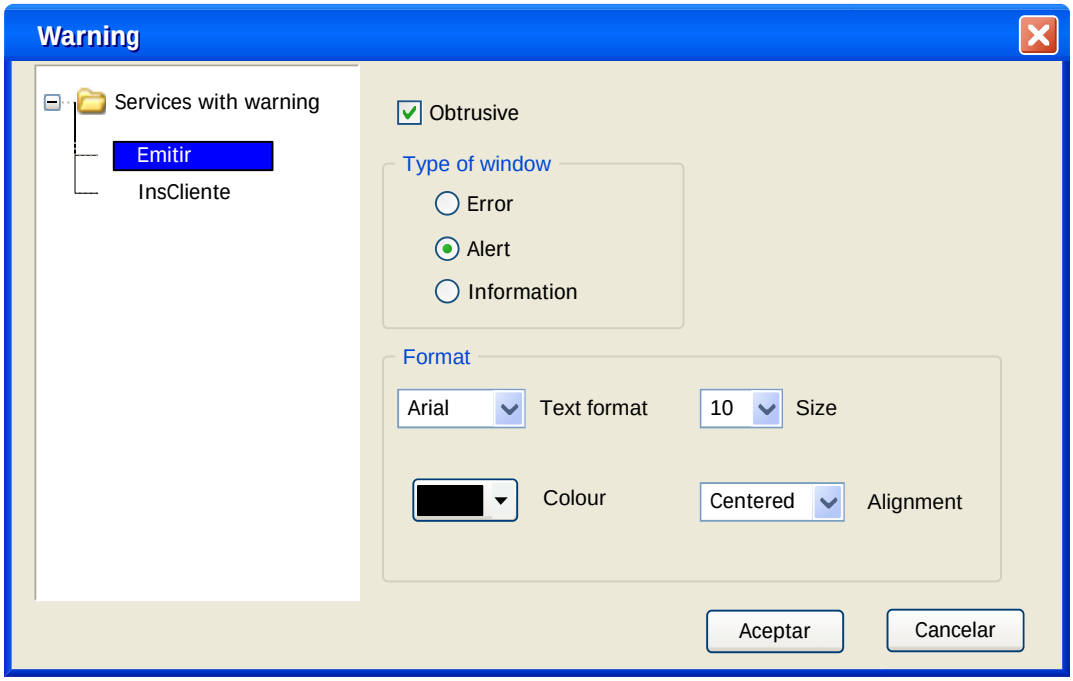

Figura 25. Prototipo de Warning en el Modelo de Interacción Concreto

En la Figura 25 se muestra cómo sería la interfaz desde la que se configura la visualización del mensaje de aviso. En el árbol de la izquierda se muestran, agrupados por clases, todos aquellos servicios que tengan una condición de aviso definida (se define en el Modelo de Objetos). El analista selecciona aquel servicio sobre el que desea configurar su visualización y a continuación pasa a especificarla.

En el ejemplo de la Figura 25 se ha definido sobre el servicio *Emitir* un mensaje de aviso que se mostrará en una ventana modal de tipo alerta, con fuente Arial, tamaño 10, color negro y alineación centrada para los mensajes de aviso.

8. CAMBIOS EN EL COMPILADOR DE MODELOS

Una vez definidos los cambios que debe implementar OO-Method para poder soportar la funcionalidad de los patrones de usabilidad de la familia Feedback, es necesario detallar los cambios a aplicar en el Compilador de Modelos y en el Código Específico. Para que la funcionalidad de las nuevas primitivas de modelado incorporadas a nivel de modelado se refleje en la aplicación final, es necesario enriquecer el Compilador de Modelos con la suficiente expresividad como para implementar el código necesario para la funcionalidad de los patrones.

La Figura 26 muestra el proceso de generación OO-Method resaltando las fases que se ven afectadas a la hora de dotar a OO-Method de la capacidad de implementar las primitivas conceptuales de los patrones. Se puede observar que dichas fases se corresponden en MDA con las fases *Platform Specific Model* y *Code Model*

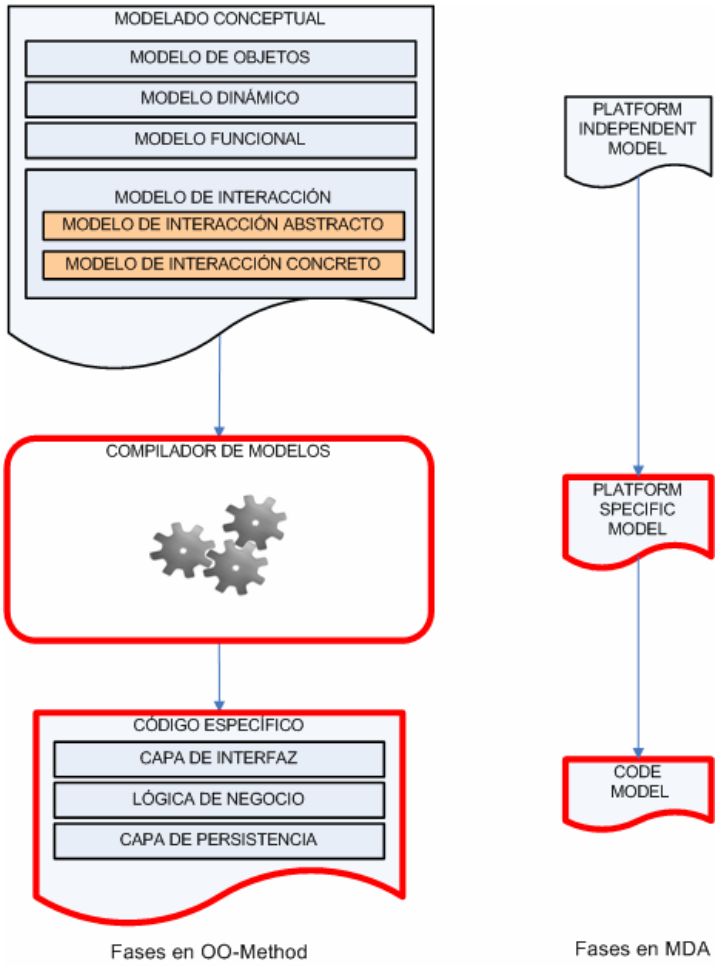

Figura 26. Fases de OO-Method y MDA afectadas por los cambios en la implementación de la funcionalidad de los patrones de usabilidad

Para explicar dichos cambios, el estudio se ha basado en el *Código Genérico* creado por el Compilador de Modelos. El Código Genérico es una abstracción del Código Específico que OO-Method genera para resolver un problema particular. Del Código Específico se pueden generalizar clases que tendrán la misma estructura en todos los sistemas generados. Es decir, sus relaciones, su razón de ser y algunos de sus métodos y atributos serán los mismos en todos ellos, solo cambiará su lógica de negocio dependiendo del problema concreto para las que fueron implementadas. El Código Genérico está formado por las clases que genera el

Compilador de Modelos para un lenguaje concreto de programación sin importar la funcionalidad concreta de cada clase, sino cual es su significado dentro de cualquier aplicación. Por lo tanto las clases del Código Genérico solo tendrán métodos que se utilicen en todos los sistemas. Los métodos que implementen una lógica específica de un problema concreto no formarán parte del Código Genérico.

Para representar cómo se incorpora la funcionalidad de los patrones de usabilidad en el Código Genérico, se ha utilizado una notación basada en Diagramas de Clase y de Secuencia. Estos diagramas están formados por las clases del Código Genérico ya generadas actualmente por OlivaNOVA y las que harían falta para implementar los patrones. Por un lado, el Diagrama de Clases se utiliza para representar los métodos, atributos y relaciones entre clases necesarias para la incorporación de los patrones. Por otro lado, el Diagrama de Secuencia muestra la secuencia ordenada de invocación de los métodos de las clases que se utilizan en la funcionalidad del patrón.

8.1. Clases generadas por OlivaNOVA

El primer paso para conocer los cambios que se deben aplicar al Compilador de Modelos, es conocer las clases en Código Genérico que genera actualmente. En esta sección se presentan las clases generadas por el Compilador de Modelos de OlivaNOVA. Está centrada en aquellas clases que se ven modificadas al incorporar la funcionalidad de los patrones correspondientes a la familia Feedback. El objetivo es por tanto explicar la funcionalidad de los métodos de estas clases antes de aplicar ningún cambio para incorporar dichos patrones. Así será más sencillo entender los cambios necesarios para implementar su funcionalidad.

Las clases del lenguaje de programación generado por OlivaNOVA van a estar agrupadas en dos tipos, las clases que se ejecutarán en el cliente y las clases que se ejecutarán en el servidor:

- Cliente: están todas aquellas clases que se encargan de generar la interfaz del usuario y las peticiones para ejecutar las acciones.
- Servidor: están todas aquellas clases que implementan la lógica de negocio y el acceso a la base de datos.

Esta separación hace posible que un mismo servidor pueda aceptar las peticiones de distintos clientes y que servidor y cliente estén en distintas máquinas conectadas por red.

8.1.1. Clases de la parte servidora

8.1.1.1. Class X action

Esta clase contiene los servicios que implementan la lógica de negocio de la *clase X,* donde X es el nombre de la clase creada en el Modelo de Objetos. Habrá un método por cada uno de los servicios que se hayan modelado en esta clase en la fase de Modelado Conceptual, dentro del Modelo de Objetos. La lógica de negocio de estos métodos se obtiene del Modelo Funcional. Además, la clase *Class X action* también contiene un apartado donde se implementan todas las restricciones de integridad que se hayan definido sobre atributos de esta clase y otra sección para comprobar si se cumplen las precondiciones asociadas a los servicios de la clase, ambas definidas en el Modelo de Objetos.

8.1.2. Clases de la parte cliente

8.1.2.1. Service wrapper

Esta clase alojada en el cliente, se encarga de hacer de intermediario entre la parte cliente y la servidora. Cliente y servidor se comunican a través de esta clase. Desde ella se hace la

invocación a los métodos que implementan la lógica del negocio en las clases del servidor, llamadas *Class X action,* donde X es el nombre con el que se haya definido la clase en el Modelo de Objetos. La clase *Service Wrapper* tiene un método por cada uno de los servicios que se hayan definido en el Modelo de Objetos del Modelado Conceptual. Cada uno de los métodos definidos en la clase *Service Wrapper* hace una invocación al método correspondiente en la clase *Class X action* que contiene la lógica del servicio definida en el Modelo Funcional.

8.1.2.2. Form X

Este nombre representa una clase en el lenguaje de programación que representa un formulario, es decir, es una clase que generará una ventana de la interfaz. Mediante este formulario, el usuario introduce el valor de los argumentos que forman el servicio que se va a ejecutar. Existe una clase *Form X* por cada una de las Unidades de Interacción de Servicio que se hayan definido en el Modelado de Interacción, donde X es el nombre del servicio definido en el Modelo de Objetos y que se ejecuta desde este formulario. Todos estos formularios tienen asociados dos botones, uno para ejecutar la acción (OK) y otro para cancelarla (Cancel). Los métodos que capturan el evento de pulsar el botón *OK* son los encargados de hacer la invocación a los métodos definidos en la clase *Service wrapper* que se encargarán de hacer las llamadas necesarias para ejecutar los servicios en la parte del servidor.

8.2. Representación abstracta de los cambios

Esta sección muestra de forma abstracta el código que debe generar el Compilador de Modelos. De esta forma, se pueden estudiar los cambios a incorporar en la estrategia de generación de código por parte del traductor de OlivaNOVA. Esta representación abstracta está formada por dos diagramas:

- **Diagrama de secuencia:** muestra gráficamente la interacción entre las clases que implementan el patrón de usabilidad y las clases genéricas del sistema software. Este diagrama es un diagrama genérico, es decir, es una abstracción del código, válida para todos los sistemas software generados con OlivaNOVA. Cada sistema generado con OlivaNOVA será una instanciación de estas clases genéricas. Tanto en este diagrama como en el de clases, se han representando las clases nuevas que implementan la funcionalidad de los patrones de usabilidad en gris. Las clases que ya generaba el compilador de modelos antes de añadir los patrones de usabilidad y han sido ampliadas con nuevos atributos y servicios, se resaltan con trama cruzada. Las clases que no sufren ninguna modificación por la incorporación de los patrones de usabilidad, se mantienen en blanco.
- **Diagrama de clases:** las clases que implementan la funcionalidad de los patrones de usabilidad se añaden al diagrama de clases formado a partir del Código Genérico.

A continuación se presentarán ambos diagramas para cada uno de los patrones de la Familia Feedback.

8.2.1. System Status Feeback

Este patrón aportaba la funcionalidad de avisar al usuario sobre el éxito o el fracaso en la ejecución de un servicio. Esta funcionalidad está representada en Diagrama de Secuencia de la Figura 27. Si el usuario ejecuta un servicio mediante la invocación a *OK_action*, la clase *Service wrapper* lanza la petición a la clase encargada de ejecutar el servicio. Esta clase se llama *ClassX action* y se encuentra en la parte servidora. Una vez que el servicio se ha ejecutado, el usuario puede ver su resultado gracias a la clase *Alert manager* que muestra un mensaje indicando el éxito o el fracaso en la ejecución del servicio.

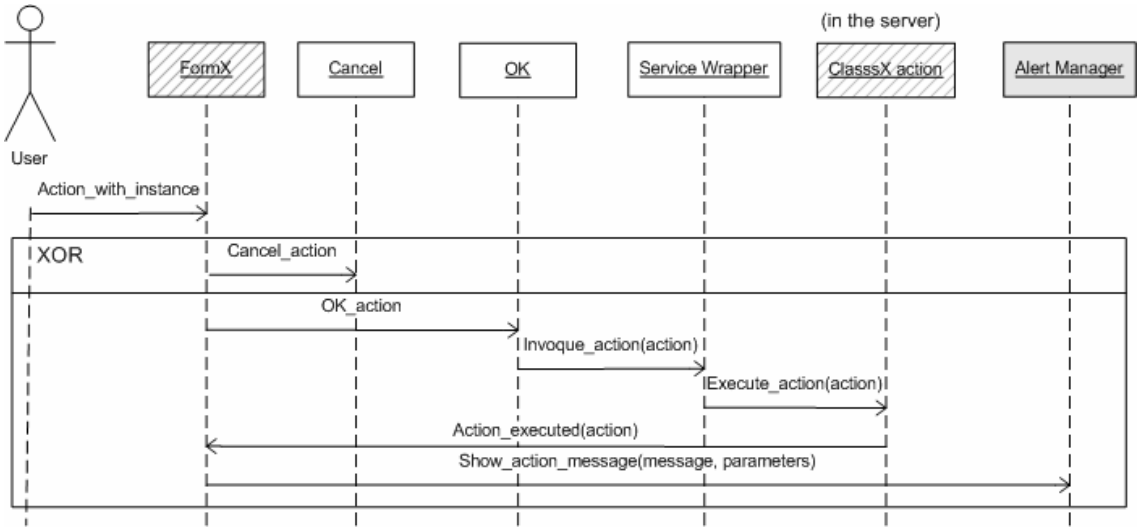

Figura 27. Diagrama de secuencia para System Status Feedback

La Figura 28 muestra las clases y métodos que implementan esta funcionalidad. Algunas clases ya existen y solo tienen que incluir nuevos métodos; este es el caso por ejemplo de *Class X action* y *FormX*. Otras clases han sido definidas desde cero, por ejemplo la clase *Alert manager*. Las clases *Service wrapper, Cancel* y *OK* no han sido modificadas. La clase *Service wrapper* contiene el servicio *Invoque_action* que inicia la ejecución de la acción. Esta clase usa el servicio *Execute_action* de la clase *ClassX action* ubicada en el servidor, donde está el código de la acción. El éxito o el fallo de la acción se mostrará al usuario usando la clase *Alert manager*. El servicio *Execute_action* de la clase *ClassX action* debe ser modificado para que informe mediante el servicio *Show_action_message* sobre el éxito o fracaso de la acción.

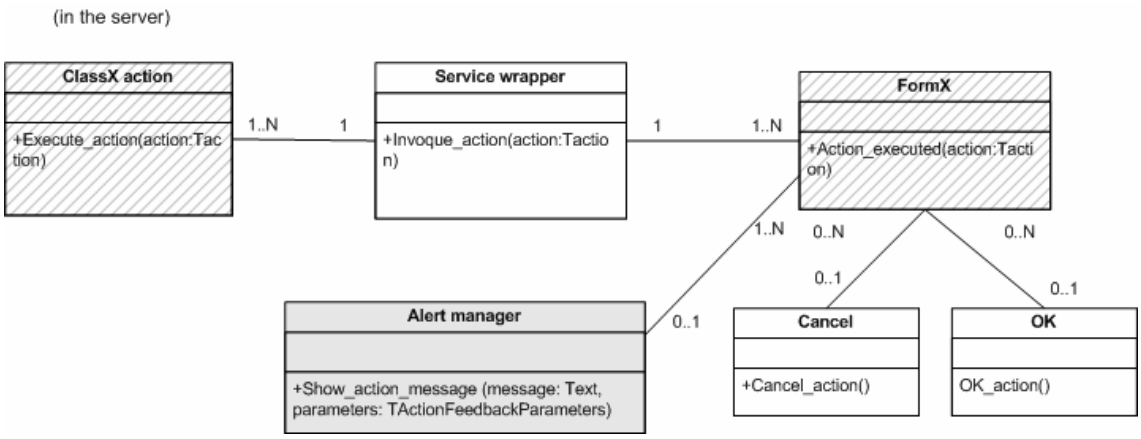

Figura 28. Diagrama de clases de System Status Feedback

Todas las posibilidades de configuración del patrón modeladas desde el Modelo de Interacción Concreto están representadas en el Diagrama de Clases de la Figura 28 en el argumento *parámetros* del método *Show_ action_message*. Este atributo es una tupla con tantos campos como parámetros pueda configurar el analista en la fase de Modelado de la Interfaz Concreta.

8.2.2. Interaction Feedback

La funcionalidad de este patrón aplicado a OO-Method consistía en deshabilitar los componentes de la interfaz mientras duraba la ejecución de la acción. Se deshabilitaban todos

los componentes excepto el botón desde el que se podía cancelar la ejecución del servicio una vez iniciado.

La Figura 29 muestra el diagrama de secuencia para esta funcionalidad de Interaction Feedback. El usuario inicia la ejecución del servicio pulsando sobre el botón *OK* de la ventana. Antes de solicitar la ejecución del servicio, la clase *Form X* deshabilita sus botones con el servicio *Disable_actions_and_widgets*. Posteriormente la clase *OK* inicia la ejecución del servicio invocando al método *Invoque_action* de la clase *Service_wrapper*, la cual invocará mediante el método *Execute_action* a la clase del servidor que tenga la lógica asociada a la ejecución del servicio. Una vez finalice la ejecución, la clase *Form X* invocará al método *Enable_action_and_widgets* para volver a habilitar los componentes de la ventana.

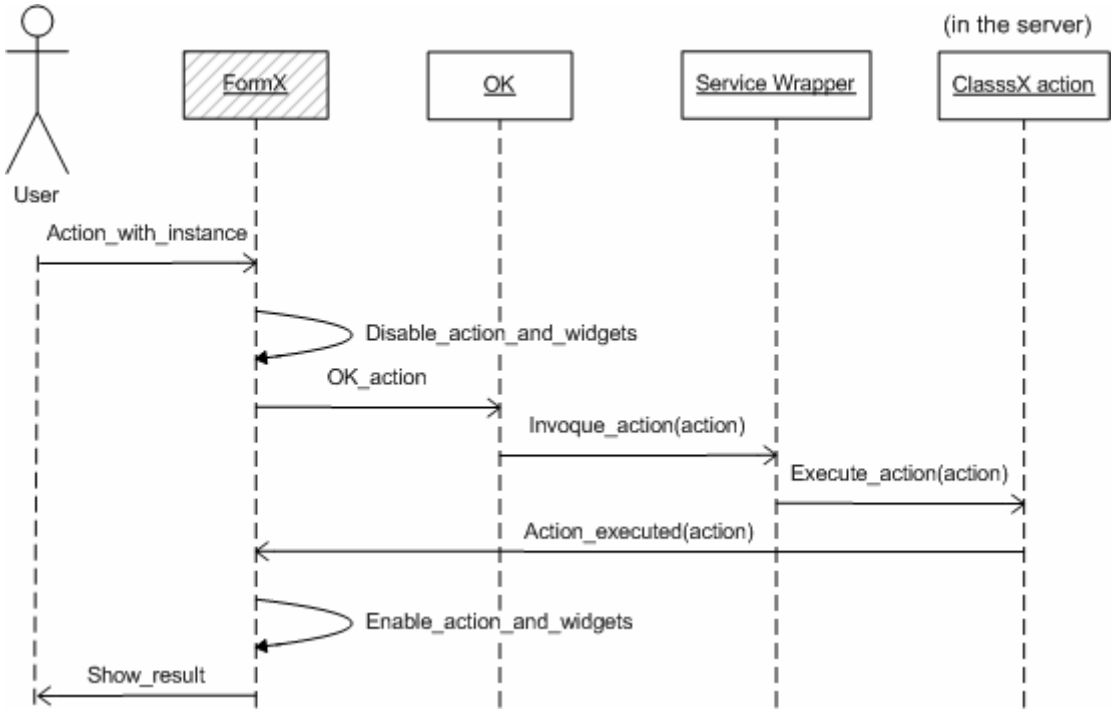

Figura 29. Diagrama de secuencia para Interaction Feedback

Las clases y métodos utilizados para implementar esta funcionalidad, se ven reflejados en el Diagrama de Clases de la Figura 30. En él se puede apreciar que los únicos cambios que se han introducido para incorporar este patrón han afectado a la clase FormX. A esta clase se le han añadido dos servicios, uno para deshabilitar los botones y los widgets de la ventana, llamado *Disable_action_and_widwets*; y otro para habilitarlos llamado *Enable_actions_and_widgets*.

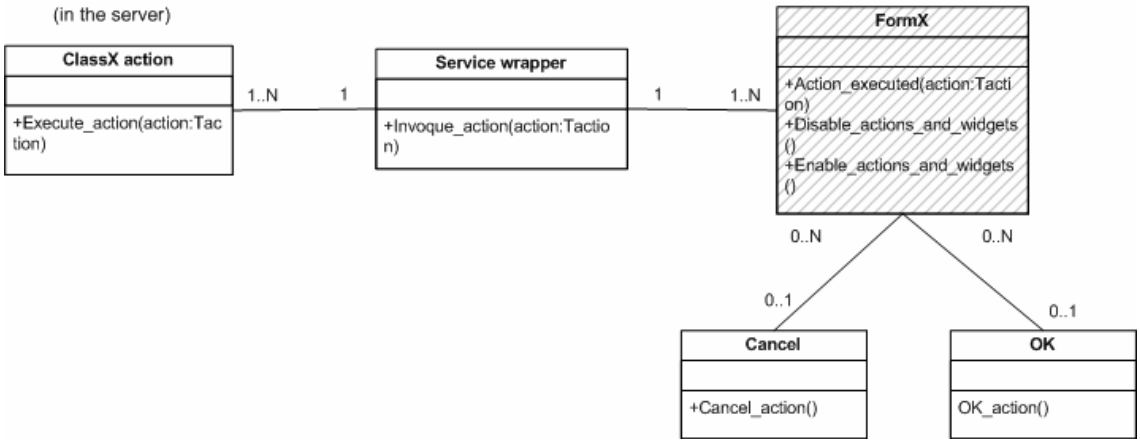

Figura 30. Diagrama de clases de Interaction Feedback

8.2.3. Progress Feedback

Este patrón aporta la funcionalidad de mostrar una barra de progreso mientras se ejecuta una transacción local o global. La barra de progreso debe indicar qué porcentaje de tareas de las que forman la transacción se han realizado en cada momento.

La Figura 31 muestra el Diagrama de Secuencia para el caso en que el patrón *Progress Feedback* se utilice en una transacción global. Esta transacción global se inicia mediante el método *OK_action,* cuando el usuario pulse el botón OK en una UI de Servicio. Después, se entra en un ciclo iterativo, donde por cada instancia se ejecutan todos los servicios asociados a la transacción global. Mediante el método *Invoque_action¸* se invoca a *Execute_action* que ejecuta el servicio en el servidor. Cada vez que se ejecutan todos los servicios para cada una de las instancias, se incrementa la barra de progreso en uno, usando para ello el método *Increment_ProgressBar*. Al ejecutarse todos los servicios para todas las instancias implicadas, se cierra la ventana de la barra de progreso usando para ello el método *Close_Form*.

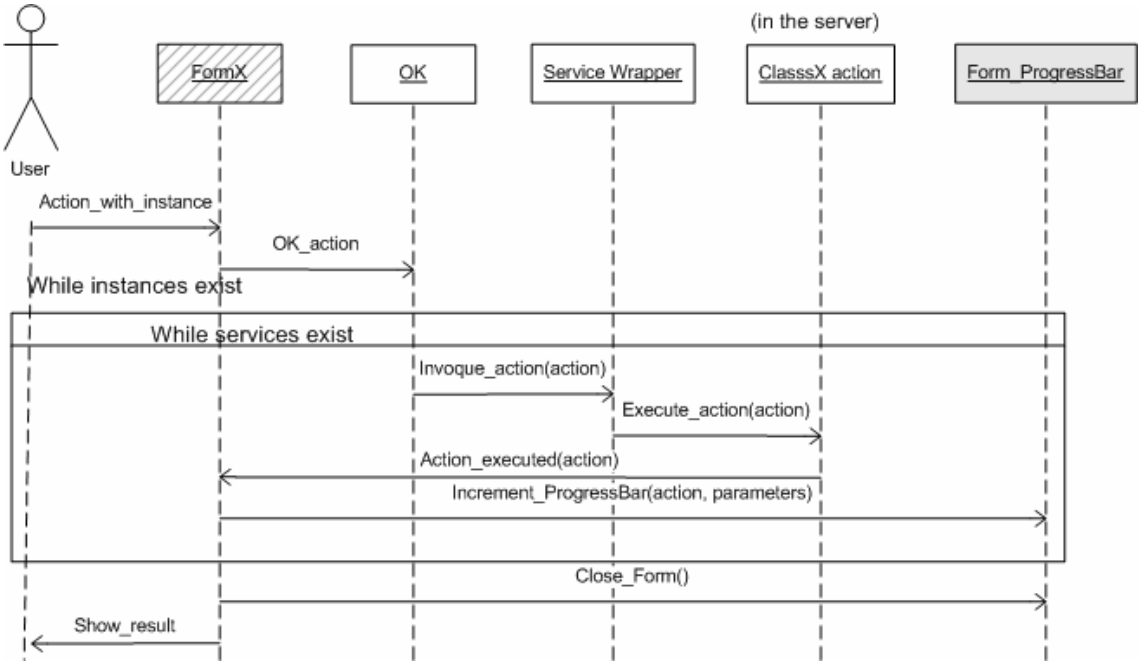

Figura 31. Diagrama de secuencia para Progress Feedback en una transacción global

La otra funcionalidad del patrón *Progress Feedback* sobre OO-Method es la incorporación de un mecanismo que informe al usuario de los servicios que se van realizando en una transacción local. Esta funcionalidad está representada en el Diagrama de Secuencia de laFigura 32. Se diferencia del anterior en que solo contiene una iteración. En una misma iteración se recorren todos los servicios que componen la transacción local. Para cada acción se invoca el método *Invoque_action* de la clase *Service wrapper* que se encargará de hacer la petición de ejecución del servicio en el servidor. Esta petición se hace mediante el método *Execute_action* de la clase del servidor *ClassX action*. Al ejecutarse el servicio, se informa a la clase *FormX* de que se ha terminado la ejecución. De esta forma, la clase *FormX* mediante el método *Increment_progress_bar* indicará que se debe informar al usuario mediante la clase *Form ProgressBar* que el servicio de la transacción local ya se ha ejecutado. Una vez se hayan ejecutado todos los servicios que componen la transacción local, se cierra la ventana con la que se informa al usuario sobre su progreso.

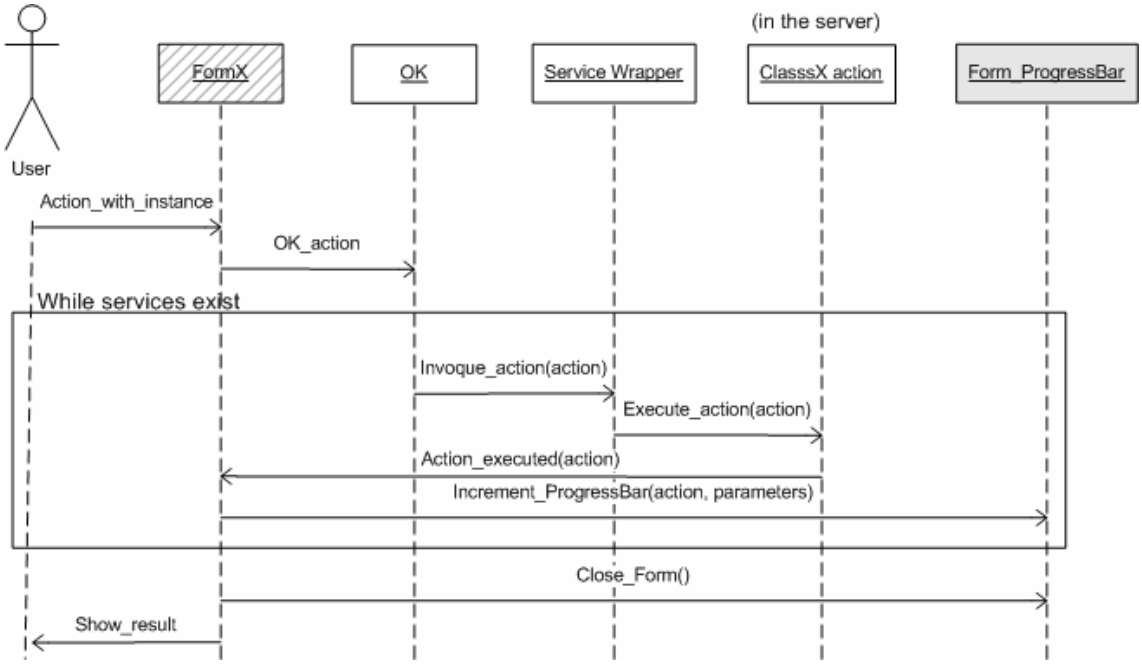

Figura 32. Diagrama de secuencia para Progress Feedback en una transacción local

El Modelo de clases de la Figura 33 muestra las clases presentadas en los Diagramas de Secuencia anteriores (tanto transacciones globales como locales) que este patrón de usabilidad utiliza para su ejecución. La única clase que se debe implementar desde cero es *Form ProgressBar* Esta clase contiene un método llamado *Increment_progressBar* para actualizar el valor de la barra de progreso. Las nuevas líneas de código que invocarán a la clase *Form_ProgressBar* se implementará en la clase *FormX*. Esta invocación se hará desde el nuevo método *Action_executed*.

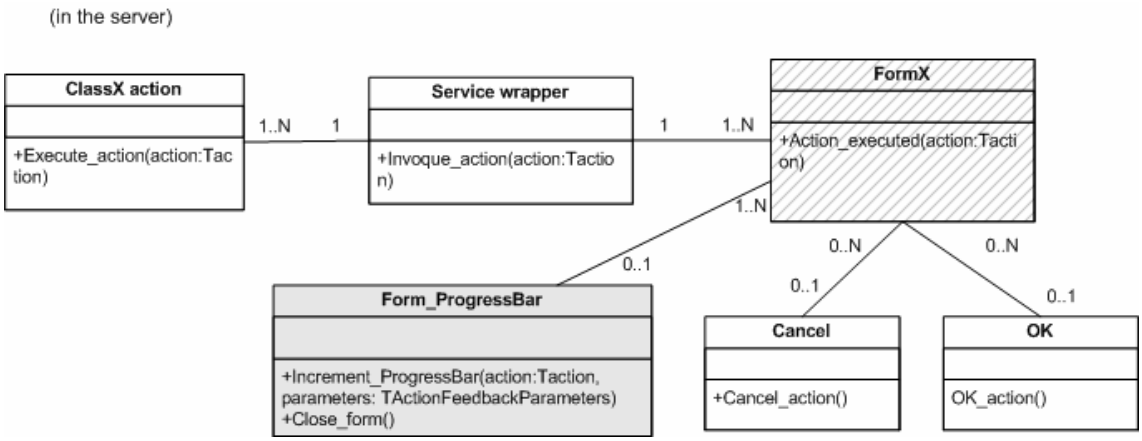

Figura 33. Diagrama de clases para *Progress Feedback*

Todas estas las posibilidades de modelado en el Modelo de Interacción Concreto están representadas en el Diagrama de Clases de la Figura 28 mediante el argumento *parámetros* del método *Increment_Progress_bar*. Este atributo es una tupla con tantos campos como parámetros pueda configurar el analista en la fase de Modelado de la Interfaz Concreta.

8.2.4. Warning

La funcionalidad de este patrón aplicada a OO-Method permite al analista definir mensajes de texto que se mostrarán al usuario cuando se cumpla cierta condición. Una vez mostrado el mensaje de aviso al usuario, éste puede decidir si permite la ejecución del servicio o lo cancela.

En la Figura 34 se puede apreciar la secuencia de acciones para implementar esta funcionalidad. La secuencia de pasos mostrada representa el caso en que la condición asociada al mensaje de aviso sea cierta y se deba mostrar el mensaje. Cuando un usuario quiere ejecutar un servicio, la clase *Service wrapper* recibe la petición. Esta clase envía la petición a la clase *ClassX action* implementada en el servidor. Esta clase se encarga de verificar si existe algún mensaje de aviso asociado con el servicio y en este caso, comprueba si la condición se satisface. Cuando la condición se satisface, la clase *Alert manager* pregunta al usuario si quiere ejecutar el servicio o no. Si el usuario decide ejecutar el servicio a pesar de la advertencia, la clase *ClassX action* se encarga de ejecutarla. Si no quiere ejecutar el servicio, se termina el proceso.

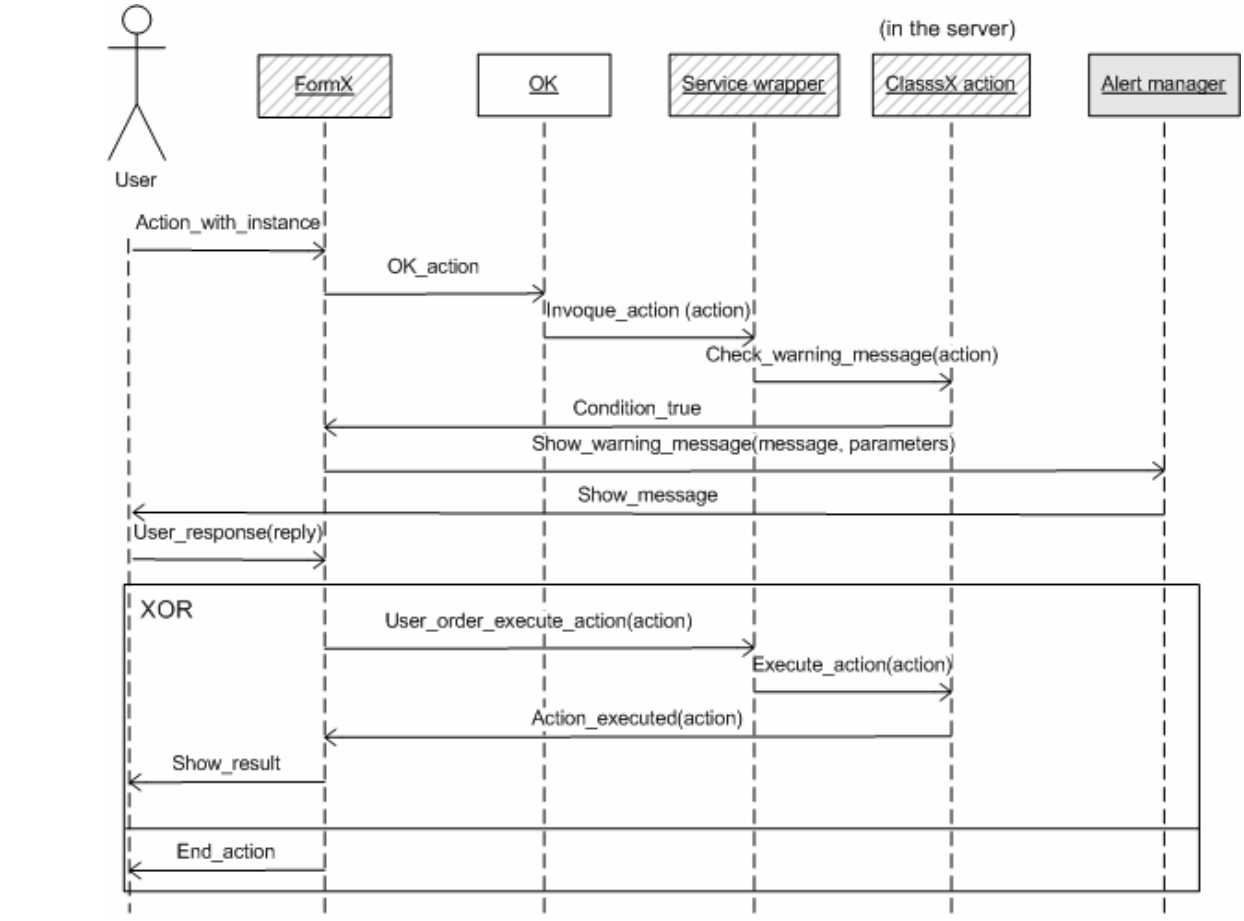

Figura 34. Diagrama de secuencia para Warning

Tal y como se muestra en la Figura 35, la funcionalidad principal de este patrón de usabilidad se implementa en la clase *ClassX action* con el método *Check_warning_message.* Este método comprueba si el servicio tiene asociado una condición y si esta condición se satisface. En caso de que se satisfaga, se muestra un mensaje al usuario invocando al método *Show_warning_message* de la clase *Alert manager*. El usuario debe decidir si quiere ejecutar el servicio o no utilizando el mismo mensaje de aviso que se le muestra. Si finalmente desea ejecutar el servicio a pesar del mensaje de aviso, el método *User_response* inicia su ejecución. Los nuevos métodos en clases ya existentes son los de *Check_warning_message* de la clase *ClassX action, Condition_true* de la clase *FormX* y *User_order_execute_action* de la clase *Service wrapper.* El resto de métodos ya existían o pertenecen a clases nuevas. La única clase nueva es la llamada *Alert manager*.

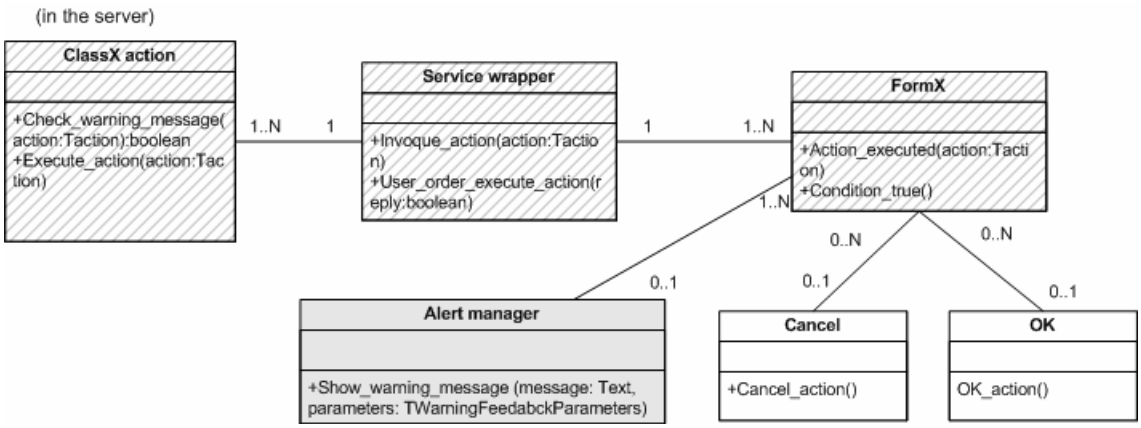

Figura 35. Diagrama de clases para *Warning*

Al igual que para el caso de los patrones anteriores, todas las posibilidades de configuración del patrón en el Modelo de Interacción Concreto están representadas en el Diagrama de Clases de la Figura 35 en el argumento *parámetros* del método *Show_warning_message*. El argumento *parámetros* es una tupla donde cada uno de sus campos representará uno de los parámetros que el analista haya definido en la parte de Modelado de la Interfaz Concreta.

8.3. Cambios en la estrategia de generación de código

En este apartado se detallan los cambios que se deben aplicar sobre las clases generadas con el actual Compilador de Modelos para incorporar la funcionalidad de los patrones de la familia Feedback. Además de modificar las clases existentes, habrá que añadir otras clases nuevas en algunos casos. En esta sección se detallan cada uno de los cambios que habría que incorporar al generador de código para contemplar cada una de las funcionalidades de los patrones de usabilidad vistos. Una vez incorporados todos estos cambios en el Compilador de Modelos, el código que soporta la funcionalidad de los patrones de usabilidad se generará de manera automática junto con el resto del sistema.

En los casos que se presentan a continuación, se ha utilizado el lenguaje C#, pero la funcionalidad de cada uno de estos patrones se puede extrapolar a cualquier lenguaje de programación. La elección del lenguaje C# ha sido porque es uno de los lenguajes que ya genera actualmente el Compilador de Modelos. Sin embargo, una vez incorporada la funcionalidad de los patrones de usabilidad, se generaría su código en C#, en Java , en ASP .NET y en JSP.

Los cambios expuestos en esta sección se incorporarán a la estrategia de generación de código de OlivaNOVA, de forma que se aplicará de la misma forma en cualquier sistema generado. Sin embargo, hay ciertos aspectos que serán específicos de cada solución, como nombres de clases y métodos. Con el fin de facilitar la comprensión de los cambios, se han adaptado los cambios propuestos para la estrategia de generación de código a un ejemplo concreto, ya que la notación en Código Genérico sería muy confusa para el lector. Por lo tanto, el código que se muestra como generado por el Compilador de Modelos será Código Específico. A partir de este trabajo, es muy sencillo extraer las reglas que se deben incorporar en la estrategia de generación de código del Compilador de Modelos para que genere el Código Específico correspondiente a los patrones de usabilidad.

Como ejemplo de Código Específico se ha elegido el de la aplicación llamada Aguas de Bullent, que se utiliza en la realidad y está actualmente en producción. Su utilidad es la de gestionar los contadores de agua de una compañía de reparto de agua potable. Las principales funciones del sistema son la de mantener el listado de contadores, de clientes y de materiales en el almacén. También dispone de toda la funcionalidad necesaria para llevar la gestión de las facturas de los clientes y gestión de los distintos proveedores. El caso de ejemplo va a estar centrado en la gestión de las facturas.

El modelado de las ventanas utilizadas para gestionar las facturas se ha hecho en el Modelo de Interacción [26] de OlivaNOVA. Tal y como se ha comentado en la sección 3.1, los elementos de este modelo se dividen en tres niveles, ofreciendo una descripción abstracta de la interfaz del sistema. Los elementos del Modelo de Interacción actual de OlivaNOVA que se han utilizado para construir las ventanas del caso de estudio son:

- Un Árbol de Jerarquía de Acciones (AJA) del Modelo de Interacción. El AJA forma el primer nivel del Modelo de Interacción y dará lugar al menú de la aplicación.
- Los elementos de segundo nivel se llaman Unidades de Interacción (UI). Para el caso de estudio se ha usado una Unidad de Interacción de Población para representar el listado de facturas. A esta Unidad de Interacción se le han asociado Patrones Elementales para precisar su comportamiento. Los patrones elementales forman el

tercer nivel del Modelo de Interacción. Los patrones de presentación Elementales que se han añadido son:

- Un Filtro para filtrar las facturas entre otras cosas por fecha, cliente o número.
- Un Criterio de Ordenación, para ordenar el listado de facturas por fecha.
- Navegaciones hacia Líneas de Factura, Facturas emitidas y SubTotales.
- Conjunto de Visualización formado por los atributos de las instancias listadas que serán visibles por el usuario.
- Acciones que el usuario podrá realizar sobre las instancias de factura listadas. Un ejemplo de acción es *Emitir factura*, donde el usuario introduce como argumentos la factura y la fecha de emisión de la misma.

- Una Unidad de Interacción de Servicio por cada una de las acciones que se puedan ejecutar sobre una instancia de la factura. Mediante una Unidad de Interacción de Servicio se modelan los diálogos para que el usuario lance un servicio sobre una instancia de factura.

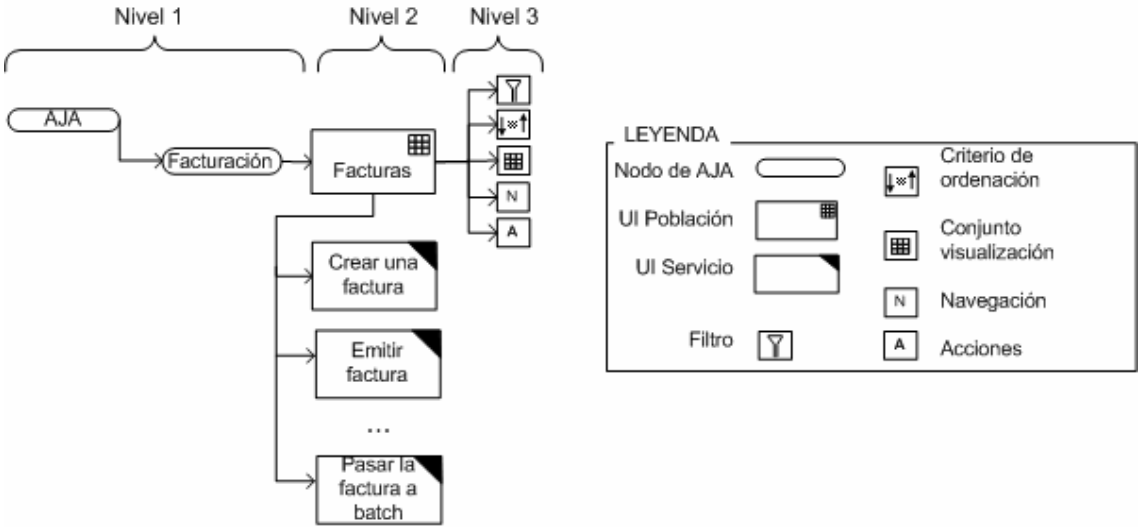

Figura 36. Modelo de Interacción para la gestión de Facturas

La Figura 36 muestra un extracto del Modelo de Interacción que representa las interfaces de la parte de la gestión de facturas de la aplicación Aguas de Bullent. Este modelo se ha realizado con las primitivas que existen actualmente en el Modelo de Interacción de OlivaNOVA. En la figura se hace una distinción clara del nivel al que pertenece cada una de las primitivas.

A continuación, para cada uno de los patrones de la familia Feedback, se presentará el código que soporta su implementación. A partir de este código se puede modificar la estrategia de generación de código del Compilador de Modelos para generar este código de forma automática.

8.3.1. System Status Feedback

Este patrón incorpora la funcionalidad de avisar al usuario sobre el éxito o el fracaso en la ejecución del servicio. Esta funcionalidad se implementa modificando la clase *Form X* actualmente ya generada por el Compilador de Modelos. Como ejemplo del caso de estudio, se ha elegido el servicio *Pasar factura a Batch* explicada en el punto tres. Así, para nuestro caso de estudio, la clase *Form X* toma el nombre de *FrmPasaraBatch*. A continuación se detallan los cambios que se deben incorporar en esta clase.

8.3.1.1. FrmPasaraBatch

Esta clase se deriva de una Unidad de Interacción de Servicio. Se encarga de ejecutar el servicio que está asociado a esta Unidad de Interacción. Actualmente ya existe código generado por el Compilador de Modelos con el propósito de verificar que no se produzcan fallos en la ejecución del servicio. Este código es el siguiente:

```
if (resp.Error.Number!=0)
{
      FrmError.Show(resp.Error);
}
```

De esta manera si se ha producido algún fallo, se muestra el formulario *FrmError* donde se explica la causa del fallo.

Dado este código, solo habría que añadir en el apartado *else*, las líneas que mostrarán que el servicio se ha ejecutado correctamente. El código para añadir esta nueva funcionalidad sería como el siguiente:

```
if (resp.Error.Number!=0)
      {
            FrmError. ShowMessage(resp.Error, parameters);
      }
      else
      {

//Líneas para llamar a Alert Manager e implementar la funcionalidad de
avisar de la ejecución de una acción del patrón de   usabilidad System
Status Feedback

      FrmAlertManager FrmAlert = new FrmAlertManager();
      FrmAlert.ShowMessage("Acción realizada
       correctamente",parameters);
```

El texto que se mostraría al usuario en caso de éxito o de fracaso sería configurable. En caso de error, el texto estaría guardado en la variable *resp.Error*, mientras que en caso de éxito, el texto se pasa como argumento al método *ShowMessage* de la clase *FrmAlertManager*. En la variable *parameters* se indica cómo se visualizará el mensaje, tal y como se definió en el Modelo de Interacción Concreto.

Faltaría por añadir la definición del formulario que se abre para avisar al usuario del éxito o fracaso en la ejecución, el llamado *FrmAlertManager*. Este formulario sería una nueva clase que se debería añadir al conjunto de clases del sistema. Se debe definir dentro de la carpeta *Generic*, ya que es un formulario genérico que se puede utilizar desde distintas ventanas del sistema.

8.3.1.2. FrmAlertManager

La definición de *FrmAlertManager* solo contiene el código necesario para construir el formulario de aviso al usuario. Esta clase tiene el método *ShowMessage* que recibe como parámetros el texto que se mostrará en el formulario y el tipo de ventana que es (de información, de error o de alerta). El código simplificado en C# que mostraría el mensaje de aviso mediante un formulario sería como el siguiente:

```
public DialogResult ShowMessage(string message, parameter Param)
      {
            //Texto a mostrar en el formulario
            lbl_text.Text=message;
            //NO se permite que el usuario pueda modificar el
tamaño de la ventana
```

```
                this.MinimumSize=this.Size;
                this.MaximumSize=this.Size;
                string windowType = Param.Type;
                switch(windowType)
                {
                //Dependiendo del valor de windowType, muestra una
ventana de información, de error o de aviso

                      case "Information":
                            btn_OK.Text="Aceptar";
                            btn_OK.Location= new Point(200, 80);
                            btn_OK.DialogResult=DialogResult.OK;
                            btn_cancel.Visible=false;
                            if (ShowDialog()==DialogResult.OK)
                                  return DialogResult.OK;
                            break;

                      case "Warning":
                            //Se representan los botones del warning,
2 botones (Aceptar y Cancelar)
                            btn_OK.Text="Aceptar";
                            btn_OK.Update();
                            btn_OK.DialogResult=DialogResult.OK;
                            btn_OK.Location= new Point(120, 80);

                            btn_cancel.Text="Cancelar";

      btn_cancel.DialogResult=DialogResult.Cancel;
                            btn_cancel.Location= new Point(248, 80);


                            if (ShowDialog()==DialogResult.OK)
                                  return DialogResult.OK;
                            else
                                  return DialogResult.Cancel;
                            }
                return DialogResult.OK;
          }
```

El método *ShowMessage* se llama desde la clase *FrmPasaraBatch* cada vez que haya que informar al usuario sobre el éxito o fracaso en la ejecución del servicio. Para simplificar el ejemplo, solo hemos tratado el argumento *Param* para averiguar el tipo de formulario. El tipo de ventana a mostrar será de tipo *Information* para el caso de *System Status Feedback*. El tipo *Warning* que también aparece en este formulario, se explicará más adelante en el documento. Además de informar sobre el tipo de ventana, el argumento *Param* se utiliza para construir el formulario conforme a lo especificado en el Modelo de Interacción Concreto.

8.3.2. Interaction Feedback

Este patrón va a incorporar la funcionalidad de deshabilitar todos los componentes del formulario hasta que se termine de ejecutar el servicio lanzado. Se deshabilitarán tanto los botones como los campos que el usuario debe rellenar para lanzar el servicio. Solo se mantendrá habilitado el botón *Cancel* para que el usuario pueda cancelar la acción a mitad ejecución.

Para ello, solo se va a modificar la clase *Form X*, para que se deshabiliten sus componentes de la ventana. Para el caso de estudio propuesto, la clase *Form X* del Código Genérico será FrmCambiaDirecDest en el Código Específico, y habrá que hacer las siguientes modificaciones:

8.3.2.1. FrmCambiaDirecDest

Esta clase es la que se encarga de solicitar al usuario los argumentos de entrada para ejecutar los servicios. Una vez introducidos estos argumentos, se solicita su ejecución en la parte servidora cuando el usuario lo pida. Basta con añadir en el evento que captura la pulsación del botón *OK,* el código necesario para deshabilitar todos los widgtes y botones antes de que se solicite la ejecución en el servidor. Para nuestro caso concreto, se deshabilitan los campos editables *oidp_thisFactura* y *inps_Drireccion* y el botón *OK*.

```
private new void btnOK_Click(object sender, System.EventArgs e)
            {
            //Se deshabilita el botón de OK y los campos editables
                  oidp_thisFactura.Enabled=false;
                  inps_Drireccion.Enabled=false;
                  btnOK.Enabled=false;
```

Tal y como muestra el código en C#, basta con poner el atributo *Enabled* a falso en cada uno de los componentes de la interfaz.

8.3.3. Progress Feedback

Este patrón representa la funcionalidad de mostrar una barra de progreso mientras se ejecuta una acción que tarda más de dos segundos. Tal y como se ha comentado anteriormente, este patrón se va a aplicar a las transacciones locales y a las globales, ya que las transacciones son operaciones complejas cuya ejecución normalmente dura bastante tiempo. A continuación se presenta cómo sería la incorporación de la barra de progreso en una transacción global. Esta funcionalidad se implementa modificando la clase *Form X* del Código Genérico. Como transacción global del caso de estudio, se ha elegido *Cambiar divisa en todas las facturas.* Esta transacción tiene como entrada dos argumentos: *Divisa nueva* y *Divisa vieja*. La función de esta transacción es la de recorrer todas las instancias de factura y cambiar la divisa cuyo valor sea igual a *Divisa vieja* por el valor del argumento *Divisa nueva. Así*, para nuestro caso de estudio, la clase *Form X* toma el nombre de *FrmSGlobalTransactions_SIU_CAMBIARDIVISA* en el Código Específico.

La incorporación de la funcionalidad de este patrón en las transacciones obliga a modificar la arquitectura del código generado por el Compilador de Modelos. Además de la modificación de la clase *Form X*, la incorporación de la funcionalidad de este patrón haría desaparecer una clase del servidor. Es la clase que se llama *GlobalTransactionAction* en el código generado para cualquier aplicación. Con el Compilador de Modelos actual, esta clase contiene el código de todas las transacciones globales definidas en el Modelado Conceptual. Posee un bucle en el cual recorre todas las instancias afectadas por la transacción global e invoca a todos los servicios que componen la transacción para cada una de estas instancias afectadas. Esta clase desaparecería con la incorporación del patrón, porque ahora este bucle se hará en la parte del cliente. Concretamente, el bucle se hará dentro de cada formulario desde el que se lance la ejecución de una transacción global. Para nuestro caso de estudio, este bucle estará dentro de la clase *FrmSGlobalTransactions_SIU_CAMBIARDIVISA*, y en cada instancia que se ejecuten los servicios de la transacción, se incrementará la barra de progreso. Esto implica perder la atomicidad de las transacciones, por lo tanto no es la mejor opción, pero es la única con la arquitectura actual del código generado.

La solución de hacer la iteración en el cliente no es la más óptima, porque además del inconveniente de perder la atomicidad, requiere tantas comunicaciones entre el cliente y el servidor como instancias se vean afectadas por la ejecución de la transacción, Para cada instancia, el cliente solicitará al servidor que ejecute los servicios que forman la transacción, y el servidor a su vez, debe informar al cliente con los resultados. Por lo tanto para cada instancia que se vea afectada por los servicios de la transacción habrá una comunicación entre cliente y servidor. Sin embargo, tal y como se genera actualmente el código con el Compilador

de Modelos, no se puede mantener el bucle en el servidor y comunicar con el cliente para que incremente la barra de progreso en cada instancia que se ejecute la transacción. Habría que incorporar algún mecanismo de comunicación directa entre servidor y cliente independiente del hilo de ejecución. De esta forma la ejecución sería más eficiente que la solución planteada anteriormente y además se mantendría la atomicidad de las transacciones.

8.3.3.1. FrmSGlobalTransactions_SIU_CAMBIARDIVISA

Esta clase se obtiene de una Unidad de Interacción de Servicio del Modelado Conceptual. Este formulario muestra dos campos de texto donde el usuario debe introducir el objeto valuado de la divisa que se desea cambiar y el valor de la nueva divisa por la que se desea cambiar la antigua. El formulario con la barra de progreso se mostrará al usuario cuando introduzca el valor de ambos campos y pulse el botón de OK para que se lance la ejecución de la transacción global. El código que se añadirá en el evento que captura la pulsación del botón OK de este formulario es el siguiente:

```
//Se crea la ventana de progreso
FrmProgressBar Ventana_progreso= new FrmProgressBar();
//Se inicializa el formulario que mostrará la barra de progreso
Ventana_progreso.Text="ProgressBar";
Ventana_progreso.total_instances=total_instances_involved ;
//Se lanza el hilo que dibujará el progreso de la barra
System.Threading.ThreadPool.QueueUserWorkItem( new
System.Threading.WaitCallback(Ejecutar_accion) ,
Ventana_progreso);
Ventana_progreso.Visible=false;
//Se invoca al método que muestra la barra de progreso
Ventana_progreso.ShowDialog(parameters);
```

El método *ShowDialog* es el encargado de iniciar el proceso para mostrar la barra de progreso. Las opciones de configuración que el analista definió en el Modelo de Interacción Concreto van en el argumento llamado *parameters.*

Además de este código, se debe añadir la definición de la función *Ejecutar_acción* que se encargará de dibujar la barra de progreso mediante un hilo. Se define un bucle para recorrer cada una de las instancias que se ven afectadas por la transacción global. Para cada de estas instancias, se ejecutan los servicios que componen la transacción y se incrementa en uno la barra de progreso. El código para nuestro caso de estudio es el siguiente, donde la barra de progreso se muestra mediante un nuevo formulario:

```
private void Ejecutar_accion( object  status)
      {
//Se define una instancia de la interfaz Thread_progressBar
donde están los métodos para trabajar con el hilo
Thread_ProgressBar callback = (status as Thread_ProgressBar);
//Todas las instancias que se verán afectadas por la transacción
global
int total_instances = callback.Devolver_total_instancias();

callback.Inicializar(0,total_instances,0);
Instance ins;
//Para cada instancia se ejecuta el servicio que compone la
transacción global
      for (int cont=0; cont< total_instances; cont++)
      {
      ins = listInstances[cont];
      resp = Wrapper.Service.Factura.CAMBIARDIVISA(
```

```
        Wrapper.OIDs.Convert.ToFacturaOID((OID)ins.OID),

        Wrapper.OIDs.Convert.ToMonedaOID((OID)oiddivisa_nueva.Value)
                                                    );
        //Se incrementa en una las instancias procesadas
              callback.Incrementar_barra();
              }
              callback.Cerrar_ventana();
        }
```

Además de estos cambios que se añaden en la clase FrmSGlobalTransactions_SIU_CAMBIARDIVISA, se debe crear un formulario nuevo para dibujar la barra de progreso. Este formulario se representa mediante la variable *Ventana_progreso* para el caso de estudio.

8.3.3.2. FrmProgressBar

Esta clase se ha incorporado totalmente nueva en el código generado por el Compilador de Modelos. Un hilo será el encargado de dibujar la barra de progreso en este formulario. Para ello se define una interfaz con los métodos que puede utilizar el hilo, llamada *Thread_ProgressBar*. El formulario *FrmProgressBar* implementa los métodos de esta interfaz. Los métodos principales son los siguientes:

```
        public void Incrementa()
        {
              current_valor++;
              this.progressBar.Value++;
        }
        public void Incrementar_barra()
        {
              Invoke( new IncrementInvoker( Incrementa ) );
        }
```

El método llamado *Incrementa* es el que se invoca desde la clase FrmSGlobalTransactions_SIU_CAMBIARDIVISA cada vez que se han ejecutado los servicios que forman la transacción global para una instancia concreta. Este método hace que el hilo invoque al método *Incrementar_barra* para que aumente la barra de progreso en un segundo plano.

8.3.4. Warning

Este patrón se utiliza para avisar al usuario de las posibles consecuencias que puede tener la ejecución de un servicio irreversible. El código generado para implementar este patrón es similar al código que ya se genera para las precondiciones. Un mensaje de aviso es lo mismo que una precondición, solo que el usuario puede decidir ejecutar el servicio a pesar del mensaje mostrado. El aviso consta de una condición que si se cumple, muestra el mensaje. Entonces el usuario puede decidir si ejecuta o no el servicio dependiendo de dicho mensaje.

La clase en la que se deben aplicar más cambios para implementar este patrón es *Form X* en el Código Genérico. Para nuestro caso de estudio, este patrón lo vamos a utilizar en el servicio *Emitir una factura*, por la tanto la clase de este ejemplo que representa la clase genérica *Form* X, se llama *FrmEmitirFactura* en el Código Específico.

8.3.4.1. FrmEmitirFactura

Dentro de esta clase, los cambios para implementar la funcionalidad del patrón Warning van a estar en el evento que captura la pulsación sobre el botón de OK. La idea es que el servicio se ejecute en el servidor de forma normal, con la salvedad de que si ese servicio tiene definido algún mensaje de aviso, se verifique su condición y en caso de que esta condición sea cierta,

lance una excepción concreta. Al lanzar esta excepción, la ejecución del servicio se interrumpirá y se mostrará al usuario un formulario en el que se le pedirá mediante un mensaje de texto confirmación para ejecutar el servicio.

```
if (resp.Error.Number == 100)
            {
      //Se trata de un aviso de warning
      //Se crea el formulario con el que se le preguntará al usuario
      FrmAlertManager FrmAlert = new FrmAlertManager();
      if(FrmAlert.ShowMessage(resp.Error.ErrorMessage,parameters)==Dia
      logResult.OK, parameters)
            {//El usuario pulsa OK, se ejecuta el servicio
            this.Cursor = Cursors.WaitCursor;
            resp = Wrapper.Service.Factura.EJECUTAREMITIRFACTURA(
      Wrapper.OIDs.Convert.ToFacturaOID((OID)oidpt_THISFactura.Value),
      System.Convert.ToDateTime(inpt_FechaEmision.Value));
```

Este código muestra que se ha utilizado el número de excepción 100 para identificar que la excepción ha sido provocada porque se ha cumplido la condición del mensaje de aviso. Al cumplirse, se muestra el formulario de la clase *AlertManager* y si el usuario pulsa *OK* se vuelve a invocar el servicio. El argumento *parameters* del método *ShowMessage* contiene las opciones de configuración del patrón *Warning* que el analista modeló en el Modelo de Interacción Concreto.

En cuanto a la clase *AlertManager* que se utiliza, es la misma que se ha mostrado en el punto 8.3.1.2. Al igual que en ese apartado, el argumento *parameter* indicaría las propiedades modeladas en el Modelo de Interacción Concreto. En la sección 8.3.1.2 solo se trata a modo de ejemplo la propiedad del tipo de ventana, pero son muchos más los aspectos configurables desde el Modelado Conceptual, tal y como se ha comentado en la sección 7.1.4.

Tal y como se ha presentado el código, si no se añade nada más, éste generaría bucles infinitos, ya que se volvería a solicitar la ejecución del servicio cuando el usuario lo deseara y al ejecutar el servicio, volvería a lanzarse la excepción. Por lo tanto, el servicio no se ejecutaría correctamente nunca. Para evitar esta situación, el Compilador de Modelos debe generar una variable de guarda. Es decir, una variable que indicará si la petición de ejecución del servicio viene a posteriori de que se haya lanzado la excepción (ya se ha mostrado el aviso al usuario), o si esta petición de ejecución es la primera que se realiza (aun no se ha mostrado el aviso al usuario). Esta variable se controla desde la otra clase que se debe modificar para incorporar el patrón de *Warning*: *FacturaAction*.

8.3.4.2. FacturaAction

Esta clase de la parte servidora contiene el código que comprueba las restricciones de integridad y las precondiciones, ambas definidas en el Modelo de Objetos. Además comprueba si el servicio se ejecuta dentro de un estado válido del objeto, según lo especificado en el Modelo Dinámico. Por lo tanto esta clase es la idónea para comprobar si la condición del mensaje de aviso se cumple.

Para lanzar la excepción que hará que la clase *FrmEmitirFactura* muestre el mensaje de aviso, se deben dar simultáneamente estas dos situaciones:

1. Que la condición que se definió en el Modelo de Objetos para el mensaje de aviso sea cierta.
2. Que la variable de guarda, llamada *VWarning* esté a falso. Si está a falso, quiere decir que el mensaje de aviso aun no se ha mostrado al usuario. Esta variable es la que impide que entremos en ciclos infinitos, tal y como se explica en el punto anterior.

```
if (!(FacturaInstance.AssocOperator002(OnContext, Instance) > new
ONNat("10000")) && VWarning= false)
      {
// Warning: Las facturas con más de 10000 euros son casos poco comunes
// Se cambia el valor de la vaiable de guarda para no volver mostrar
el mensaje de aviso
VWarning= true;
ListDictionary lParameters = new ListDictionary();

throw new ONWarningException(null,
"Clas_1103105228800242Pre_5_MsgError", "La factura que está emitiendo
es de más de 10.000 €. ¿Está seguro que desea emitirla?",
lParameters);
                  }
```

Tal y como se muestra en el código, la excepción lleva asociado el texto que se mostrará al usuario y que se definió en el Modelo de Objetos al modelar la funcionalidad del patrón *Warning.* De esta forma se lanza la excepción que se captura y trata desde la clase *FrmEmitirFactura*.

8.4. Cambios en el Modelo de Usabilidad

Tal y como se ha comentado anteriormente (sección 6.2), el Modelo de Usabilidad propuesto ha sido enriquecido con cuatro nuevos atributos, uno por cada uno de los patrones de la familia Feedback. Estos cuatro atributos, al igual que el resto, deben ser clasificados según la etapa en la cual se pueden medir: en el Modelado Conceptual, en el Compilador de Modelos o en la aplicación final. De los tres grupos de atributos, el presente documento solo va a tratar los correspondientes al primer grupo.

La incorporación de nuevas primitivas de modelado para soportar la funcionalidad de los patrones de usabilidad (sección 7.1) hace posible que su medición se pueda hacer en la fase de Modelado Conceptual, a excepción del atributo *Realimentación de interacción* (obtenido a partir del patrón Interaction Feedback). Este atributo solo se puede medir en el Compilador Modelos, ya que siempre se aplica de la misma forma al código generado y por lo tanto no está soportado por primitivas conceptuales.

Los atributos que son medibles en la fase de Modelado Conceptual necesitan tener definida una fórmula para obtener una métrica de la usabilidad y un indicador para dar significado al valor numérico obtenido mediante la fórmula. Las fórmulas de las métricas están basadas en las primitivas conceptuales incorporadas para modelar la funcionalidad de cada uno de los patrones de usabilidad. A continuación se presentan las métricas e indicadores de los atributos: *realimentación del estado del sistema, realimentación de progreso* y *aviso*.

8.4.1. Métricas

En esta sección se muestran las fórmulas que componen las métricas de cada uno de los atributos derivados de los patrones de la familia Feedback que tienen primitivas de modelado conceptual. Estas métricas se utilizan para medir el grado de usabilidad de cada uno de estos atributos.

- **Realimentación del estado del sistema:** este atributo se utiliza para indicar si el sistema tiene la capacidad de avisar al usuario sobre el éxito o el error en la ejecución de un servicio. Se han incorporado primitivas a nivel de Modelado Conceptual para definir ambos tipos de mensajes y la forma en la que se mostrarán estos mensajes. Por lo tanto hay dos aspectos que se deben medir: la idoneidad del mensaje que se muestra, y la forma en la que se muestra. Ambos aspectos se miden tanto para los mensajes de éxito como para los mensajes de error.

- Idoneidad del mensaje: el Compilador de Modelos genera por defecto un mensaje tanto para el caso de éxito como para el caso de error en la ejecución de un servicio. Este mensaje, por defecto, es un mensaje genérico igual para todos los servicios y para todos los sistemas, por lo tanto la información que proporciona es pobre para un servicio concreto. Sobre todo para el caso de un mensaje de error, ya que no proporciona información sobre cómo arreglar el error. Esta métrica consiste en medir los mensajes de tipo texto cuyo contenido ha sido definido explícitamente por el analista. El hecho que esté definido por el analista no garantiza que sea el mensaje más apropiado, pero sí es un buen indicador de que el mensaje mostrado será más adecuado que el mensaje por defecto, ya que el analista se ha molestado en definirlo. Esta métrica solo tiene sentido si el mensaje es de tipo texto, para el caso en que el mensaje se muestre mediante iconos, o se oculte, no se tendrá en cuenta en la fórmula.

$$\forall x \in MensajeDeEstadoTipoTexto : \frac{\sum DefinidosPorAnalista(x)}{\sum x} = RES1$$

- Forma del mensaje: de forma automática no se puede decidir si un mensaje es más adecuado mostrarlo mediante texto o mediante iconos, o incluso medir la posición de la pantalla principal donde sería más adecuado mostrarlo, ya que todos estos factores son meramente subjetivos por parte de el usuario que va a utilizar la aplicación. Por lo tanto todos estos factores serían medibles en la aplicación final mediante tests de usuarios. En cambio, sí que es posible medir en qué mensajes el analista ha ocultado el muestreo de los mensajes. Estos casos reducen la usabilidad de la aplicación considerablemente y deben por lo tanto evitarse en la medida de lo posible. Esta métrica por lo tanto va a estar basada en una fórmula que cuente cuántos mensajes no están ocultos para el usuario.

$$\forall x \in Servicio : \frac{\sum MensajeEstadoNoOculto(x)}{\sum x} = RES2$$

- **Realimentación de progreso**: este atributo mide si el sistema es capaz de mostrar al usuario que su petición de ejecución de un servicio está siendo atendida, además de mostrar el tiempo restante hasta la finalización de su ejecución. Esta métrica está pensada para medir si la forma de visualización de esta barra de progreso es la más indicada para el sistema. Existen muchas formas de mostrar la barra, como por ejemplo, en un nuevo formulario, en la ventana principal, además la barra puede ser gráfica o de tipo de texto. La idoneidad de todas estas alternativas depende del usuario y es algo subjetivo y por tanto solo puede ser medible por el propio usuario en la aplicación final. Sin embargo, sí que se pueden medir aquellos servicios en los que el analista ha optado por ocultar la barra de progreso. Por lo tanto la fórmula asociada a esta métrica mide los servicios que no tienen ocultada la barra de progreso, ya que los servicios con barra de progreso incrementan la usabilidad del sistema.

$$\forall x \in Servicio : \frac{\sum Barra\,\mathrm{Pr}\,ogresoNoOculta(x)}{\sum x} = RDP$$

- **Aviso:** este atributo indica si el sistema ha definido, para aquellos servicios que sean irreversibles, un mensaje de aviso para mostrar al usuario cada vez que se vayan a ejecutar. El usuario al leer este mensaje puede continuar con la ejecución del servicio o bien cancelarlo. Estos mensajes siempre van asociados a una condición que hace de guarda para mostrar el mensaje (el mensaje solo se muestra si la condición es cierta). En la fase del Modelado Conceptual, no se puede saber qué servicios van a realizar una acción irreversible por el sistema. Esta propiedad de los servicios solo la sabe el analista gracias a los requisitos capturados. A partir de estos requisitos es él el que debe construir los Modelos Conceptuales de forma que sean coherentes con los requisitos. Sin embargo, sí que se puede calcular el porcentaje de servicios que tienen definido un mensaje de aviso. Aquellos sistemas que no tengan definido ninguno de

estos mensajes indican que serán poco usables, porque algunos de los servicios será irreversible, por ejemplo la acción de borrado de instancias, y no se han tenido en cuenta en la fase de modelado. Por lo tanto, esta fórmula se va a encargar de calcular el porcentaje de mensajes de aviso por número de servicios definidos en el sistema.

$$\forall x \in Servicio : \frac{\sum MensajeAvisoDefinido(x)}{\sum x} = AV$$

8.4.2. Indicadores

Los indicadores se utilizan para dar un significado al valor de las métricas obtenidas mediante las fórmulas. Estos indicadores se obtienen a partir de heurísticos [27], criterios ergonómicos [4] y guías de usabilidad [41], además de la definición de los patrones de STATUS [22]. El valor que puede obtener cada una de las métricas definidas en el punto anterior pueden ser cinco: Muy Bueno (MB), Bueno (B), Regular (R), Malo (M), Muy Malo (MM). Para asignar uno de estos valores cualitativos a las fórmulas, hay que definir unos rangos que en base al valor numérico obtenido en la fórmula, se le asigne uno de estos valores cualitativos al atributo. La Tabla 32 define dichos rangos.

En la Tabla 32 aparecen las métricas *RES1* y *RES2* repetidas, ya que se definen indicadores para cuando el mensaje es de confirmación y para cuando el mensaje es de error. RES*(OK)* indica el atributo *realimentación del estado del sistema* para el caso de una ejecución correcta del servicio y *RES(ER)* indica el mismo atributo pero para el caso de un error en la ejecución. Es importante hacer esta distinción porque en el caso de los mensajes que avisan de un error es más importante que sean definidos por el analista, ya que además de informar del error deben proponer una solución al problema. Esta solución solo la conoce el analista y no aparece en el mensaje que genera el Compilador de Modelos por defecto. Además de esta diferencia, también es más grave que no se avise al usuario de la existencia de un error, que de que el servicio se haya ejecutado correctamente. Todos los servicios deberían tener la posibilidad de informar al usuario de los errores que ocurran, en cambio la posibilidad de mostrar la confirmación de ejecución no es tan importante para conseguir un sistema usable:

Medida	MB	B	R	M	MM
RES(OK)1	RES1 ≥ .85	.85> RES1 ≥ .70	.70>RES1 ≥ .55	.55>RES1 ≥ .40	RES1 <.40
RES(ER)1	RES1 ≥ .95	.95> RES1 ≥ .85	.85>RES1 ≥ .75	.75>RES1 ≥ .65	RES1<.65
RES(OK)2	RES2 ≥ .99	.99> RES2 ≥ .95	.95>RES2 ≥ .90	.90>RES2 ≥ .85	RES2<.85
RES(ER)2	RES2 ≥ .90	.90> RES2 ≥ .80	.80> RES2 ≥ .70	.70>RES2 ≥ .60	RES2<.60
RDP	RDP ≥ .99	.99> RDP ≥ .95	.95> RDP ≥ .90	.90>RDP ≥ .85	RDP <.85
AV	AV ≥ .40	.40> AV ≥ .30	.30> AV ≥ .20	.20> AV ≥ .10	AV <.10

Tabla 32. Rango de indicadores para los nuevos atributos al aplicar el patrón de Feedback

8.4.3. Validación de las formulas

Por último, una vez definidas las métricas y los indicadores solo resta demostrar que las fórmulas en las que se apoyan las métricas son correctas. Para ello se ha vuelto a utilizar el método propuesto por Poels llamado DISTANCE [34]. De forma similar a como se validaron todas las fórmulas de los atributos de usabilidad en la sección 1, en esta sección se hace la evaluación de los nuevos atributos derivados de los patrones de usabilidad de la familia Feedback.

A continuación se presentan las definiciones de las fórmulas mediante DISTANCE. El poder representar estas fórmulas mediante dicha notación garantiza que las fórmulas sean válidas.

Nombre de la fórmula		RES1 (Retroalimentación del Estado del Sistema 1)
Aspecto a medir (*attr)*		Porcentaje de mensajes de realimentación de estado de tipo texto que han sido definidos explícitamente por el analista para indicar el éxito y error en la ejecución de los servicios (Esta fórmula se aplica tanto en caso de éxito como de error).
Entidad de Software (p∈ P)		MCOO ∈ UMCOO
Resultados de DISTANCE		
abs	abs_{RES1}:UMCOO → ℘(UMRT) : MCOO → MRT(MCOO)	UMRT es el Universo de los Mensajes de Realimentación del sistema de tipo Texto, cuyo texto ha sido definido por el analista. MRT es la operación que devuelve el conjunto de Mensajes de Realimentación del sistema de tipo Texto, cuyo texto ha sido definido por el analista.
T_e	$T_{0\text{- RES1}}$= ℘(UMRT) → ℘(UMRT) : s → s ∪ {e} con e ∈ UMRT $T_{1\text{- RES1}}$= ℘(UMRT) → ℘(UMRT) : s → s - {e} con e ∈ UMRT	Las funciones homogéneas (T_e) agregan o eliminan mensajes de realimentación de estado de tipo texto, cuyo texto ha sido definido por el analista.
δ	δ_{RES1}: ℘(UMRT) x ℘(UMRT) → ℜ : (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de mensajes de realimentación de estado de tipo texto, cuyo texto ha sido definido por el analista, a otro.
ref	ref_{RES1}: UMCOO → ℘(UMRT): MCOO → ϕ	Sería el conjunto formado por todos los mensajes de realimentación de estado que no sean de tipo texto o que no estén definidos por el analista.
μ	μ_{RES1}= δ (abs_{RES1} (MCOO),ref_{RES1} (MCOO)) μ_{RES1}= |MRT(MCOO)|	La medida sería el número de mensajes de realimentación de estado de tipo texto, cuyo texto ha sido definido por el analista, que existen en el conjunto MRT(MCOO). A partir de este número se podría obtener el porcentaje de mensajes de realimentación de estado de tipo texto, y cuyo texto ha sido definido por el analista, que existen en todo el sistema. Para ello solo hay que contar el total de mensajes de realimentación de estado de tipo texto existentes.

Tabla 33. Definición basada en DISTANCE de la fórmula RES1

Nombre de la fórmula		RES2 (Retroalimentación del Estado del Sistema 2)
Aspecto a medir (*attr)*		Porcentaje de mensajes de realimentación de estado que no están ocultos para el usuario entre todos los servicios (Esta fórmula se aplica tanto en caso de éxito como de error).
Entidad de Software (p∈ P)		MCOO ∈ UMCOO
Resultados de DISTANCE		
abs	abs_{RES2}:UMCOO → ℘(UMRO) : MCOO → MRO (MCOO)	UMRO es el Universo de los Mensajes de Realimentación de estado que no están Ocultos. MRO es la operación que devuelve el conjunto de Mensajes de Realimentación de estado que no están Ocultos. .
T_e	$T_{0\text{- RES2}}$ = ℘(UMRO) → ℘(UMRO) : s → s ∪ {e} con e ∈ UMRO $T_{1\text{- RES2}}$ = ℘(UMRO) → ℘(UMRO) : s → s - {e} con e ∈ UMRO	Las funciones homogéneas (T_e) ocultan o desocultan los mensajes de realimentación de estado.

δ	δ_{RES2}: $\wp$ (UMRO) x $\wp$ (UMRO) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de mensajes de realimentación de estado visibles, a otro distinto.
ref	*ref* $_{RES2}$: UMCOO → $\wp$ (UMRO): MCOO → ϕ	Sería el conjunto formado por todos los mensajes de realimentación de estado ocultos para el usuario.
μ	$\mu_{RES2} = \delta$ (*abs* $_{RES2}$ (MCOO),*ref* $_{RES2}$ (MCOO)) μ_{RES2} = | MRO (MCOO)|	La medida sería el número de mensajes de realimentación de estado no ocultos, que existen en el conjunto MRO(MCOO). A partir de este número se podría obtener el porcentaje de mensajes de realimentación no ocultos que existen en todo el sistema. Para ello solo hay que contar el total de servicios existentes.

Tabla 34. Definición basada en DISTANCE de la fórmula RES2

Nombre de la fórmula	RDP (Realimentación De Progreso)	
Aspecto a medir (*attr)*	Porcentaje de barras de progreso que no están ocultas para el usuario entre todos los servicios.	
Entidad de Software (p∈ P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{RDP}$:UMCOO → $\wp$ (UBPO) : MCOO → BPO (MCOO)	UBPO es el Universo de las Barras de Progreso que no están Ocultas. BPO es la operación que devuelve el conjunto de Barras de Progreso que no están Ocultas.
T_e	$T_{0\text{- RDP}} = \wp$ (UBPO) → $\wp$ (UBPO) : s → s ∪ {e} con e ∈ UBPO $T_{1\text{- RDP}} = \wp$ (UBPO) → $\wp$ (UBPO) : s → s - {e} con e ∈ UBPO	Las funciones homogéneas (T_e) ocultan o desocultan la barra de progreso asociada a un servicio.
δ	δ_{RDP}: $\wp$ (UBPO) x $\wp$ (UBPO) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de barras de progreso visibles, a otro distinto.
ref	*ref* $_{RDP}$: UMCOO → $\wp$ (UBPO): MCOO → ϕ	Sería el conjunto formado por todas las barras de progreso ocultas para el usuario.
μ	$\mu_{RDP} = \delta$ (*abs* $_{RDP}$ (MCOO),*ref* $_{RDP}$ (MCOO)) μ_{RDP} = | BPO (MCOO)|	La medida sería el número de barras de progreso no ocultas, que existen en el conjunto BPO(MCOO). A partir de este número se podría obtener el porcentaje de barras de progreso no ocultas que existen en todo el sistema. Para ello solo hay que contar el total de servicios existentes.

Tabla 35. Definición basada en DISTANCE de la fórmula RDP

Nombre de la fórmula	AV (AViso)	
Aspecto a medir (*attr)*	Porcentaje de mensajes de aviso definido entre todos los servicios.	
Entidad de Software (p∈ P)	MCOO ∈ UMCOO	
Resultados de DISTANCE		
abs	abs $_{AV}$:UMCOO → $\wp$ (UBPO) : MCOO → MAD (MCOO)	UMAD es el Universo de los Mensajes de Aviso Definidos.

		MAD es la operación que devuelve el conjunto de los Mensajes de Aviso Defnidos.
T_e	$T_{0\text{- AV}} = \wp$ (UMAD) → $\wp$ (UMAD) : s → s $\cup$ {e} con e $\in$ UMAD $T_{1\text{- AV}} = \wp$ (UMAD) → $\wp$ (UMAD) : s → s - {e} con e $\in$ UMAD	Las funciones homogéneas (T_e) añaden o eliminan mensajes de aviso.
δ	δ_{AV}: $\wp$ (UMAD) x $\wp$ (UMAD) → $\Re$: (s, s') → c (|s - s'| + |s' - s|)	Número de transformaciones elementales necesarias para pasar de un número de mensajes de aviso definidos, a otro distinto.
ref	*ref* $_{AV}$: UMCOO → $\wp$ (UMAD): MCOO → ϕ	Sería el conjunto formado por ningún mensaje de aviso definido.
μ	$\mu_{AV} = \delta$ (*abs* $_{AV}$ (MCOO),*ref* $_{AV}$ (MCOO)) μ_{AV} = | MAD (MCOO)|	La medida sería el número de mensajes de aviso definidos, que existen en el conjunto MAD(MCOO). A partir de este número se podría obtener el porcentaje de mensajes de aviso que existen en todo el sistema. Para ello solo hay que contar el total de servicios existentes.

Tabla 36. Definición basada en DISTANCE de la fórmula AV

9. CASO DE ESTUDIO

En esta sección se va a aplicar a un caso de estudio el método de evaluación de la usabilidad propuesto a lo largo de todo el presente documento. La idea es medir su usabilidad a partir del Modelo Conceptual del caso de estudio. El caso de estudio elegido para este fin ha sido el mismo que el utilizado en la sección 8.3, el de Aguas de Bullent, un sistema encargado de gestionar una compañía de reparto de agua potable en los domicilios.

Este sistema es un sistema de tamaño medio y por lo tanto su funcionalidad y su Modelo Conceptual son complejos. Medir la usabilidad de todo el sistema a mano sería una tarea muy larga y pesada para el lector. Además, el objetivo del método de evaluación de la usabilidad es que esta evaluación se haga de forma automática y no manualmente. El hecho de hacer la evaluación se hace con el propósito de mostrar la aplicación práctica al lector mediante un ejemplo concreto. Por ello se ha optado por centrarse en unas pocas tareas del sistema. La medición de usabilidad se hará solo sobre estas tareas, de esta forma el uso del método de evaluación de la usabilidad será más claro para el lector que si se aplica a todo el sistema.

Las tareas se han elegido para, además de aplicar el Modelo de Usabilidad a la evaluación de la usabilidad del sistema, poder aplicar las nuevas primitivas de modelado surgidas al aplicar los patrones de usabilidad de la familia Feedback a OO-Method. Por lo tanto el objetivo que se pretende con este caso de estudio es doble: evaluar la usabilidad del sistema mediante el Modelo de Usabilidad y ver la aplicabilidad de las nuevas primitivas de modelado incorporadas a OO-Method.

Empezaremos presentando las tareas que formarán parte del estudio realizado. Estas tareas son las únicas que se tendrán en cuenta a lo largo de todo el caso de estudio.

9.1. Tareas del caso de estudio

Todas las tareas utilizadas están en el módulo del sistema llamado *Facturas*. Este módulo se encarga de gestionar todo el sistema de facturación de la empresa de reparto de agua. El Modelo de Interacción en el que están basadas todas las tareas utilizadas es el que se muestra en la Figura 36. A continuación se presentan las tareas representadas en la Figura 36 que forman parte del caso de estudio.

9.1.1. Pasar la factura a batch

Esta tarea se utiliza para pasar la factura al lote de facturas resueltas. Las facturas resueltas son aquellas facturas que ya han sido emitidas, pagadas y en las que el cliente ha estado conforme.

Esta tarea es accesible en el sistema desde la interfaz llamada *Facturas*. En esta interfaz se presenta un listado con todas las facturas almacenadas en el sistema. Dentro de esta interfaz, la tarea está representada por el servicio llamado *Pasar factura a batch*, tal y como se muestra en la Figura 37.

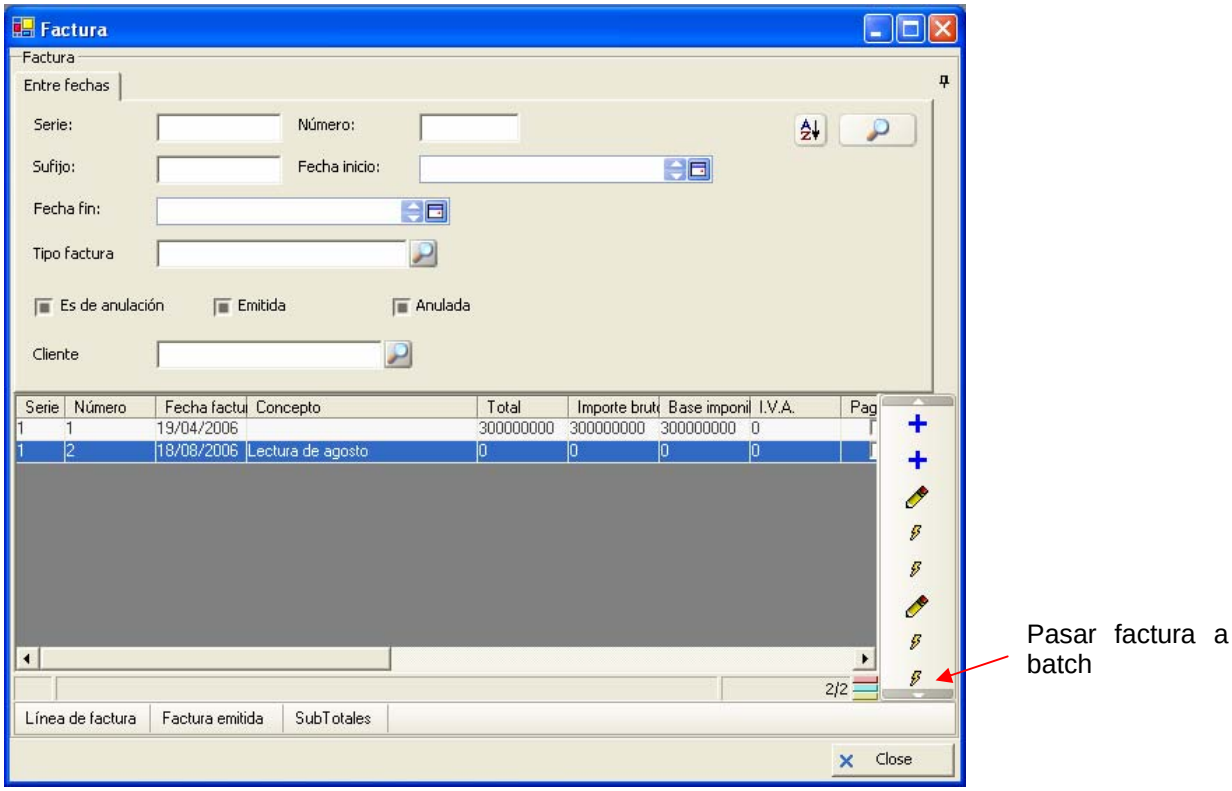

Figura 37. Interfaz desde la que se selecciona el servicio *pasar factura a batch*

Una vez pulsado el botón de servicio llamado *Pasar factura a batch,* se muestra al usuario la interfaz mostrada en la Figura 38. En esa interfaz el usuario solo debe seleccionar la factura sobre la que se ejecutará el servicio de pasar la factura a batch. Una vez seleccionada la factura, el usuario pulsa sobre el botón *OK* y se ejecuta el servicio.

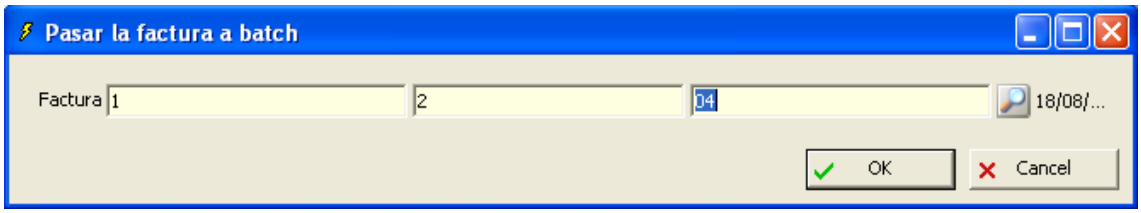

Figura 38. Interfaz desde la que se ejecuta el servicio *pasar factura a batch*

9.1.2. Cambiar la dirección de destino

Esta tarea se utiliza para cambiar la dirección de destino de una factura ya creada. Esta tarea es útil cuando el operario que utiliza el sistema se ha equivocado en la dirección de destino de la factura al darla de alta y desea cambiar dicha dirección posteriormente. La Figura 39 muestra la interfaz desde la cual se selecciona el servicio de *Cambiar la dirección de destino*.

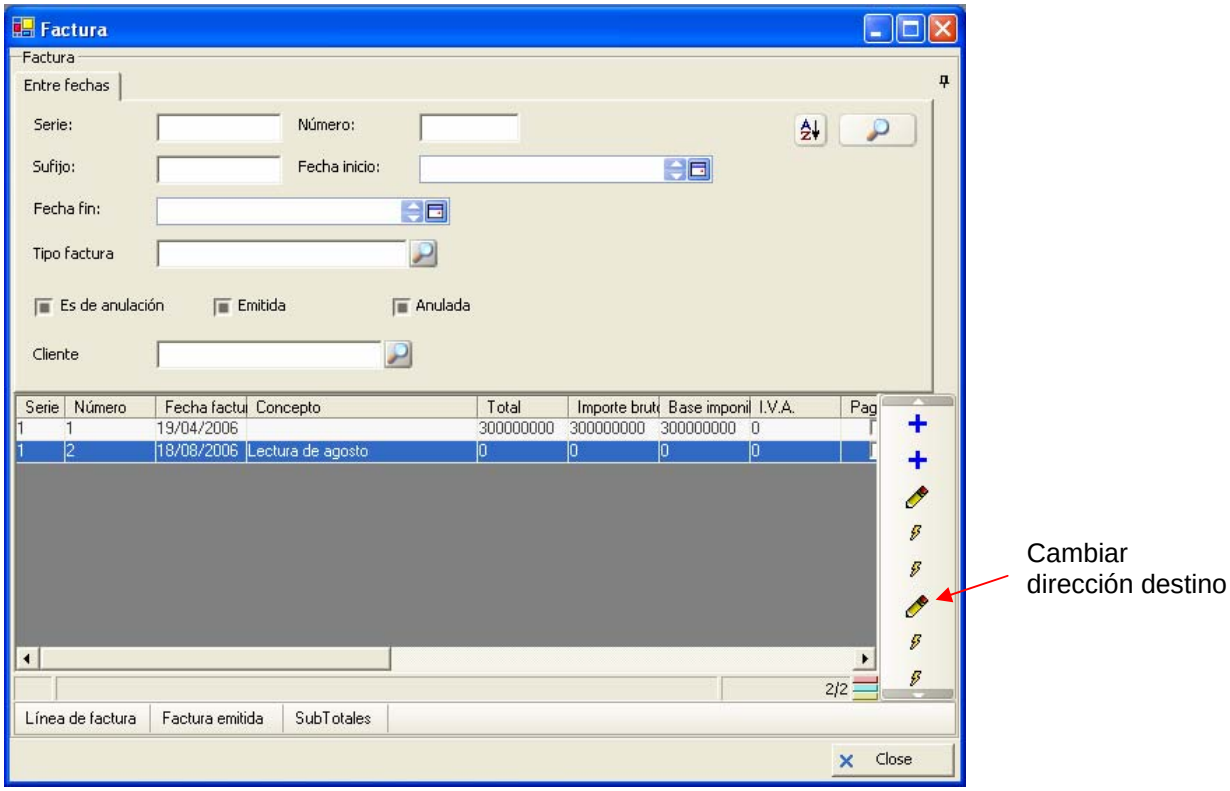

Figura 39. Interfaz desde la que se selecciona el servicio *cambiar dirección destino*

Una vez pulsado el botón de acción resaltado en la Figura 39, se muestra una interfaz desde la cual el usuario selecciona la factura a la que desea cambiarle su dirección de destino. Una vez seleccionada la factura, el usuario debe introducir la dirección de destino correcta. Al pulsar el botón de *OK*, la nueva dirección introducida se asigna a la factura seleccionada como su nueva dirección de destino.

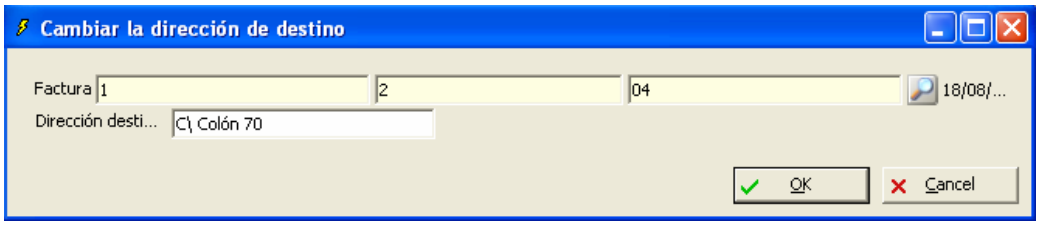

Figura 40. Interfaz desde la que se ejecuta el servicio *cambiar dirección destino*

9.1.3. Cambiar divisa en todas las facturas

Esta tarea representa la implementación de una transacción global en OO-Method. Esta tarea recorre todas las instancias de factura guardadas en el sistema para actualizar sus divisas. Permite cambiar el valor de una divisa ya existente en el sistema por el de otra nueva introducida por el usuario. Este servicio se invoca desde el botón de acción marcado en la Figura 41.

.

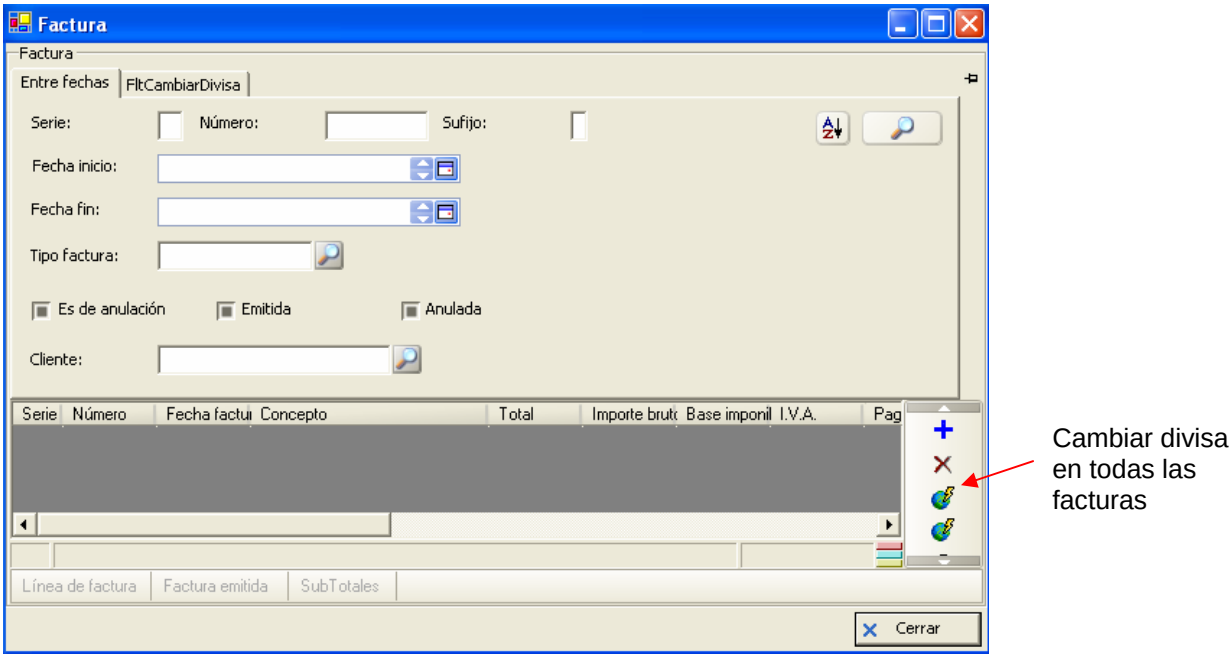

Figura 41. Interfaz desde la que se selecciona el servicio *cambiar divisa en todas las facturas*

Una vez pulsado el botón de acción *Cambiar divisa en todas las facturas*, se muestra la interfaz de la Figura 42. Desde esta interfaz el usuario introduce el valor de la *Divisa vieja* y el de la *Divisa nueva*. Una vez pulsado el botón de *OK,* esta tarea cambia la divisa de todas las facturas cuya divisa sea igual a *divisa vieja* por el valor de *divisa nueva*.

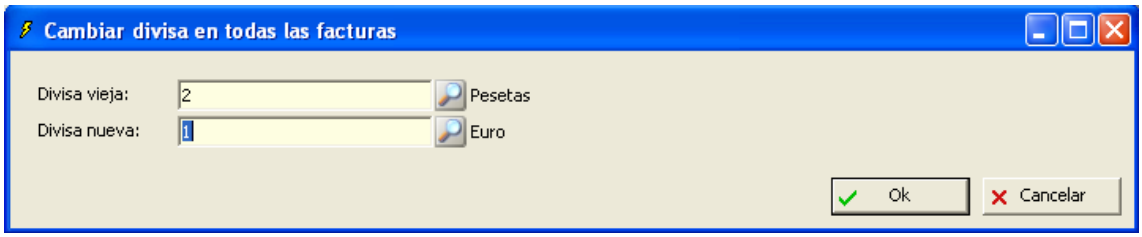

Figura 42. Interfaz desde la que se ejecuta el servicio *cambiar divisa en todas las facturas*

9.1.4. Emitir factura

Esta tarea marca una factura como emitida, es decir, se registra en el sistema que la factura se ha enviado al cliente junto con la fecha en la cual se ha enviado. Esta tarea se invoca pulsando el botón de acción resaltado en la Figura 43.

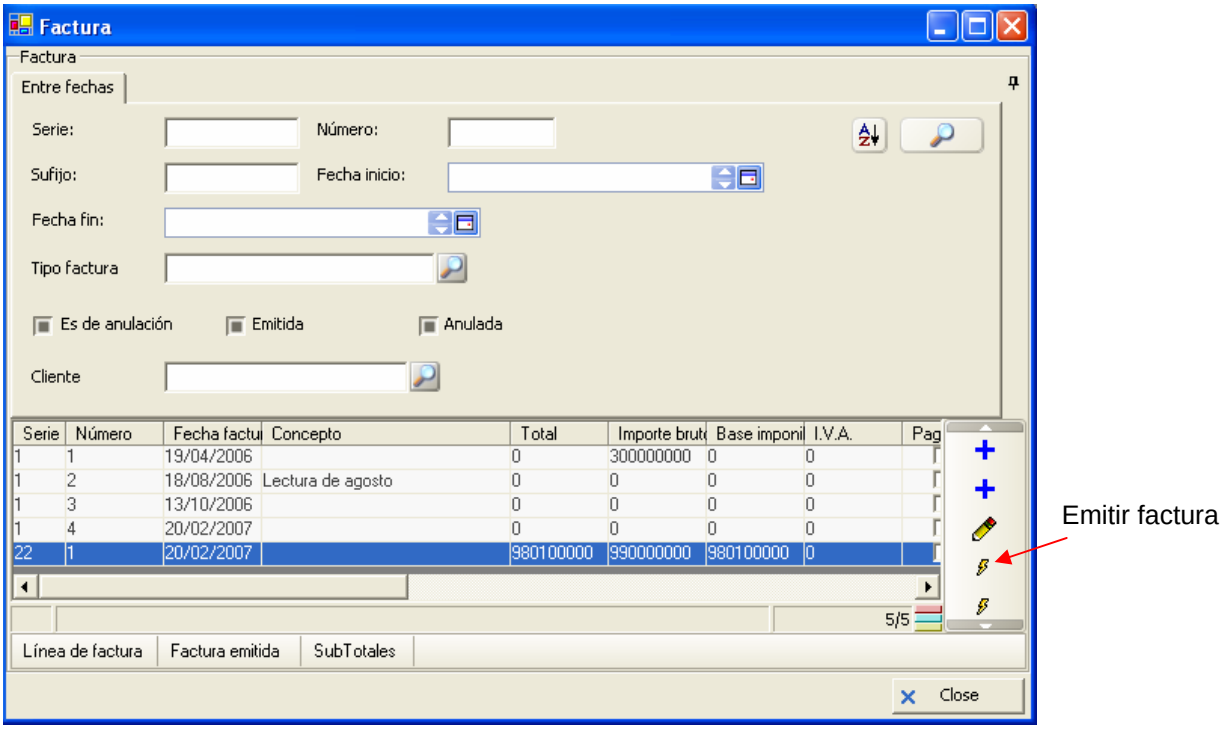

Figura 43. Interfaz desde la que se selecciona el servicio *emitir factura*

Al pulsar el botón de acción *Emitir factura* se presenta una interfaz como la mostrada en la Figura 44. En esta interfaz el usuario debe seleccionar la factura a emitir e indicar la fecha en la cual se emite. Al pulsar el botón de *OK* se almacena dicha información en el sistema.

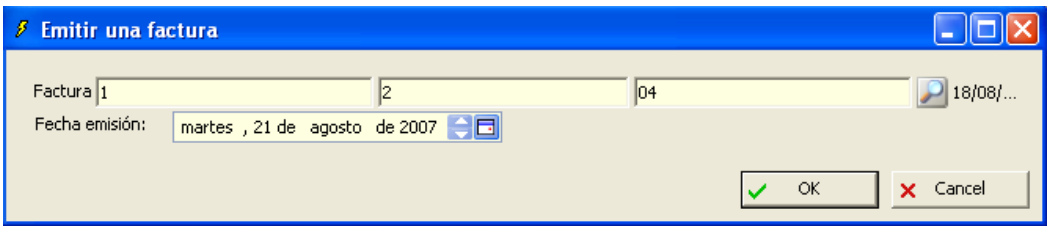

Figura 44. Interfaz desde la que se ejecuta el servicio *emitir factura*

9.2. Problemas de usabilidad de las tareas

En esta sección se presentan qué problemas de usabilidad plantean las tareas explicadas en la sección anterior. Todos los problemas aquí explicados se resuelven mediante el uso de las nuevas primitivas de modelado incorporadas a OO-Method derivadas del patrón Feedback. De esta forma se puede observar el beneficio obtenido al incorporarlas a OO-Method.

Los problemas de usabilidad detectados no son exclusivos de una única tarea. Algunos de estos problemas son compartidos por todas las tareas. A continuación se presentan los problemas y las tareas a las cuales afectan cada uno de ellos.

9.2.1. Ciertas acciones no informan al usuario de su ejecución

Una vez se han ejecutado las acciones, no todas muestran al usuario si la ejecución ha finalizado o no con éxito. Por ejemplo, tal como muestra la Figura 45, al emitir una factura, sí

que aparece un checkbox que puede servir al usuario para verificar si el servicio de *Emitir factura* se ha ejecutado o no correctamente.

Sin embargo, hay otro tipo de acciones como por ejemplo, *Pasar la factura a batch,* que no informan al usuario si su ejecución finaliza con éxito. Solo le avisa si existe algún error, usando para ello un mensaje genérico creado por el Compilador de Modelos. El usuario para comprobar que el resultado de este servicio ha sido correcto, tendría que abrir otra ventana (mediante navegación, por ejemplo) en la cual se pudiera comprobar que la factura ya ha sido pasada a batch. Además, el analista no tiene la posibilidad de modificar el texto de los mensajes de error.

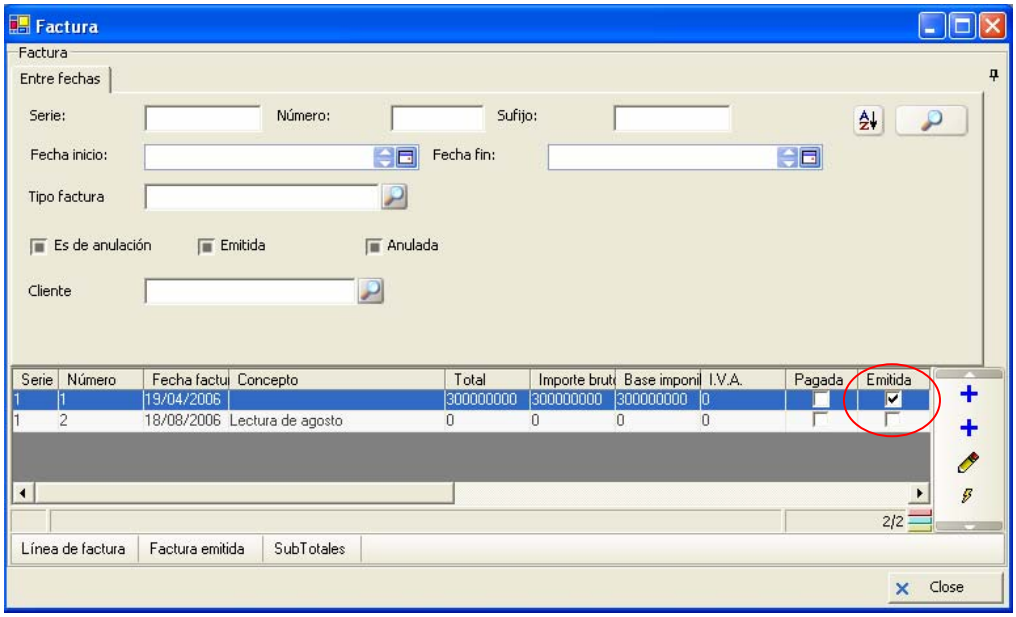

Figura 45. Indicación al usuario del resultado del servicio *Emitir factura*

Además, cuando se lanza un servicio mientras el objeto está en un estado desde el cual no se puede lanzar ese servicio según el Diagrama Dinámico de OO-Method, aparece un mensaje poco comprensible por el usuario. La Figura 46 muestra el mensaje de error cuando se lanza el servicio *Emitir factura* estando la factura en un estado desde el cual no se puede ejecutar este servicio.

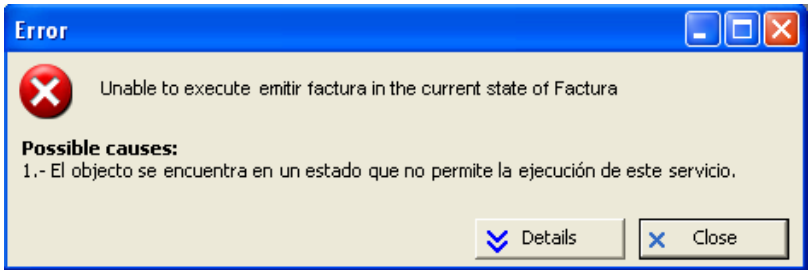

Figura 46. Mensaje de error provocado por una transacción no válida entre estados del objeto

De todas las tareas que forman el caso de estudio, ninguna muestra confirmación explícita de que el servicio se ha ejecutado correctamente. En cuanto a mensajes de error genéricos, todas las tareas muestran el mismo error genérico que el mostrado en la Figura 46 cada vez que se solicita su ejecución estando el objeto en un estado no válido para la ejecución de ese servicio.

9.2.2. Al lanzar un servicio el usuario no puede verificar si el sistema está procesando su petición

Con el Compilador de Modelos de OO-Method actual, la única forma de mostrar al usuario que su petición está siendo procesada es a través del puntero del ratón. Cada vez que el usuario lanza un servicio, el puntero del ratón pasa a convertirse en un reloj de arena. Mediante este mecanismo, el usuario puede verificar si su petición está siendo atendida o no. Esta funcionalidad ya está contemplada en el actual Compilador de Modelos.

El mecanismo de cambio de aspecto del puntero del ratón puede ser suficiente, sin embargo existen otros mecanismos para informar al usuario de que su petición está siendo atendida. Cuantos más mecanismos se utilicen, más fácilmente recibirá el usuario la información. Cuando un usuario lanza la ejecución de un servicio y no hay un mecanismo adecuado para avisar al usuario que su petición se está procesando, puede que por error o por impaciencia, el usuario vuelva a lanzar nuevamente el mismo servicio creyendo que su petición no ha sido atendida. Así el sistema ejecutaría dos veces el mismo servicio, mientras que el usuario solo quería ejecutarla una vez.

Esta funcionalidad sería útil en cualquier formulario del caso de estudio generado a partir de una Unidad de Interacción de Servicio del Modelo de Interacción. Es decir, esta funcionalidad es útil para las interfaces desde las cuales el usuario debe introducir el valor de los argumentos para ejecutar un servicio. Por lo tanto, este problema afecta a las cuatro tareas descritas.

9.2.3. El usuario no sabe cuánto falta para terminar un servicio complejo

Cuando el usuario ejecuta un servicio no puede estimar cuánto tiempo resta para que este servicio finalice. Para acciones que requieren poco tiempo, esta funcionalidad no tiene sentido, ya que el usuario apenas apreciará el tiempo de espera. Actualmente OO-Method ya contempla la barra de progreso para algunas acciones. Por ejemplo, si un servicio se aplica sobre un conjunto de instancias seleccionadas, se mostrará una barra de progreso para indicar el tiempo que resta para que finalice la ejecución del servicio en todas las instancias seleccionadas.

Sin embargo, para otro tipo de acciones complejas como por ejemplo, las transacciones, que engloban varios servicios o incluso otras transacciones, el tiempo de espera que el usuario está esperando la finalización del servicio puede ser considerable y actualmente no se muestra una barra de progreso en su ejecución.

De las cuatro tareas que forman el caso de estudio, este problema solo lo plantea la tarea llamada *Cambiar divisa en todas las facturas*. Esta tarea está modelada en OO-Method como una transacción global, ya que su funcionalidad afecta a un conjunto de instancias de la clase *Factura.*

9.2.4. El sistema no muestra mensajes de advertencia dependiendo de cierta condición

Hay ciertas reglas de negocio que aconsejan lanzar un mensaje de aviso al usuario antes de ejecutar el servicio. El mensaje va por tanto asociado a una condición similar a la definida en el apartado de precondiciones de OO-Method. En el caso de estudio esta funcionalidad se podría utilizar para la tarea *Emitir una factura*.

La tarea Emitir factura se utiliza muy frecuentemente, sin embargo, de todas las facturas emitidas, la mayoría no supera los 10.000 euros. Por lo tanto, aquella factura que supere dicha cantidad, puede que sea un error. Además, la ejecución de este servicio es irreversible y

crítica para el usuario, por lo tanto se debería solicitar confirmación al usuario si esta condición se cumple. Con el actual Modelado Conceptual de OO-Method el modelar este mensaje de aviso es imposible.

9.3. Solución al aplicar el patrón Feedback

Una vez presentadas los problemas detectados en las tareas del caso de estudio, en este apartado se presentará cómo mejoraría el sistema al aplicar la funcionalidad de dichos patrones.

El resultado obtenido es un prototipo de cómo quedaría el sistema de Aguas de Bullent si el Modelo Conceptual de OO-Method incorporara los patrones de la familia *Feedback*. El analista solo tendría que modelarlos y sería el Compilador de Modelos el encargado de generar el código que reflejara sus funcionalidades. Con este ejemplo, se puede observar la mejora de usabilidad de las aplicaciones generadas con OO-Method una vez se hayan incorporado los patrones de usabilidad en el Compilador de Modelos. Es importante resaltar que todas estas mejoras se conseguirían de forma automática gracias al Compilador de Modelos.

A continuación se presenta para cada uno de los patrones de la familia Feedback cual es la solución que aporta a los problemas detectados.

9.3.1. System Status Feedback

Como se ha explicado en la sección 9.2.1, el servicio *Pasar la factura a batch* no informa al usuario de su correcta ejecución, y el usuario debería visitar otras ventanas para verificar que el servicio ha tenido éxito. Aplicando el patrón System Status Feedback sobre dicho servicio, se informa al usuario del éxito o fracaso de la ejecución. Los pasos del sistema aplicando este patrón serían los siguientes:

1. El usuario selecciona la factura que desea pasar a batch y pulsa sobre el botón *Pasar factura a batch* en la interfaz, mostrada en la Figura 38.
2. A continuación se solicita al usuario que confirme la ejecución del servicio *Pasar factura a batch* mediante la interfaz mostrada en la Figura 47. Obsérvese que es solo una confirmación ya que en este caso la interfaz no necesita que el usuario introduzca ningún argumento, el argumento del servicio es el propio objeto (THIS).

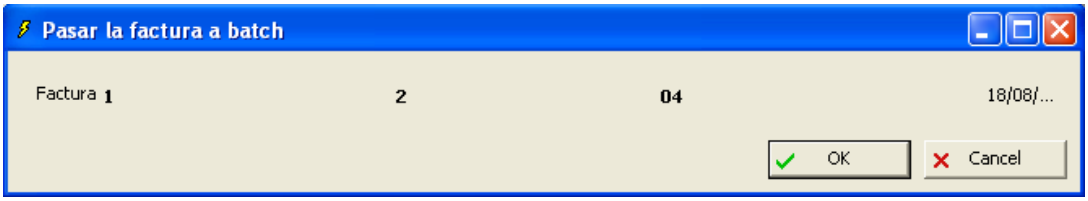

Figura 47. Confirmación para *pasar la factura a batch*

3. Si la ejecución de la acción ha tenido éxito, se muestra un mensaje al usuario informándole que se ha ejecutado correctamente.

Figura 48. Información al usuario del éxito del servicio *Pasar factura a batch*

De esta forma el usuario puede saber si la factura se ha pasado a batch correctamente sin necesidad de buscar la información en la propia ventana o incluso navegando a otra.

Una manera menos molesta de avisar al usuario de que el servicio se ha ejecutado correctamente es mostrar el icono en una esquina de la ventana principal. La selección de la opción de mostrar la ventana modal o este icono se hace en el Modelado Conceptual.

En caso de que el servicio no se haya podido ejecutar debido a que se intenta ejecutar el servicio en un estado no válido (según lo definido en el Modelo Dinámico), se mostraría al usuario una ventana de error donde se informaría de las causas de este error. El texto que se mostraría sería un texto genérico generado por el Compilador de Modelos o el texto que haya introducido el analista en el modelado del patrón System Status Feedback. Si se opta por la primera opción, no se resuelve el problema de poca precisión en los mensajes de error y tendríamos el mismo problema que el planteado en la sección 9.2.1. Por esta razón es aconsejable que el analista introduzca a mano todos los mensajes de error y no dejar esta tarea al Compilador de Modelos.

9.3.2. Interaction Feedback

Según la sección 9.2.2, con el actual Compilador de Modelos, el usuario solo es informado por el puntero del ratón de que su petición de ejecución de un servicio está siendo atendida. Además, si no se informa correctamente al usuario de que sus peticiones están siendo atendidas, éste puede volver a lanzar el mismo servicio varias veces antes de que termine de ejecutarse la primera petición creyendo que sus peticiones no se atienden. Al incorporar en el Compilador de Modelos la funcionalidad del patrón Interaction Feedback, el servicio *Cambiar la dirección de destino*, por ejemplo, quedaría de la siguiente forma:

1. El usuario selecciona la factura a la que desea cambiarle la dirección de destino. Esta selección se hace sobre la ventana de listar facturas, mostrada en la Figura 39. Una vez seleccionada la factura, se pulsa sobre el botón de acción *Cambiar la dirección de destino* para iniciar la ejecución del servicio.
2. Se muestra un formulario como el de la Figura 40 donde el usuario puede seleccionar la factura sobre la que se aplicará el cambio de la dirección de destino (sino lo ha hecho ya en la pantalla donde se listan las facturas) y la nueva dirección de destino.
3. Una vez introducidos los valores en los campos, el usuario pulsa el botón *OK.* Entonces se deshabilitan los campos editables y el botón de *OK,* tal y como muestra la Figura 49. Al finalizar la ejecución, la ventana se cerrará.

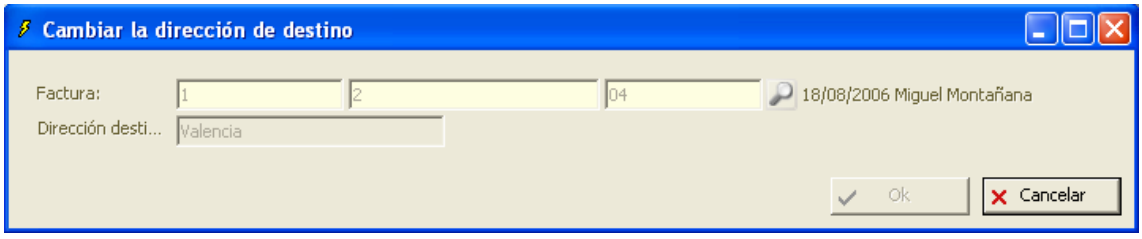

Figura 49. Interfaz de cambiar la dirección de destino deshabilitada

9.3.3. Progress feedback

Tal y como se muestra en la sección 9.1.3, el servicio *Actualizar divisa en todas las facturas* es una acción que requiere más de dos segundos para su ejecución. Aplicando el patrón de Progress feedback sobre este servicio, se mostraría al usuario una barra de progreso en la cual se reflejara el tiempo que resta para que el servicio termine de ejecutarse.

1. El usuario abre la ventana con la lista de facturas usando el menú *Facturación*\ *Factura.* La ventana donde se listan las facturas es como la mostrada en la Figura 41.
2. El usuario selecciona la divisa que desea cambiar y la nueva divisa por la cual se desea cambiar en la Figura 42.
3. Una vez el usuario haya seleccionado la divisa vieja a modificar y la nueva divisa por la que desea modificar la vieja, pulsa sobre el botón de *OK* y se lanza la ejecución del servicio. Al aplicar el patrón Progress Feedback se muestra un formulario como el siguiente donde se puede ver el progreso de la ejecución.

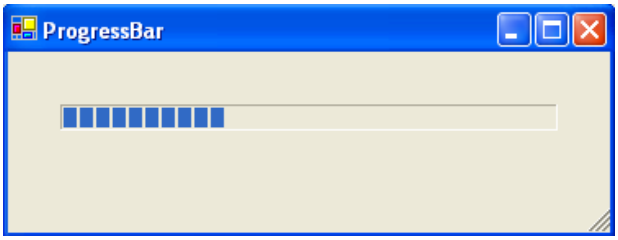

Figura 50. Interfaz que implementa la barra de progreso

9.3.4. Warning

El uso de este patrón hace posible que se puedan definir mensajes de aviso dependiendo de cierta condición, resolviendo el problema que se ha planteado en la sección 9.2.4. Para el caso de estudio propuesto, se podría modelar este patrón en el servicio *Emitir factura.* Se definiría con la condición de que se mostrara el mensaje cuando el importe de la factura fuera superior a 10.000 euros. De esta forma, el sistema generado avisará cada vez que el usuario emita una factura con un precio de más de 10.000 euros. Una vez el Compilador de Modelos incorpore la funcionalidad del patrón Warning*, los* pasos que se ejecutarían en el sistema serían:

1. El usuario abre la ventana para visualizar el conjunto de facturas existentes en el sistema, tal y como muestra la Figura 43.
2. El usuario selecciona de la lista, la factura que desea emitir y pulsa el botón *Emitir la factura.*
3. Se muestra una ventana como la de la Figura 44 en la que el usuario debe introducir la fecha de emisión y seleccionar la factura (si no ha sido previamente seleccionada). Una vez introducidos estos argumentos, el usuario debe pulsar el botón *OK* para iniciar la ejecución del servicio.
4. Al pulsar en emitir, el sistema comprueba que el importe de la factura es superior a 10.000 euros y por tanto muestra la siguiente advertencia al usuario.

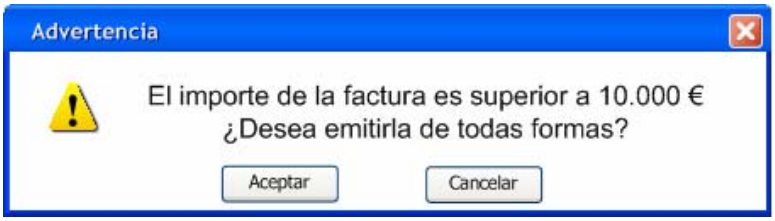

Figura 51. Advertencia al usuario para *Emitir factura*

Si el usuario pulsa sobre el botón *Aceptar,* el servicio se ejecutará a pesar de la advertencia mostrada. En caso de que el usuario pulse sobre *Cancelar*, el servicio no se ejecutará y se volverá a la pantalla de listado de facturas (Figura 43).

9.4. Medición de la usabilidad del sistema

Una vez presentadas las mejoras que aportan cada uno de los patrones de la familia Feedback al aplicarlos a OO-Method, se va a medir la usabilidad del sistema Aguas de Bullent. Tal y como se ha comentado anteriormente, para facilitar la comprensión del método de evaluación, el proceso solo va a utilizar las cuatro tareas que forman nuestro caso de estudio.

La medición de la usabilidad está basada en el método de evaluación planteado a lo largo de este documento y que utiliza el Modelo de Usabilidad propuesto, que fue definido en la Tabla 2. De dicho modelo solo se medirán aquellos atributos que sean medibles en la fase de modelado conceptual. Para hacer la medición de dichos atributos se seguirán los pasos siguientes:

1. **Aplicar métricas:** cada uno de los atributos que forman el Modelo de Usabilidad tiene definida una o varias fórmulas con las cuales calcular su valor de usabilidad. En esta fase se aplican las fórmulas con el fin de obtener el valor de usabilidad del atributo.
2. **Aplicar indicadores:** cada uno de los valores obtenidos mediante las fórmulas se debe interpretar utilizando la tabla de indicadores definida en la Tabla 4. De esta forma se dota de significado a cada uno de los valores numéricos obtenidos mediante las fórmulas.
3. **Agregar:** cada uno de los atributos pertenece a una subcaracterística en el Modelo de Usabilidad propuesto. Es decir, cada subcaracterística está formada por varios atributos. Una vez obtenido el valor de usabilidad de estos atributos, se deben agrupar los valores obtenidos para calcular el valor de usabilidad de las subcaracterísticas, que a su vez deben agrupar sus valores de usabilidad para obtener el valor de la característica usabilidad.

Estos tres pasos se van a aplicar a todas las tareas que forman el caso de estudio. Este proceso nos dará una aproximación de la usabilidad real que tiene el sistema generado.

9.4.1. Aplicar métricas

En esta fase, a partir del Modelo Conceptual del sistema Aguas de Bullent, se van a aplicar todas las fórmulas que aparecen en el Anexo I. Los valores que se obtendrán serán valores numéricos a los que se les proporcionará significado en la próxima fase.

Los datos extraídos de las cuatro tareas que forman el caso de estudio son los que se muestran a continuación. Todos estos valores se han obtenido a partir del Modelo Conceptual de las cuatro tareas del caso de estudio. Estos datos serán los utilizados posteriormente en las fórmulas para obtener el valor de la usabilidad.

- Número de atributos= 50
- Número de servicios= 4
- Número de argumentos del servicio *Pasar la factura a batch*=1
- Número de argumentos del servicio *Cambiar la dirección de destino*=2
- Número de argumentos del servicio *Cambiar divisa en todas las facturas*=2
- Número de argumentos del servicio *Emitir factura*=2
- Número de argumentos con valor inicial=1
- Número de argumentos objeto valuados=3

- Número de grupos de argumentos=4
- Número de patrones de Introducción definidos=0
- Número de patrones de Selección Definida definidos=0
- Número de patrones de Información Suplementaria=0
- Número de IIUs=0
- Número de PIUs=1
- Número de SIUs=4
- Número de patrones de Acción=1
- Número de Elementos de acción=4
- Número de patrones de Navegación=1
- Número de Elementos de navegación=3
- Número de patrones de Criterio de Ordenación=1
- Número de patrones de Filtro=1
- Número de Variables de filtro=10
- Número de atributos del Conjunto de Visualización=18
- Número de Filtros con alias=1
- Número de Criterios de Ordenación con alias=0
- Número de Servicios con alias=4
- Número de atributos con alias=50
- Número de Mensajes de error =2. Uno con longitud 12 y otro con 25
- Número de Mensajes de Estado de tipo texto definidos por el analista=1
- Confirmación de ejecución por iconos
- Número de Mensajes de Estado ocultos=0
- Número de Barras de Progreso ocultas=0
- Número de Mensajes de Aviso=1

A partir de estos datos, se pueden aplicar las fórmulas definidas en la sección 4.2.1. Es importante remarcar que algunas de estas fórmulas no obtienen valores significativos, por ejemplo algunos porcentajes obtienen resultados del 100%. Estos resultados no son frecuentes a la hora de aplicar las métricas a sistemas reales. Esto es debido al pequeño tamaño del caso de estudio. Las cuatro tareas solo dependen de una clase (*Facturas*) y por tanto muchas de las primitivas de modelado utilizado no aparecen en un número significativo, lo cual lleva a que el valor de algunas fórmulas sea 0 o 100%.

- **Facilidad de aprendizaje**
  - Completitud de documentación: CDD= $\frac{65}{70} = 0{,}92$
  - Determinación de la acción : DA1= $\frac{5}{6} = 0{,}83$ DA2= $\frac{0}{3} = 0$
  - Realimentación del estado del sistema: RES(ER)1= $\frac{1}{4} = 0{,}25$

    RES(OK y ER)2= $\frac{4}{4} = 1$

    RES(OK)1 no tiene ningún valor porque todos los mensajes de confirmación se muestran mediante iconos.

- Realimentación de progreso: RDP= $\frac{4}{4} = 1$
- Aviso: AV= $\frac{1}{4} = 0{,}25$

- **Comprensibilidad**
  - Agrupación por disposición: APD= $\frac{7}{4} = 1{,}75$
  - Densidad de la información: DI1= $\frac{7}{4} = 1{,}75$ DI2= $\frac{4}{1} = 4$ DI3= $\frac{18}{1} = 18$ DI4= $\frac{3}{1} = 3$ DI5= $\frac{1}{1} = 1$ DI6= $\frac{10}{1} = 10$ DI7= $\frac{1}{1} = 1$
  - Significación del etiquetado: SDE= $\frac{50}{50} = 1$
  - Completitud de valores iniciales: CVI= $\frac{1}{7} = 0{,}14$
  - Calidad de los mensajes: CDM= $\frac{37}{2} = 18{,}5$
  - Navegabilidad: NV= $\frac{1}{1} = 1$
- **Operabilidad**
  - Consistencia de orden: CO1= $\frac{1}{1} = 1$ CO2= $\frac{1}{1} = 1$ CO3= $\frac{5}{5} = 1$
  - Consistencia de etiquetado: CE1= $\frac{1}{1} = 1$ CE2= $\frac{1}{1} = 1$ CE3= $\frac{1}{1} = 1$ CE4= $\frac{1}{1} = 1$
  - Prevención de errores: PDE= $\frac{0}{3} = 0$

9.4.2. Aplicar indicadores

Una vez obtenidos los valores de usabilidad mediante las fórmulas, se deben interpretar dichos valores. Para ello se utilizan los indicadores definidos en el Anexo II. Dependiendo del rango entre el que se encuentre cada uno de los valores numéricos de las fórmulas, se le asignará a cada fórmula el valor Muy Bueno (MB); Bueno (B); Regular (R); Malo (M); Muy Malo (MM). Estos resultados se agruparán posteriormente para obtener el valor de usabilidad de los atributos y de las subcaracterísticas.

Los valores del caso de estudio al aplicar la tabla de indicadores son:

CDD=B	NV=MB
DA1=R	CO1=MB
DA2=MM	CO2=MB
APD=MB	CO3=MB
DI1=MB	CE1=MB
DI2=MB	CE2=MB
DI3=MM	CE3=MB
DI4=MB	CE4=MB
DI5=MB	PDE=MM
DI6=MM	RES(ER)1=MM
DI7=MB	RES(OK)2=MB
SDE=MB	RES(ER)2=MB
CVI=R	RDP=MB
CDM=B	AV=R

De estos valores se puede deducir que los atributos de la subcaracterística *Facilidad de aprendizaje* no han obtenido valores muy buenos en la medición de la usabilidad. En cambio, los atributos de la subcaracterística *Comprensibilidad* han obtenido puntuaciones muy altas que desembocarán, al aplicar la agregación, en un buen valor de usabilidad para la subcaracterística. Los mejores resultados se han obtenido en los atributos de la subcaracterística *Operabilidad*. Estos valores tan buenos de *Operabilidad* se deben a que las fórmulas de sus atributos están pensadas para utilizar un gran número de clases, y en el caso de estudio solo se ha centrado en la clase *Factura*. Sin embargo, con la aplicación de estas fórmulas se alcanza el objetivo principal de esta sección que es el de mostrar mediante un ejemplo la utilización práctica de las fórmulas en un caso de estudio.

9.4.3. Agregar

El último paso consiste en agregar los valores obtenidos mediante los indicadores. En un primer paso se agregan los valores de las fórmulas que definan atributos dependientes de más de una fórmula. Una vez están agregados todos los valores de los atributos, se deben agregar los valores de los atributos para calcular el valor de usabilidad de las subcaracterísticas. El método de agregación es el descrito en la sección 4.2.3

Para facilitar la operación de agregar, es recomendable construir un árbol con las fórmulas, atributos y subcaracterísticas como el mostrado en la Figura 11. En las hojas del árbol están las fórmulas, cuyos nodos padre son los atributos, que a su vez tienen como nodos padre a las subcaracterísticas, que se agrupan todas en la raíz usabilidad. Además de estas restricciones jerárquicas, existe la restricción de que cada nodo padre solo puede tener dos nodos hijo. Esta restricción es necesaria porque el método de agregación toma valores de dos en dos. Para que esta restricción se cumpla se deben construir nodos intermedios que vayan tomando nodos hijo en grupos de dos. Una vez construido el árbol, se empieza a agregar nodos desde las hojas hasta la raíz.

El árbol del caso de estudio se ha dividido en varias secciones, ya que su tamaño es mayor al de una página. Hay una sección por cada una de las subcaracterísticas cuyos atributos se pueden medir en el Modelado Conceptual. Posteriormente todas estas secciones de árbol se agrupan en otro árbol para calcular el valor de la Usabilidad del sistema.

La notación utilizada para la nomenclatura de los árboles ha sido el nombrar a los nodos hoja con el mismo nombre que las fórmulas (o el mismo nombre que el atributo si éste solo depende

de una fórmula). Los nodos intermedios son el nombre de atributos (para el caso de que éstos dependan de varias fórmulas) o el nombre de la subcaracterística a la que pertenecen junto a un número utilizado para diferenciar los nodos entre sí. Por último el nodo raíz de cada sección se corresponde con el nombre de la subcaracterística que se está representando con esa sección del árbol.

Las secciones son las siguientes:

- Facilidad de aprendizaje:

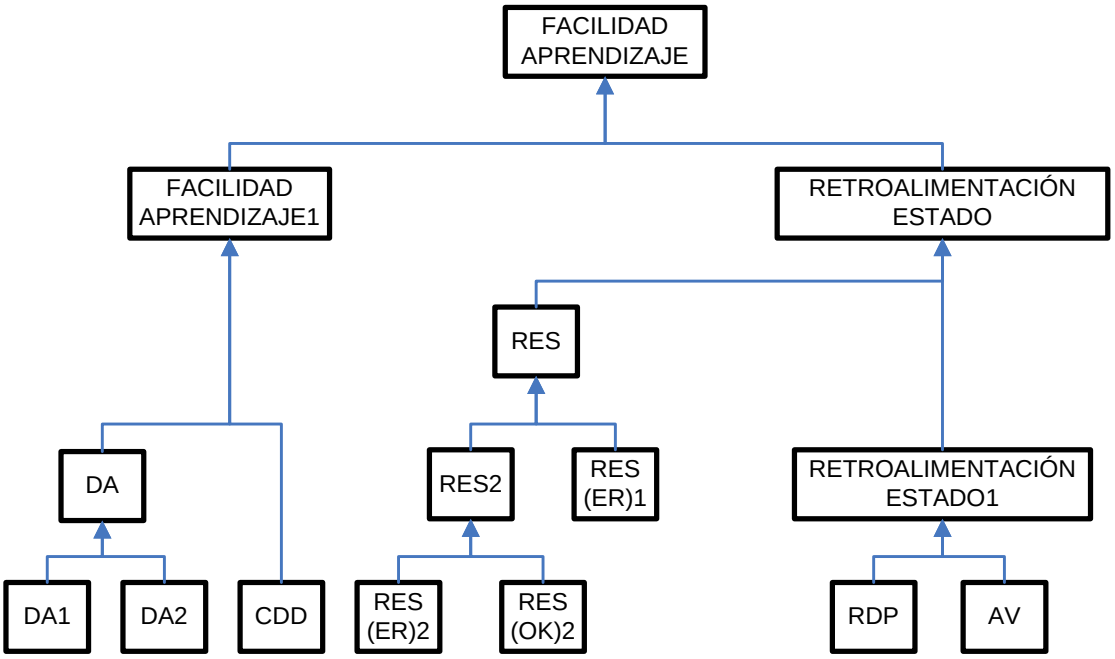

Figura 52. Árbol de agregación para la subcaracterística *Facilidad de aprendizaje*

- Comprensibilidad:

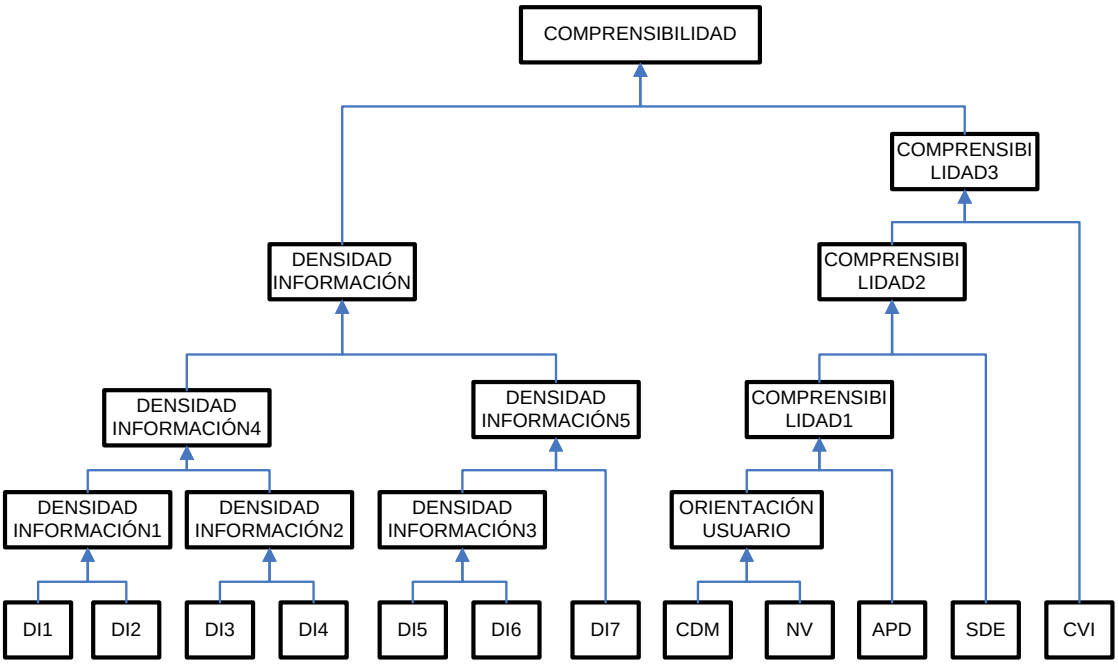

Figura 53. Árbol de agregación para la subcaracterística *Comprensibilidad*

- Operabilidad:

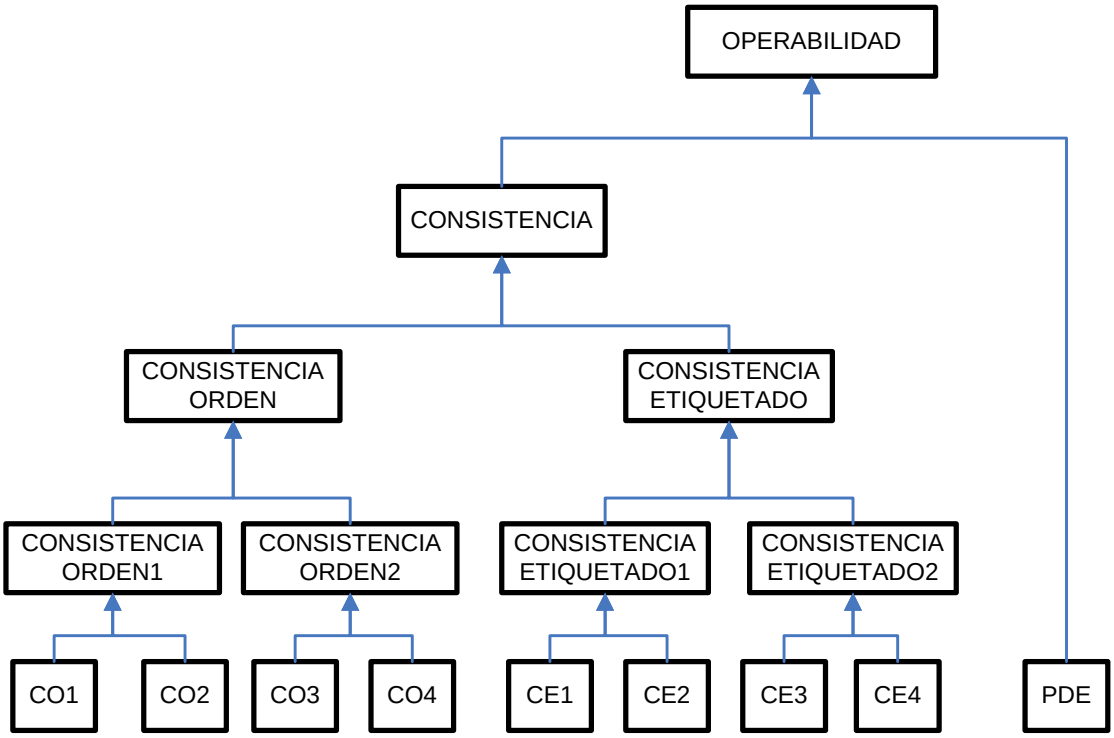

Figura 54. Árbol de agregación para la subcaracterística *Operabilidad*

- Usabilidad: por último se agregan todas las subcaracterísticas en el nodo raíz, llamado usabilidad.

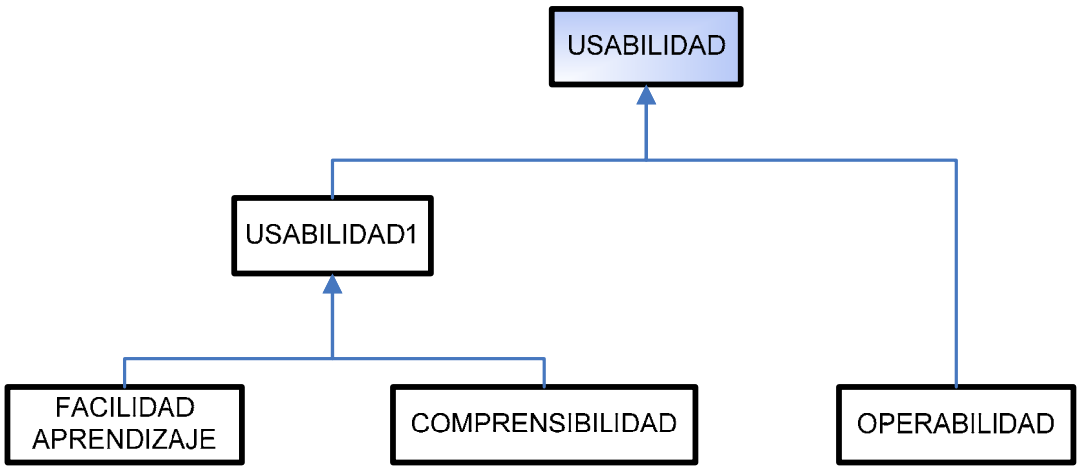

Figura 55. Árbol de agregación para *Usabilidad*

A partir de estos árboles se pueden aplicar las reglas de agrupación definidas en la sección 4.2.3.1. Se empieza agrupando el valor de las fórmulas, para seguir con el de los atributos y finalmente con el de las subcaracterísticas. Tal y como se ha comentado anteriormente, la agrupación va de las hojas del árbol hacia la raíz. Por ejemplo, en el árbol de la Figura 52 donde se representa la subcaracterística *Facilidad de aprendizaje*, se empezaría agrupando los nodos hoja DA1 y DA2. DA1=R y DA2=MM, por lo tanto según las reglas de agrupación, el

nodo padre DA tiene el valor MM. A continuación se muestran los árboles de agrupación asignando valor a sus nodos.

- Facilidad de aprendizaje: a la hora de hacer la agregación de esta subcaracterística se obtienen valores muy dispares en sus atributos. El atributo *Determinación de la Acción* es el que obtiene peor resultados (Muy Mal). En cambio el conjunto de atributos llamado *Retroalimentación de Estado* que engloba a los atributos *Realimentación de Estado, Realimentación de Progreso* y *Aviso*, obtiene resultados muy buenos, (Muy Bueno). El resultado final para la subcaracterística es el de Regular, debido a la disparidad de valores obtenidos entre sus atributos.

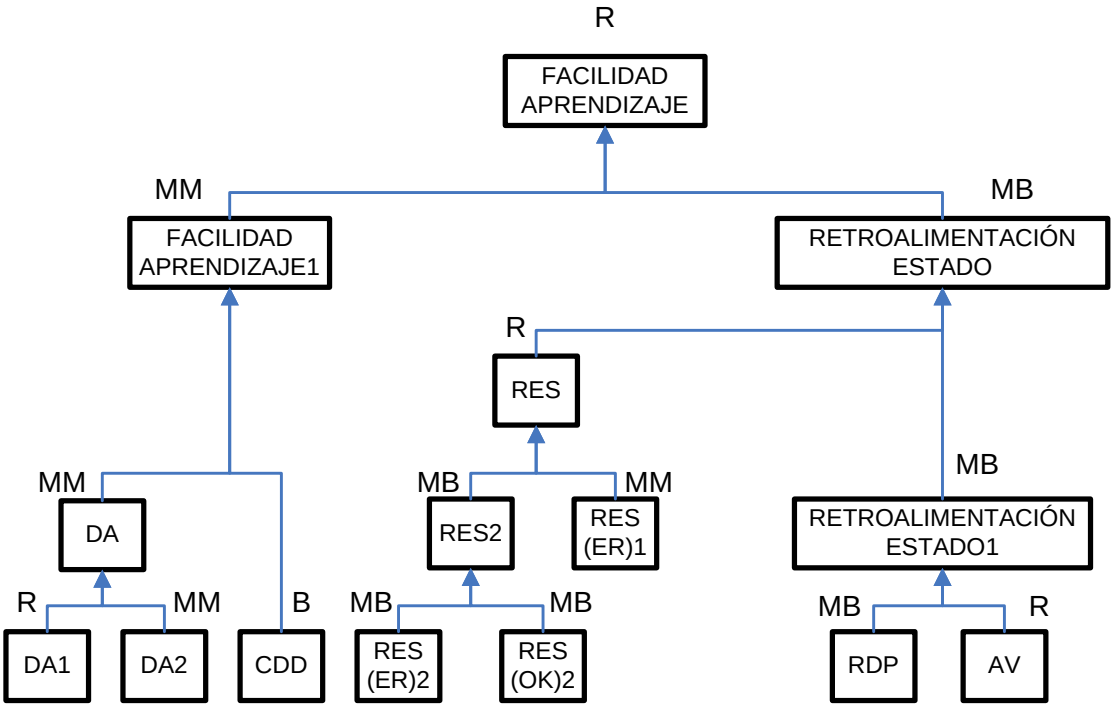

Figura 56. Resultado al agregar la subcaracterística *Facilidad de aprendizaje*

- Comprensibilidad: el valor para esta subcaracterística es el de Muy Bueno, tal y como muestra la Figura 57. La mayoría de los atributos obtienen el valor MB (Muy Bueno) que al hacer la agrupación se obtiene como resultado dicho valor para el caso de la subcaracterística *Comprensibilidad*.

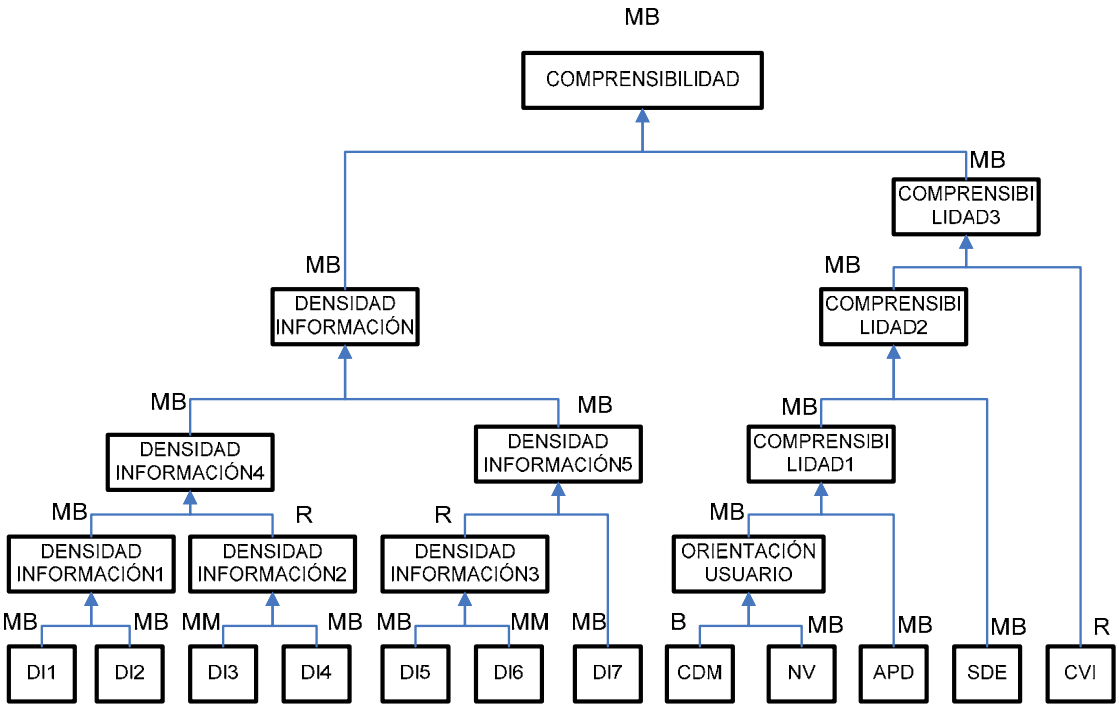

Figura 57. Resultado al agregar la subcaracterística *Comprensibilidad*

- Operabilidad: el valor que obtiene esta subcaracterística es Regular (R). Este resultado, que indica que esta subcaracterística no es muy usable, se obtiene por el resultado Muy Malo (MM) obtenido en el atributo *Predicción de errores*. Aunque el resto de atributos *Consistencia de Orden* y *Consistencia de Etiquetado* tienen valores Muy Buenos (MB), el resultado negativo de *Predicción de errores* implica que el resultado final de la subcaracterística sea Regular.

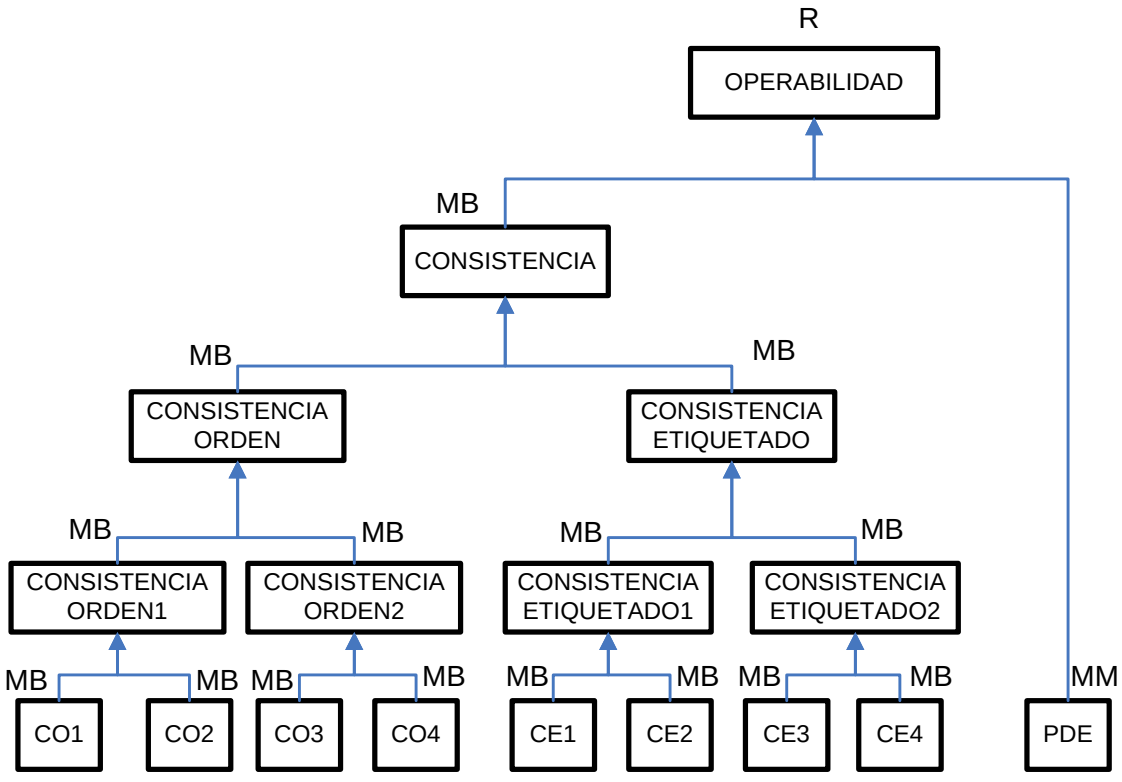

Figura 58. Resultado al agregar la subcaracterística *Operabilidad*

- Usabilidad: una vez están calculados los valores de usabilidad de todas las subcaracterísticas, solo resta agrupar dichos valores para obtener la usabilidad del sistema. Tal y como muestra la Figura 59, el valor obtenido es Muy Bueno (MB). A pesar de los valores Regulares obtenidos en las subcaracterísticas *Facilidad de Aprendizaje* y *Operabilidad*, el valor Muy Bueno de la subcaracterística *Comprensibilidad* es el que ha propiciado que el valor de la *Usabilidad* del sistema sea Muy Bueno.

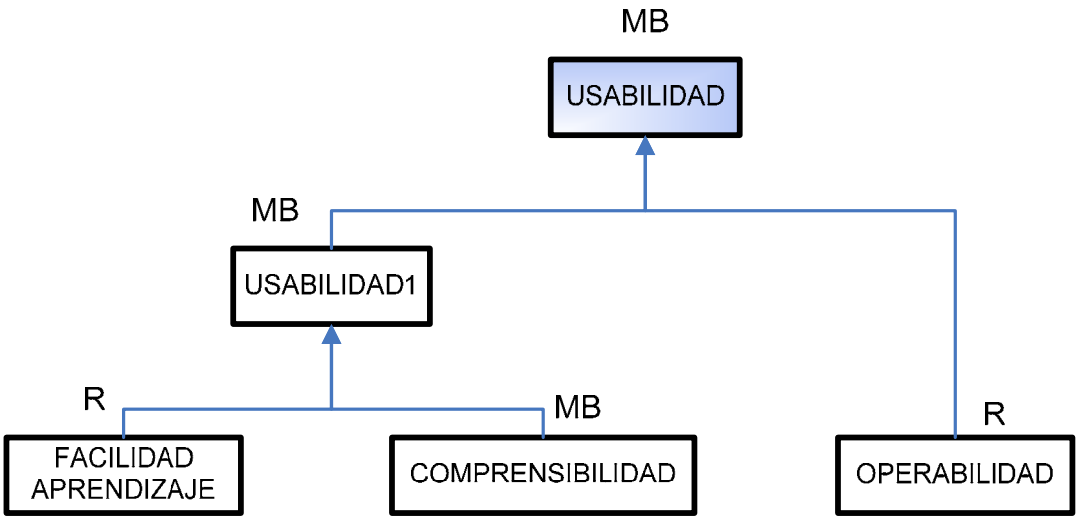

Figura 59. Resultado al agregar las subcaracterística para obtener el valor de la *Usabilidad* del sistema

Por lo tanto, se puede afirmar que según el método de evaluación de la usabilidad propuesto en este documento, el sistema Aguas de Bullent es un sistema muy usable. A pesar de este resultado tan favorable, ha habido resultados en algunos atributos bastante malos para la usabilidad del sistema. A la hora de aplicar la agregación, estos valores han desaparecido porque había un mayor porcentaje de atributos usables que de atributos no usables. Sin embargo, el modelador de OO-Method puede mostrar una lista de aquellos atributos que han obtenido valores Malos (M) y Muy Malos (MM) para que el analista los mejore si lo considera oportuno. Además de mostrar aquellos atributos que se deben mejorar, el sistema debería indicar cómo alcanzar la mejoría de estos atributos, aunque sea una mera descripción de las primitivas que afectan a estos atributos. Esta descripción guía al analista para mejorar el resultado de usabilidad obtenido.

Para el caso de estudio, los atributos que debería mejorar el analista son los siguientes:

- Determinación de la Acción: este atributo se puede mejorar añadiendo el patrón de Interacción de OO-Method llamado Información Suplementaria a cada uno de los argumentos objeto valuados.
- Densidad de la Información: para mejorar este atributo son dos las acciones que se pueden hacer.
    - Disminuir el número de atributos del Conjunto de Visualización asociado a la Unidad de Interacción de Población de la clase Factura.
    - Disminuir el número de Variables de Filtro del filtro asociado a la Unidad de Interacción de Población de la clase Factura.
- Prevención de Errores: este atributo se puede mejorar añadiendo el patrón de Interacción de OO-Method llamado Selección Definida a aquellos atributos de la clase Factura de tipo string cuya longitud sea menor que cuatro. En concreto a los atributos *Serie, Sufijo* y *DigControl.*

- Realimentación del Estado del Sistema: para mejorar este atributo, el analista debería definir explícitamente los mensajes que se mostrarán al usuario cuando éste solicite un servicio en un estado del objeto desde el cual la ejecución del servicio no esté permitida (error debido a una transición entre estados no definida).

Tal y como se puede apreciar, mediante este método de evaluación automática, el analista puede predecir en muy poco tiempo lo usable que será el sistema generado y los cambios que debe hacer si desea mejorar la usabilidad del sistema. Todo el proceso que en este caso de estudio se ha realizado a mano, al incorporarlo a OO-Method sería aplicado automáticamente por el mismo modelador de OlivaNOVA. De forma que el analista no debe conocer nada de este proceso, ni las fórmulas, ni los indicadores, ni cómo agrupar los atributos y subcaracterísticas.

A partir de la información proporcionada por este método de evaluación, el analista puede mejorar el Modelo Conceptual antes de generar el código para que el sistema sea más usable de lo que ya lo es.

10. CONCLUSIONES

En este documento se ha presentado un mecanismo de medición de la usabilidad basado en un Modelo de Usabilidad publicado anteriormente por Abrahao et al [2][11]. En este Modelo de Usabilidad están reflejados todos los aspectos que influyen en la usabilidad de los sistemas. Este modelo está formado por dos componentes: *subcaracterísticas*, que representan un aspecto de la usabilidad; *atributos*, que son los aspectos medibles de las subcaracterísticas.

De todo el Modelo de Usabilidad, el mecanismo de evaluación propuesto solo se centra en aquellos atributos medibles desde las primitivas de los Modelos Conceptuales, dejando de lado los atributos medibles a través del Compilador de Modelos (siempre se obtiene el mismo resultado con ellos) y aquellos medibles en la Aplicación Final (atributos subjetivos del usuario). Dependiendo del método de producción software, los atributos del Modelo de Usabilidad que formarán parte de la evaluación temprana serán unos u otros, dependiendo de la expresividad del método de producción de software sobre el cual se realice la medición. Si el método de evaluación se aplica a un método de producción de software basado en una implementación a mano, los atributos del Compilador de Modelos se interpretan como aquellos atributos que dependen totalmente de las decisiones del programador.

El Modelo de Usabilidad se utiliza dentro de un método de evaluación de la usabilidad genérico. Al instanciar el método de evaluación para aplicarlo en los Modelos Conceptuales de un generador automático de código basado en modelos, el mecanismo a seguir se divide en tres fases:

1. Definición de métricas: a cada uno de los atributos del Modelo de Usabilidad se le asocia una o varias fórmulas con las cuales medir su usabilidad. Estas fórmulas están basadas en las primitivas de los Modelos Conceptuales del entorno de desarrollo de software sobre el que se realice la medición. Para este documento el entorno elegido ha sido OO-Method.
2. Definición de indicadores: a partir de guías de usabilidad [21], heurísticos [27] y criterios ergonómicos [4] se definen los rangos entre los que pueden oscilar los valores de las métricas para considerar un atributo usable o no usable.
3. Agregación: una vez cada atributo tiene un indicador que muestra cuánto de usable es, se agregan estos valores para calcular el valor de usabilidad de las subcaracterísticas del Modelo de Usabilidad.

Estos pasos a seguir dentro del método de evaluación se han aplicado a un método de generación automática de software a partir de Modelos Conceptuales llamado OO-Method [32]. Sin embargo, estos pasos son aplicables a cualquier otro método de producción de sistemas basado en el Modelado Conceptual. De los tres pasos, el único que varía de un método de producción de sistemas a otro son las métricas. Las métricas dependen de las primitivas conceptuales y cada método suele tener unas primitivas distintas, dependiendo de la expresividad de sus Modelos Conceptuales. Por lo tanto, cada método de producción de sistemas tendrá sus propias métricas.

Como ejemplo práctico de aplicación del método de evaluación de la usabilidad sobre OO-Method, se ha elegido una aplicación real modelada con OlivaNOVA. De esta aplicación solo se ha elegido la parte del Modelo Conceptual correspondiente a cuatro de las tareas que componen el sistema, con el fin de facilitar la comprensión del método. Mediante este caso de estudio se explica la aplicabilidad del método a un sistema real.

Tal y como se ha podido apreciar mediante el caso de estudio, el uso de este mecanismo de evaluación en cualquier entorno de producción de software proporciona un rápido análisis de la usabilidad del sistema, posibilitando al analista mejorar la usabilidad en los Modelos Conceptuales antes de generar la aplicación final. Junto al valor de usabilidad de cada una de las subcaracterísticas y del sistema, se muestra al analista una serie de pasos que debe

realizar si desea mejorar el resultado de usabilidad obtenido. Para aquellos atributos que no hayan obtenido un buen valor de usabilidad, el modelador debe crear un informe donde se recojan estos atributos junto con las acciones que debe realizar el analista para mejorarlos. El informe ayuda al analista en la tarea de mejorar la usabilidad, guiándole en las primitivas que debe modificar para mejorar la usabilidad del sistema. Sin embargo, a pesar de todas estas ventajas, este método de evaluación tiene un inconveniente, y es la ausencia de atributos subjetivos en las métricas de medición.

Los atributos subjetivos no se pueden medir mediante los Modelos Conceptuales porque a nivel de modelado no se pueden medir los gustos de los usuarios. Son atributos que dependen exclusivamente del usuario y solo se pueden medir mediante un test de usuarios sobre la aplicación final. Aun con esta carencia, el método de evaluación propuesto es un buen indicador de lo usable que será el sistema cuando se implemente. Los atributos que son totalmente subjetivos son pocos y su peso no es determinante para el cálculo de la usabilidad final. Con el método propuesto, el analista puede predecir los problemas de usabilidad que se encontraría al realizar los test de usuarios y le ofrece la posibilidad de repararlos rápidamente. Además, una vez esté el sistema generado y el método de evaluación indique que el sistema es usable, la medición de la usabilidad se puede complementar con un test de usuario que mida la usabilidad de los atributos subjetivos.

Uno de los trabajos futuros sería precisamente el de comparar los resultados obtenidos mediante el mecanismo de evaluación propuesto con los resultados de usabilidad obtenidos mediante un test de usuarios. De esta forma se podría confirmar si el valor de usabilidad obtenido mediante el mecanismo propuesto se corresponde con la usabilidad percibida por parte del usuario en su interacción con la aplicación final. Además de esta comparación, con los test de usuario se podrían refinar los rangos de los indicadores para acercarlos más al mundo real, ya que estos rangos se han establecido a partir de estudios teóricos extraídos de la literatura. Además, como trabajo futuro se podría establecer un mecanismo para que fuera el propio analista el que estableciera los rangos de los indicadores dependiendo del tipo de aplicación que se esté modelando. Podrían existir también plantillas con indicadores predeterminados que el analista seleccionara a la hora de hacer la evaluación. El analista elegiría una plantilla determinada dependiendo del tipo de aplicación a desarrollar y del tipo de usuario al cual va a ir dirigida.

Además de la contribución del método de evaluación basado en los Modelos Conceptuales, este trabajo ha presentado un conjunto de mejoras sobre el Modelo de Usabilidad ya existente previamente [2][11][1]. En este documento se han presentado atributos nuevos que se han incorporado al Modelo de Usabilidad. Estos atributos se han derivado de una serie de patrones de usabilidad definidos en un proyecto llamado STATUS *(SofTware Arquitectures That support USability)* [42]. En este proyecto se definieron un conjunto de patrones que debían ser incorporados en la arquitectura de los sistemas para que éstos fueran considerados como usables. A partir de estos patrones se han deducido nuevos atributos que deben enriquecer el Modelo de Usabilidad.

Para poder medir en OO-Method los nuevos atributos incorporados al Modelo de Usabilidad, se han planteado una serie de cambios a lo largo de todo el proceso OO-Method. Estos cambios afectan tanto a la fase de Modelado Conceptual como a la de generación de código, llevada a cabo esta última por el Compilador de Modelos. Son varios los modelos que se deben modificar para albergar el modelado de los nuevos atributos de usabilidad. Aunque estos atributos derivan de patrones de interacción con el usuario, no basta solo con enriquecer mediante nuevas primitivas el Modelo de Interacción de OO-Method, ya que hay muchos aspectos de usabilidad que están fuertemente relacionados con la funcionalidad del sistema y por tanto deben ser modelados desde los modelos que representan dicha funcionalidad.

Junto con los cambios a nivel conceptual para incorporar los nuevos atributos, también se ha detallado en este documento cómo incorporar los nuevos atributos en el resto del proceso de generación de código. Es decir, se detallan los cambios que se deben aplicar en el Compilador

de Modelos para que el código refleje la funcionalidad de los patrones de usabilidad de los que se han derivado los atributos. De esta forma, queda definido el método para incorporar los nuevos atributos en todo el proceso de generación de código de OO-Method, desde el Modelado Conceptual hasta el código final que implementa la aplicación. Una vez incorporadas las primitivas que soportan el modelado de los atributos derivados de los patrones de usabilidad, las aplicaciones generadas con OO-Method serán más usables de lo que ya lo son actualmente

Para describir todos los cambios que se deberían realizar en OO-Method con el objetivo de soportar las primitivas que representen la funcionalidad de los patrones de usabilidad, este trabajo se ha centrado en detallar los cambios generados por la familia del patrón Feedback. El elegir solo una familia ha favorecido que se pueda profundizar en los detalles de cómo llevar a cabo los cambios sobre OO-Method. Otra de las líneas de trabajo futuro es el enriquecer el Modelo de Usabilidad con el resto de patrones de STATUS. Además, habría que modificar del mismo modo que se ha hecho para la el patrón Feeback, el Modelado Conceptual y el Compilador de Modelos de OO-Method.

Una vez incorporada a OO-Method toda la funcionalidad de los patrones de usabilidad, sería interesante hacer una validación empírica para demostrar que las aplicaciones generadas son más usables que las que generaba OO-Method antes de incluir las nuevas primitivas de modelado.

Por último, otro de los trabajos futuros sería el dotar a los Modelos Conceptuales de OO-Method de más primitivas conceptuales para que la mayoría de los atributos que se han clasificado actualmente como medibles por el Compilador de Modelos, sean medibles por los Modelos Conceptuales. La mayoría de los atributos del Modelo de Usabilidad están actualmente clasificados como dependientes del Compilador de Modelos de OO-Method porque el analista no puede modelar ningún aspecto de esos atributos. Esto implica que su valor de usabilidad sea siempre el mismo porque el código generado es siempre el mismo, y no necesariamente debe ser un buen valor de usabilidad. Cuantos más atributos pertenezcan al tipo de atributos medibles por los Modelos Conceptuales, más usables serán las aplicaciones generadas por OO-Method, porque se podrán ajustar mejor a las demandas de los usuarios.

11. REFERENCIAS

[1] Abrahao, S., Insfrán, E. (2006). Early Usability Evaluation in Model Driven Architecture Environments. Sixth International Conference on Quality Software (QSIC'06).pp. 287-294.

[2] Abrahão, S., Ó. Pastor, Olsina, L. (2005). A Quality Model for Early Usability Evaluation. INTERACT05 International COST 294 Workshop on User Interface Quality Models (UIQM), Rome, Italy.pp. 68 - 77.

[3] Bass, L, John, B. E., Kates, J. Achieving Usability Through Software Architecture, TECHNICAL REPORT CMU/SEI-2001-TR-005, March 2001.

[4] Bastien, J. M. C., & Scapin, D. L. (1993). Ergonomic criteria for the evaluation of human-computer interfaces (Report No. 156). Rocquencourt, France: Institut National de Recherche en Informatique et en Automatique.

[5] CARE: http://www.care-t.com Última visita: Jul-2007.

[6] Chung, L., Nixon, B., Yu, E., Mylopoulos, J. (2000). Non-Functional Requirements in Software Engineering. London, Kluwer Academic Publishing.

[7] Condori, N., Abrahao, S., Pastor, O. (2004). Una Experiencia en la Validación Teórica de Métricas de Software. IDEAS 2004, Perú.

[8] Dix, A., Abowd, G., Beale, R. and Finlay, J. (1998), Human-Computer Interaction, Prentice Hall Europe.

[9] Dromey R.G. (1998) Software Product Quality: theory, Model and Practice, Technical report, Software Quality Institute, Griffith University, Nathan, Brisbane, Australia.

[10] Dujmovic, J. J. (1996). A Method for Evaluation and Selection of Complex Hardware and Software Systems. 22nd Int'l Conference for the Resource Management and Performance Evaluation of Enterprise CS.pp. 368-378.

[11] España, S., Pederiva, I., Panach, I., Abrahão, S., Pastor, O. (2006). Evaluación de la Usabilidad en un Entorno de Arquitecturas Orientadas a Modelos. IDEAS 2006, Argentina.pp. 331-344.

[12] Ferré, X., Juristo, N., Moreno, A., Sánchez, I. (2003). A Software Architectural View of Usability Patterns. INTERACT 2003. 2nd Workshop on Software and Usability Cross Pollination: The Role of Usability Patterns, Zürich, Suitzerland.

[13] Fitzpatrick, R. and C. Higgins (1998). Usable Software and Its Attributes: A Synthesis of Software Quality, European Community Law and Human-Computer Interaction Proceedings of HCI on People and Computers XIII.Springer-Verlag: 3-21.

[14] Folmer E., Bosch J. (2004) Architecting for usability: A Survey, Journal of Systems and Software, 70(1) 61-78.

[15] Genero, M. (2001). Defining and Validating Metrics for Conceptual Models. España, Tesis Doctoral, Departamento de Informática, Universidad de Castilla-La Mancha.

[16] Gilb, T. (1976). Software Metrics. Cambridge, MA., Chartwell-Bratt.

[17] Hix, Deborah and H. Rex Hartson (1993), Developing User Interfaces: Ensuring Usability Through Product & Process. New York, NY: John Wiley & Sons.

[18] ISO 9241-11 (1998) Ergonomic requirements for office work with visual display terminals - Part 11: Guidance on Usability.

[19] ISO/IEC 9126-1 (2001), Software engineering - Product quality - 1: Quality model.

[20] Jeffrey, R. (2001). Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests, Wiley; 1 edition.

[21] Jenny, P. (1993). A Guide to Usability. Human Factors in Computing, Addison-Wesley.

[22] Juristo, N., López, M., Moreno, A., Sánchez, I. (2003). Improving software usability through architectural patterns. International Conference on Software Engineering. Workshop "Bridging the Gaps Between Software Engineering and Human-Computer Interaction". Portland, USA.

[23] Keinonen, T. (1998). One-dimensional usability - influence of usability on computers' product preference publication A21. Helsinki.

[24] Leavit, M., Shneiderman, B. (2006). Research-Based Web Design & Usability Guidelines, U.S. Government Printing Office. Web: http://www.usability.gov/pdfs/guidelines.html

[25] McCall J. A., Richards P. K. and Walters G. F. (1977)Factors in software quality, Vols. IIII, Rome Aid DefenceCentre, Italy.

[26] Molina, P. J. (2003). Especificación de interfaz de usuario: de los requisitos a la generación automática. PhD. Departamento de Sistemas Informáticos y Computación. Valencia, Universidad Politécnica de Valencia: 382.

[27] Nielsen, J. (1993). Usability Engineering, AP Professional.

[28] Nielsen, J. (2006). Prioritizing Web Usability, New Riders Press; 1 edition.

[29] Nunes, N. J. y J. F. e. Cunha (2000). "Wisdom: a software engineering method for small software development companies." Software, IEEE 17(5); pp. 113-119.

[30] Olsina, L., Rossi, G. (2002). A Quantitative Method for Quality Evaluation of Web Sites and Applications. IEEE Multimedia Magazine.pp. 20-29.

[31] OMG (2003) MDA Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf. Última visita: Jul-2007.

[32] Pastor, O., Gómez, J., Insfrán, E. Pelechano, V. (2001) The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems, 26(7) 507–534.

[33] Pederiva, I., Vanderdonckt, J., España, S., Panach, I., Pastor, O. (2007). The Beautification Process in Model-Driven Engineer-ing of User Interfaces. INTERACT07, Brasil.

[34] Poels, G., Dedene, G. (1999). DISTANCE: A Framework for Software Measure Construction., Research Report 9937, Dep. Of Applied Economics, Katholieke Universiteit Leuven.

[35] Quispe Cruz, M., Condori Fernández, N. (2007). Requisitos No Funcionales: Evaluando el Impacto de Decisiones. VI Jornada Iberoamericana de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC07), Lima(Peru).
[36] QVT Partners. The QVT Partners Manifesto., 2003 http://www.qvtp.org/downloads/manifesto20030422.pdf., Última visita: Jul-2007.
[37] Radeke, F., Forbrig, P., Seffah, A., Sinning, D. (2006). PIM Tool: Support for Pattern-Driven and Model-Based UI Development. TAMODIA, Belgium.
[38] Rubin, J. (1994). Handbook of Usability Testing, Wiley.
[39] Shneiderman, B. (1998), Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley, Reading, Massachusetts, USA.
[40] Shneiderman, B., Plaisant, C. (2006). Diseño de Interfaces de Usuario. Cuarta Edicion. Estrategias para una Interacción Persona-Computadora Efectiva. Madrid, Addison Wesley.
[41] Spool, J., Scanlon, T., Schroeder, W., Snyder, C., DeAngelo, T. (1999). Web Site Usability: A Designer's Guide, Morgan Kaufmann Publishers, Inc.
[42] STATUS Project: http://is.ls.fi.upm.es/status. Última visita: Jul-2007.
[43] Suppes, P. K. D. M., Luce, R. D., Tversky, A. (1989). Foundations of Measurement: Geometrical, Threshold, and Probabilistic Representations. San Diego, California, Academic Press.
[44] Vanderdonckt, J., Q. Limbourg, et al. (2004). USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, Greece.
[45] Welie, M. v., G. C. v. d. Veer and A. Eliëns (1999). Breaking Down Usability. NTERACT 99, Edinburgh. IOS Press, pp. 613-620.

ANEXO I: MÉTRICAS PARA OO-METHOD

FACILIDAD DE APRENDIZAJE	**Completitud de documentación:** $\forall x \in Atributos \vee Servicios \vee Argumentos \vee PatronIntroduccion \vee SeleccionDefinida \vee IIU \vee PIU \vee SIU \vee MD \vee Acciones \vee Navegaciones \vee CriterioOrdenación \vee Filtro$: $\frac{\sum MensajeAyuda(x)}{\sum x} = CDD$
	Determinación de la acción: $\forall x \in Servicio \vee Filtro \vee Ordenacion : \frac{\sum AliasDist\,\mathrm{int}\,oDeNombreYdeNulo(x)}{\sum x} = DA1$ $\forall x \in InformacionSuplementaria, \forall y \in ArgumentoObjetoValuado : \frac{\sum x}{\sum y} = DA2$
	Realimentación del estado del sistema (OK y error): $\forall x \in MensajeDeEstadoTipoTexto : \frac{\sum DefinidosPorAnalista(x)}{\sum x} = RES1$ $\forall x \in Servicio : \frac{\sum MensajeEstadoNoOculto(x)}{\sum x} = RES2$
	Realimentación de progreso: $\forall x \in Servicio : \frac{\sum Barra\,\mathrm{Pr}\,ogresoNoOculta(x)}{\sum x} = RDP$
	Aviso: $\forall x \in Servicio : \frac{\sum MensajeAvisoDefinido(x)}{\sum x} = AV$
COMPRENSIBILIDAD	**Agrupación por disposición:** $\forall x \in GrupoDeArgumentos(Argumentos) : \frac{\sum_{i=1}^{n} x_i}{\sum x} = APD$
	Densidad de la información: $\forall x \in GrupoDeArgumentos(Argumentos) : \frac{\sum_{i=1}^{n} x_i}{\sum x} = DI1$ $\forall x \in UIP \vee UII : \frac{\sum_{i=1}^{n} ElementosAccion(x_i)}{\sum x} = DI2$

	$\forall x \in UIP \vee UII : \frac{\sum_{i=1}^{n} AtributosDelConjuntoVisualización(x_i)}{\sum x} = DI3$ $\forall x \in ElementoDeNavegación, \forall y \in UIP \vee UII : \frac{\sum x}{\sum y} = DI4$ $\forall x \in UIPconAlMenosUnFiltro, \forall y \in UIP : \frac{\sum x}{\sum y} = DI5$ $\forall x \in VaribleDeFiltro, \forall y \in Filtro : \frac{\sum x}{\sum y} = DI6$ $\forall x \in UIPconAlMenosUnCriterioOrdenacion, \forall y \in UIP : \frac{\sum x}{\sum y} = DI7$
	Significación del etiquetado: $\forall x \in AtributoConAliasDefinido, \forall y \in Atributos \frac{\sum x}{\sum y} = SDE$
	Completitud de valores iniciales: $\forall x \in ArgumentoConValorInicial, \forall y \in Argumento : \frac{\sum x}{\sum y} = CVI$
	Calidad de los mensajes: $\forall xMensajeError : \frac{\sum_{i=1}^{n} Longitud(x_i)}{\sum x} = CDM$
	Navegabilidad: $\forall x \in NavegacionesDefinidas, \forall y \in UIP \vee UII : \frac{\sum x}{\sum y} = NV$
OPERABILIDAD	**Consistencia de orden:** $\forall x \in MismoOrden(Crear, Modificar), \forall y \in ClasesCon(Crear, Modificar) : \frac{\sum x}{\sum y} = CO1$ $\forall x \in AccionConEvento(Crear, Modificar, E\lim inar) : \frac{\sum Ordenados(x)}{\sum x} = CO2$ $\forall x \in ConjuntoCamposAsociadoaUI, \forall y \in UIP \vee UII \vee UIS : \frac{\sum Ordenados(x)}{\sum y} = CO3$
	Consistencia de etiquetado: $\forall x \in DestinoDeNavegaciónYaExistente : \frac{\sum AliasIguales(x)}{\sum x} = CE1$

	$\forall x \in \text{Re}\ ferenciaServicioYaExistente : \frac{\sum AliasIguales(x)}{\sum x} = CE2$ $\forall \text{Re}\ ferenciaAtributoyaExistenteEnConjuntoVis : \frac{\sum AliasIguales(x)}{\sum x} = CE3$ $\forall \text{Re}\ ferenciaYaExistenteEnVarFiltro : \frac{\sum AliasIguales(x)}{\sum x} = CE4$
	Prevención de errores: $\forall x \in Long(ArgumentoTypoString) < 4 : \frac{\sum SelecciónDefinida(x)}{\sum x} = PDE$

ANEXO II: INDICADORES

Medida	MB	B	R	M	MM
CDD	CDD≥.95	.95>CDD≥.85	.85>CDD≥.75	.75>CDD≥.65	CDD <.65
DA1	DA1≥.95	.95> DA1≥.85	.85> DA1≥.75	.75> DA1≥.65	DA1<.65
DA2	DA2≥.95	.95> DA2≥.85	.85> DA2≥.75	.75> DA2≥.65	DA2<.65
APD	APD≤ 15	15< APD ≤20	20< APD ≤25	25< APD ≤30	APD >35
RES (OK)1	RES1≥.85	.85>RES1≥.70	.70>RES1≥.55	.55>RES1≥.40	RES1 <.40
RES (ER)1	RES1≥.95	.95>RES1≥.85	.85>RES1≥.75	.75>RES1≥.65	RES1<.65
RES (OK)2	RES2≥.99	.99>RES2≥.95	.95>RES2≥.90	.90>RES2≥.85	RES2<.85
RES (ER)2	RES2≥.90	.90>RES2≥.80	.80>RES2≥.70	.70>RES2≥.60	RES2<.60
RDP	RDP≥.99	.99>RDP ≥.95	.95>RDP ≥.90	.90>RDP ≥.85	RDP <.85
AV	AV≥.40	.40>AV ≥.30	.30>AV ≥.20	.20>AV ≥.10	AV <.10
DI1	DI1≤ 15	15< DI1 ≤20	20< DI1 ≤25	25< DI1 ≤30	DI1 >35
DI2	DI2≤ 10	10< DI2 ≤13	13< DI2 ≤16	16< DI2 ≤19	DI2 >19
DI3	DI3≤ 7	7< DI3 ≤10	10< DI3 ≤13	13< DI3 ≤16	DI3 >16
DI4	DI4≤ 9	9< DI4 ≤12	12< DI4 ≤15	15< DI4 ≤18	DI4 >18
DI5	DA2≥.90	.90> DA2≥.80	.80> DA2≥.70	.70> DA2≥.60	DA2<.60
DI6	DI6≤ 3	3< DI6 ≤5	5< DI6 ≤7	7< DI6 ≤9	DI6 >9
DI7	DI7≥.85	.85> DI7≥.70	.70> DI7≥.55	.55> DI7≥.40	DI7<.40
SDE	SDE ≥.95	.95> SDE ≥.85	.85> SDE ≥.75	.75>SDE ≥.65	SDE <.65
CVI	CVI ≥.20	.20> CVI ≥.15	.15> CVI ≥.10	.10> CVI ≥.5	CVI <.5
CDM	CDM ≤ 15	15< CDM ≤25	25< CDM ≤35	35< CDM ≤45	CDM >45
NV	NV ≥.90	.90> NV ≥.80	.80> NV ≥.70	.70> NV ≥.60	NV <.60
CO1	CO1 ≥.95	.95> CO1 ≥.90	.90> CO1 ≥.85	.85>CO1 ≥.80	CO1 <.75
CO2	CO2 ≥.95	.95> CO2 ≥.90	.90> CO2 ≥.85	.85>CO2 ≥.80	CO2 <.75
CO3	CO3 ≥.95	.95> CO3 ≥.90	.90> CO3 ≥.85	.85>CO3 ≥.80	CO3 <.75
CE1	CE1 ≥.90	.90> CE1 ≥.80	.80> CE1 ≥.70	.70>CE1 ≥.60	CE1 <.60
CE2	CE2 ≥.90	.90> CE2 ≥.80	.80> CE2 ≥.70	.70>CE2 ≥.60	CE2 <.60
CE3	CE3 ≥.90	.90> CE3 ≥.80	.80> CE3 ≥.70	.70>CE3 ≥.60	CE3 <.60
CE4	CE4 ≥.90	.90> CE4 ≥.80	.80> CE4 ≥.70	.70>CE4 ≥.60	CE4 <.60
PDE	PDE ≥.90	.90> PDE ≥.00	.80> PDE ≥.70	.70>PDE ≥.60	PDE <.60

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS