UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Autogeneración de documentación web a partir de un modelo funcional en la herramienta *Recover*

Memoria

Trabajo Final de Grado

modalidad empresa en colaboración con

Carlos Massa Marcos

Tutor: Albert Tort Pugibet

Ponente: Ernest Teniente López (ESSI)

Grado en ingeniería informática

Especialización de Ingeniería del Software

Facultad de Informática de Barcelona

Resumen

Actualmente muchas empresas del sector tecnológico, se dedican a la producción de *software* para diversos clientes que requieren un servicio, cuya finalidad suele ser facilitar las tareas cuotidianas.

El problema viene dado cuando estos proyectos *software*, requieren una ampliación o mejora y no tienen una documentación actualizada, carecen de ella o es demasiado extensa para modificarla con facilidad, todo esto tiene como consecuencia una pérdida del conocimiento funcional del sistema.

El propósito de este proyecto, consiste en dar a la herramienta *Recover* la capacidad de generar una documentación web automática y actualizada de un sistema *software* a partir de la información proporcionada por la herramienta, reduciendo así los costes de mantenimiento derivados de cualquier cambio inherente al desarrollo *software*.

Resum

Actualment moltes empreses del sector tecnológic, es dediquen a la producció de software per diversos clients que requereixen un servei, la finalitat del qual sol ésser donar facilitat a tasques quotidianes.

El problema ve donat quan aquests projectes software, requereixen d'una ampliació o millora i no tenen una documentació actualitzada, careixen d'aquesta o es massa extensa per modificar-la amb facilitat, tot això te com a conseqüència una pérdua del coneixement funcional del sistema.

El proposit d'aquest projecte, consisteix en donar a l'eina Recover la capacitat de generar una documentació web automática i actualitzada d'un sistema software a partir de la informació proporcionada per l'eina, reduïnt així els costs de manteniment derivats de qualsevol canvi inherent al desenvolupament software.

Abstract

These days many companies of the technology sector, are engaged in software production for clients that require a service, the purpose of which is usually to ease daily tasks.

The problem arises when these software projects, require an extension or improvement and its documentation is outdated, non-existent or too long to edit with ease, all of this with the consequence of loss of functional knowledge of the system.

The purpose of this project, consists into give Recover tool the capacity of generating an automatic and up-to-date web documentation of a software system from the information provided by the tool, deceasing then the costs derived from maintenance of any inherent change coming from software development.

Agradecimientos

Quiero dar las gracias al ponente, Ernest Teniente, por las ideas aportadas para el proyecto y los consejos que hacen ver las cosas de otra manera, tanto profesionalmente cómo personalmente.

También quiero agradecer a mi director Albert Tort, por la oportunidad de trabajar en este proyecto, por todo lo que he podido aprender y por proporcionar la ayuda y el conocimiento necesario en todo momento.

A mi amigo y compañero Xavi, por dar una visión crítica durante el proceso de realización.

A Laura, por el apoyo moral y el impulso para terminar el trabajo.

Por último, pero no menos importante, gracias a mi familia que me ha apoyado en todo momento en este camino, especialmente a mis abuelos, sin su ánimo e insistencia esto no habría sido posible.

TABLA DE CONTENIDO

FIGURAS

TABLAS

1 INTRODUCCIÓN

El Trabajo Final de Grado "Autogeneración de documentación web a partir de un modelo funcional en la herramienta Recover", se enmarca en los estudios del Grado en Ingeniería Informática, con especialización en Ingeniería del Software impartido en la Facultad de Informática de Barcelona. Se trata de un trabajo realizado en cooperación con la empresa *Sogeti*[1], que ofrece servicios de *testing* a clientes del sector de las tecnologías de la información.

Concretamente, el proyecto tiene como objetivo la adición de un nuevo módulo a una herramienta comercial de Sogeti, con el fin de generar documentación web navegable a partir de la información proporcionada por dicha herramienta (modelos UML, casos de prueba, descripciones textuales, etc.). Este trabajo, por tanto, tiene como contexto la contribución directa a esta herramienta, desarrollada en el marco de investigación de la empresa.

1.1 FORMULACIÓN DEL PROBLEMA

En el área de los sistemas software, debido a la gran magnitud que pueden llegar a tener algunos proyectos realizados en equipos con grandes recursos humanos, la creación y mantenimiento de una documentación actualizada y representativa sobre el estado y el funcionamiento del sistema resulta imprescindible.

A simple vista, parece algo trivial, sin embargo, aun siendo una buena práctica, todavía podemos encontrar numerosos casos en los que una documentación funcional del sistema o bien no existe, o no se actualiza ante cambios. Esto desencadena en un futuro elevado coste por mantenimiento debido a la falta de documentación previa, dificultades en el diseño y ejecución de pruebas o modificaciones de un área no identificada en el sistema en la fase de desarrollo.

Pero si vamos más allá, este hecho afecta también en el área de los recursos humanos, ya que hace más difícil las nuevas posibles incorporaciones a un proyecto, que para solucionarlo requiere de una inversión de tiempo para explicar las particularidades del sistema y una atención constante a que las funcionalidades introducidas se realizan de manera correcta, en concordancia con el resto del sistema.

Por otro lado, en los casos en que existe documentación, muchas veces esta no tiene suficiente calidad y es poco mantenible, pues se trata de texto no estructurado y de gran extensión. Lo anterior implica dificultades para mantener la integridad del documento y frecuentemente desemboca en una documentación desactualizada. Todo esto, conlleva una pérdida del conocimiento actual (*know-how)* del sistema, lo que provoca dificultades inevitables para un proyecto software, debido a que aprender de nuevo el conocimiento funcional de un sistema tiene como consecuencia un coste tanto temporal como económico.

1.2 OBJETIVOS DEL TRABAJO

A raíz de los problemas que supone el no tener una documentación actualizada de un sistema *software* y las consecuencias que puede tener, nace este proyecto, que pretende dar solución a todos estos problemas a partir de una documentación que se autogenera en base al estado de un modelo funcional y que es ampliamente entendible por personas no tan especializadas en el sector. ¿Pero esto que quiere decir? ¿Cómo se puede llevar a cabo? ¿Qué tendrá esta documentación? ¿Qué problemas concretos soluciona y qué ventajas tiene?

Para empezar, se debe entender que un modelo funcional consiste en una representación formal de las funciones que puede realizar un sistema y qué transformaciones pueden sufrir los datos dentro de este. Este tipo de modelos se van construyendo poco a poco, mediante iteraciones donde se indican sus funciones hasta acabar de definirse por completo.

Una vez tenemos claro que es y cómo se forma un modelo funcional, hemos de profundizar en la manera en que se definen las funciones sobre este. Está claro que la primera opción y la más trivial que nos viene a la cabeza, es escribir todo el conocimiento funcional del sistema directamente, especificando que elementos interactúan entre si y como lo hacen. Pero esta no es la única manera, ¿Qué pasa si aprovechamos este conocimiento que tenemos sobre un sistema para escribir las pruebas que se pueden hacer sobre él?

Al escribir las pruebas primero (método conocido como *Test-Driven Development*), ya estamos especificando de manera implícita el comportamiento del sistema, por lo que extrayendo la información de las pruebas, se puede construir un modelo funcional completo.

Al relacionarlo con los objetivos, en este paso es donde entra en juego la forma con la que se puede llevar a cabo la generación automática de documentación; si tenemos las pruebas especificadas en un formato concreto y los resultados del modelo funcional generado a partir de estas, podemos automatizar un proceso que mediante toda esta información, la procese y cree una documentación formal sobre el estado de un sistema.

Esta documentación estará compuesta de los diversos artefactos mostrados a continuación:

- **Diagramas de clases UML (*Unified Modeling Language* o Lenguaje Unificado de Modelado):** Este tipo de diagrama se utiliza para describir la estructura interna de un sistema, como están relacionados sus componentes, los atributos y operaciones que contienen; cada uno de estos componentes recibe el nombre de clase.

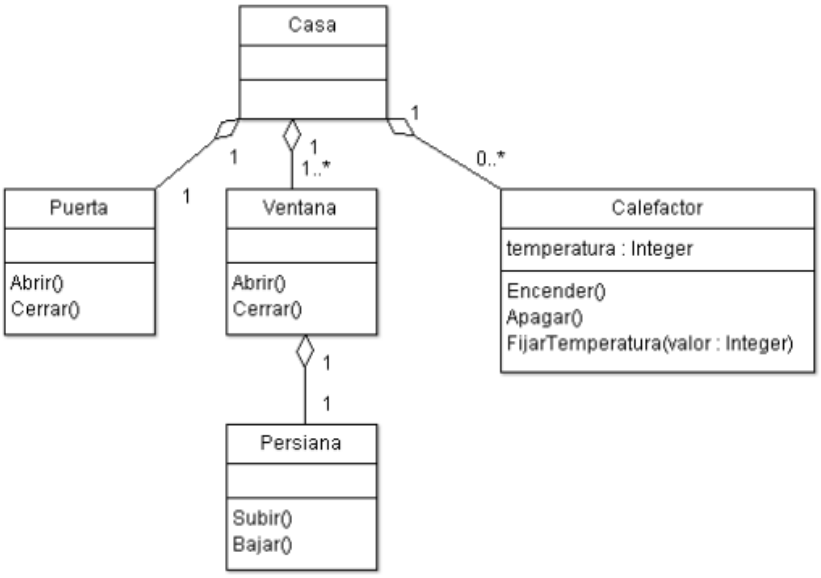


Figura 1.- Ejemplo de diagrama de clases

- **Diagramas de secuencia:** Define la interacción a través del tiempo que tendrán los componentes de un sistema entre ellos a la hora de realizarse la función concreta.

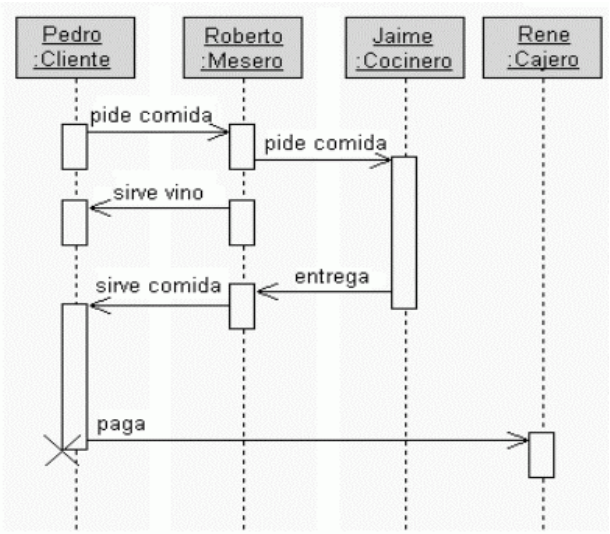


Figura 2.- Ejemplo de diagrama de secuencia

- **Diagramas de actividad:** Se utiliza para definir mediante una representación gráfica, el flujo de un proceso de un sistema.

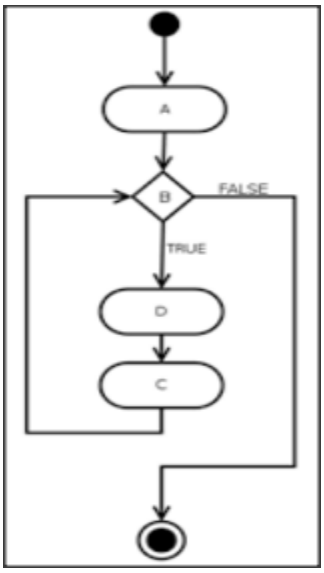


Figura 3.- Ejemplo de diagrama de actividad

- **Lenguaje *OCL (Object Constraint Language)*:** Es un lenguaje que permite definir expresiones formales de los modelos *UML*, mediante este lenguaje se pueden definir aspectos como el comportamiento de las funciones de una clase.

```
self.movimiento->select(m | m.oclIsTypeOf(Reintegro))
"Todos los reintegros de la cuenta"
```

Figura 4.- Ejemplo de código OCL

- Listado completo de las clases del sistema, donde cada clase tendrá la siguiente información:
  - Diagrama de clases *UML* de la clase concreta (elementos que interactúan con esta clase).
  - Descripción de los atributos.
  - Descripción y código *OCL* de las operaciones.
  - Traducción a lenguaje natural de las operaciones *OCL* y descripción a lenguaje natural del diagrama de clases *UML*.
- Listado completo de las operaciones de cada clase en el sistema.
- Listado de las enumeraciones existentes en el diagrama de clases del sistema, donde para cada una se muestra sus valores posibles.
- Un listado de los casos de uso del sistema, donde para cada uno se muestran las siguientes características:
  - Descripción del caso de uso
  - Diagrama de actividad generado automáticamente a partir de los casos de prueba de un caso de uso.
  - Diagramas de secuencia asociado a los escenarios del caso de uso.

En Mayo de 2002, el *National Institute of Standards and* Technology [2], estimó que los errores de *software* supusieron un coste de aproximadamente 60 billones de dólares, de los cuales una tercera parte se podría haber ahorrado [3]. Este dato, muestra como los errores de *software* están presentes a gran escala y hace notable la necesidad de tener esta documentación para poder desarrollar un sistema con el mínimo margen de error.

Llegado a este punto, las ventajas de realizar esta documentación son evidentes:

- Tener una documentación que se actualiza automáticamente, reduce los costes de mantenimiento.
- Se aborda directamente el problema de la gestión del conocimiento. El conocimiento funcional de un sistema está siempre actualizado en función a sus pruebas.
- El sistema está completamente especificado con exactitud.
- Se puede generar una documentación bien estructurada, en función a las necesidades, con tan solo cambiar el motor de generación, permite añadir partes nuevas con facilidad.
- El hecho de tener una traducción a lenguaje natural, facilita la comprensión de la documentación por parte de personas no especializadas, como por ejemplo directivos de empresas.

En definitiva, es un objetivo con muchos beneficios, que puede solucionar con creces las pérdidas económicas de muchas empresas de software en el mantenimiento de documentación sobre sus sistemas desarrollados.

1.3 INTRODUCCIÓN A *RECOVER*

Recover es una herramienta propietaria de la empresa *Sogeti*[1] que surge de la idea de la tesis doctoral *"Testing and test-driven development of conceptual schemas"*[4] de Albert Tort. Se basa en los trabajos previos *"An approach to test-driven development of conceptual schemas"*[5] y *"Testing conceptual schema satisfiability"*[6], en los que se propone una solución para su posterior uso en análisis y proyectos basados en modelos. Con el TFG presentado en este documento, se pretende añadir a Recover la generación de documentación navegable en HTML, extraíble de la información gestionada por Recover.

Para lograr este objetivo, *Recover* se basa en el método *Test-Driven Conceptual Modeling (TDCM),* en el cual se aplican las ideas esenciales de *Test-Driven Development (TDD)* junto con la idea del *Conceptual Schema-Centric Development (CSCD)*[7]*; d*e esta manera, mediante la especificación de los casos de prueba, que son tratados por el sistema a partir del lenguaje especializado *Conceptual Schema Testing Language (CSTL)*[8] (representado en la Figura 5), se consigue obtener y validar un esquema conceptual basado en las pruebas.

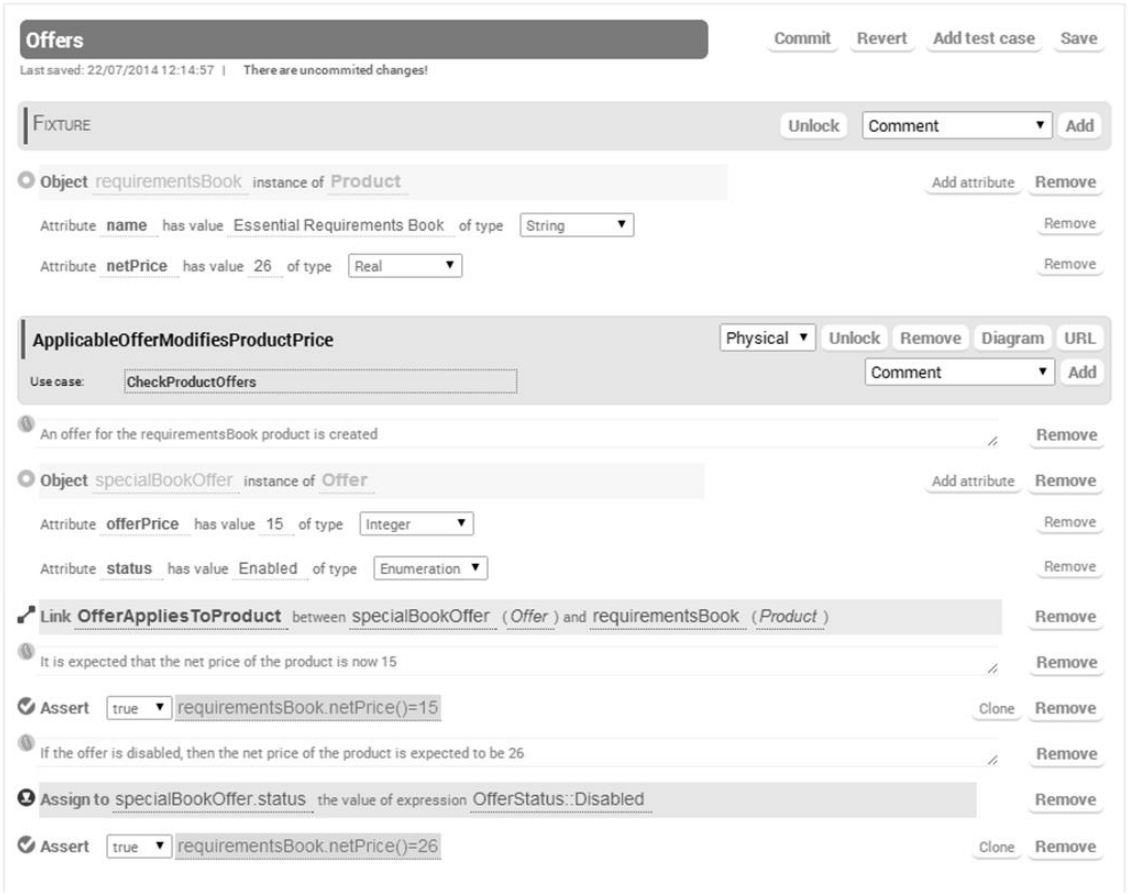


Figura 5.- Especificación de casos de prueba en Recover

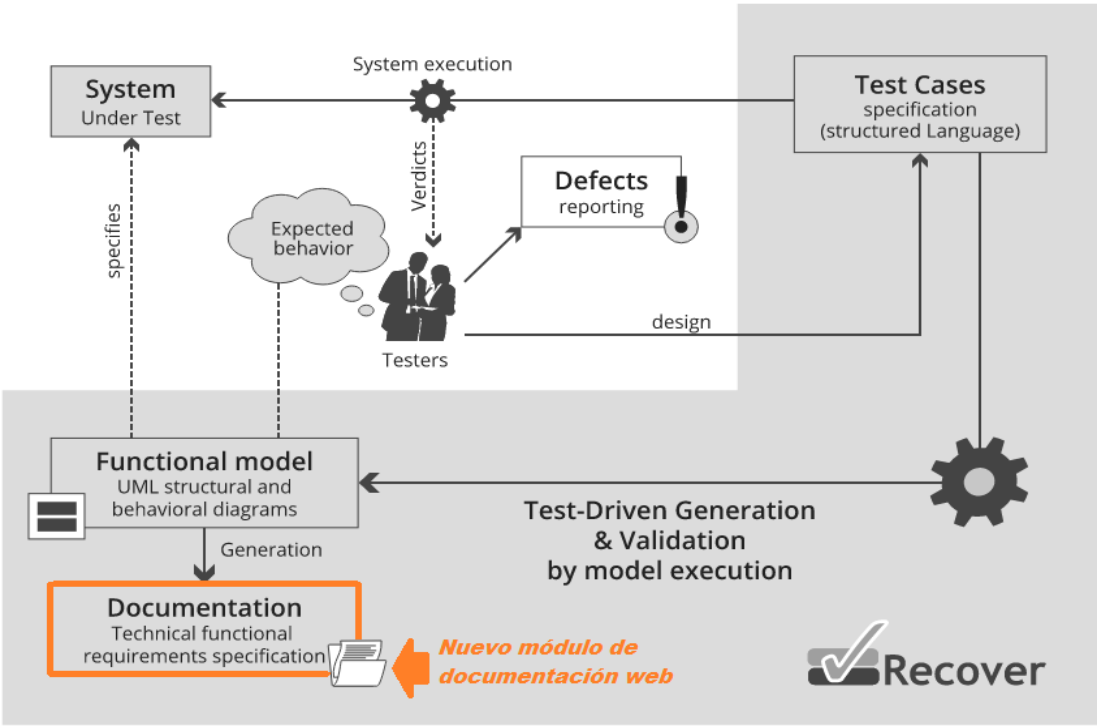


Figura 6.- Funcionamiento completo de Recover

Como se puede observar en la Figura 6, *Recover* intenta recuperar el conocimiento funcional de un sistema *software*, que está extraviado o directamente no existe, abordando así el aspecto de la gestión del conocimiento. Este conocimiento, que es adquirido incrementalmente durante la fase de *testing* de un proyecto, es introducido en forma de casos de prueba por parte de los *testers* y utilizado para la generación automática y *test-driven* de un modelo funcional en forma de diagramas *UML* estructurales y de comportamiento. De esta forma, se consigue una alineación entre las pruebas y el modelo funcional en base a sus requisitos. Además, disponer de un modelo permite utilizarlo para la validación de dicho alineamiento ante cambios. Si hay cambios en el modelo, se pueden detectar, por ejemplo, el impacto del cambio en los casos de prueba definidos.

Si se añaden nuevos casos de prueba o modifican los existentes, Recover permite actualizar el modelo de forma guiada, para mantener el conocimiento derivado de las pruebas actualizado y especificado.

Dado este escenario, *Recover* pretende reinventar el concepto de documentación, con una visión dónde cualquier cambio sobre el sistema, se ve reflejado rápidamente sin necesidad de recurrir manualmente a una modificación, obteniendo así una documentación *up-to-date* y siempre en concordancia con el diseño de las pruebas.

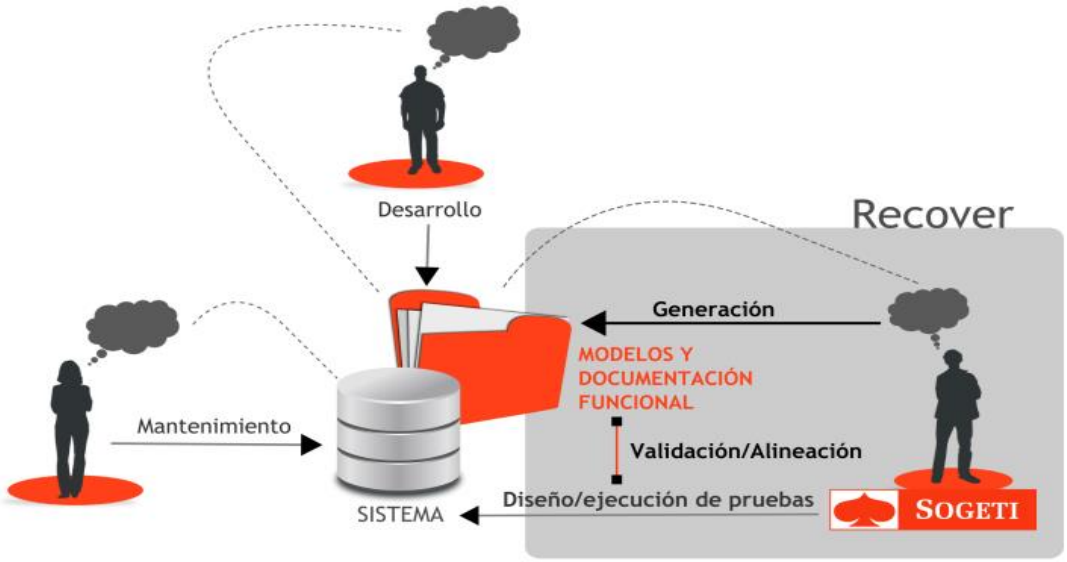


Figura 7.- Interacción de Recover frente a un sistema

Esto es posible gracias a que los ingenieros de pruebas, a lo largo de la vida del proyecto de software, adquieren incrementalmente conocimiento del sistema a partir de la fase de *testing*. Las pruebas se introducen en *Recover* y un motor se encarga de validar y alinear las pruebas con el modelo (ver Figura 7), y en caso de que *Recover* detecte la necesidad de regenerar/actualizar la documentación, se aplican acciones de evolución sobre el modelo (ver Figura 8).

A continuación se muestra el flujo a seguir y que se plantea utilizar para llevar a cabo el objetivo de generar una documentación web actualizada:

Figura 8.- Estrategia de alineamiento modelo/casos de prueba

Una vez generada la documentación, aparece otro problema a tener en cuenta, que tiene que ver con la tecnicidad de su contenido, pues al tratarse de la especificación de un sistema de información, se compone principalmente de *UML* y *OCL*, lo que supone un impedimento en cuanto a comprensión por parte de personas no técnicas, por tanto, es necesario un mecanismo para transformar estas partes de *UML* y *OCL* a lenguaje natural.

De esta manera, se añade la generación de una documentación estructurada del conocimiento especificado explícitamente en el modelo, con un formato navegable y más fácil de consultar que el modelo generado por Recover.

2 CONTEXTUALIZACIÓN

2.1 ESTADO DEL ARTE

En este apartado, se muestra de manera separada una revisión de la literatura existente de cada uno de los temas implicados en los objetivos del trabajo. Dado que el proyecto tiene como objetivo la generación de documentación web a partir de los modelos generados por Recover y que esta documentación sea también comprensible para diversas audiencias no expertas en la comprensión de *UML/OCL*, el estado del arte abarca tres ejes:

- Trabajo previo sobre generación automática de documentación web
- Trabajo previo sobre traducción de *UML* y *OCL* a lenguaje natural
- Trabajo previo sobre generación de diagramas de actividad y secuencia

2.1.1 Trabajo previo sobre generación automática de documentación web

Cuando hablamos de documentación web, se entiende principalmente como una documentación interactiva, donde el usuario puede navegar a partir de lo que se va mostrando, y en este aspecto, esto se corresponde de manera análoga a lo que conocemos como página web.

Pero si además, nos centramos en el aspecto de su generación automática basada en un sistema, destacan *Doxygen*[9] y *Javadoc*[10], herramientas que son capaces de generar documentación *up-to-date* de todo el código fuente de un programa de manera rápida y eficaz; la primera tiene soporte para los lenguajes más utilizados, mientras la segunda se centra el en lenguaje Java.

Desafortunadamente, de estas herramientas solo podemos aprovechar la idea, ya que al ser tan especializadas, adaptarlas supondría un esfuerzo mayor que crear nuestro propio método; además, el hecho de diseñar una solución propia, nos permitirá personalizarla, dándole un estilo y formato ajustado a nuestro propio criterio; en caso contrario, tan solo obtendríamos una plantilla rellena con datos, con poco toque diferenciador y muy limitada.

2.1.2 Trabajo previo sobre traducción de *UML* y *OCL* a lenguaje natural

El procesamiento de lenguaje natural, es un tema ampliamente trabajado en el ámbito de las ciencias de la computación y lingüística, donde a grandes rasgos, se intenta estudiar la manera de interactuar mediante el lenguaje humano con un computador. En este caso, el objetivo consiste en traducir los lenguajes *UML* y *OCL* a lenguaje humano.

En cuanto a tratamiento de *OCL,* se encuentran trabajos como *"A natural language processing approach to generate SBVR and OCL"*[11], centrado en la traducción de lenguaje natural a *OCL* y *"Translating Formal Software Specifications to Natural Language (A Grammar-Based Approach)"*[12], donde se traduce *OCL* a lenguaje natural y se consiguen resultados excelentes a partir del uso del *Grammatical Framework (GF)*[13].

En el caso de *Recover*, el sistema está especificado internamente mediante el lenguaje imperativo *SOIL (Simple OCL-based Imperative Language),* de las librerías del programa *USE (UML-based Specification Environment)*[14][15]; este lenguaje se utiliza tanto para el modelo, como para sus restricciones de integridad, esto implica que la librería contiene su propio analizador sintáctico y su jerarquía de clases, por este motivo, deberá diseñarse una nueva solución compatible con la estructura interna de la librería, para traducir el esquema conceptual y sus operaciones.

2.1.3 Trabajo previo sobre generación automática de diagramas de actividad y secuencia

Actualmente existen numerosos programas capaces de permitir al usuario la creación de diagramas de todo tipo, incluidos de actividad y de secuencia. Concretamente podemos encontrar programas como Microsoft Visio[16], Visual Paradigm[17] o Enterprise Architect[18], siendo este último uno de los más completos.

Sin embargo, aun siendo las mejores alternativas encontradas en cuanto a calidad, todos estos requieren trabajo manual para generarse, porque ninguno de ellos pone a disposición librerías independientes del propio *software,* para poder realizar la integración a una herramienta web.

Teniendo en cuenta la limitación anterior, las alternativas más populares son las librerías *jsUML2*[19] y *JUMLY*[20], ambas están basadas en JavaScript, lo que permite una fácil integración y la posibilidad de idear un mecanismo automático de generación de diagramas de actividad y secuencia a partir de un metamodelo, basado en el lenguaje de *Recover.*

	Diagrama de actividad	**Diagrama de secuencia**
jsUML2	✓	✓
JUMLY	✗	✓

Figura 9.- Comparativa entre jsUML2 y JUMLY

Finalmente al analizar las dos posibilidades, dado que el objetivo es generar los dos tipos de diagrama mencionados y la librería *JUMLY* no nos lo permite (véase Figura 9), la opción más apropiada es utilizar *jsUML2*, así se podrá aprovechar esta y adaptarla para la consecución del objetivo previsto.

2.1.4 Conclusión

Después de analizar el estado del arte, para conseguir añadir la nueva funcionalidad de documentación web, deberemos utilizar las librerías *USE* y *jsUML2* como apoyo, mediante la modificación de estas librerías y la creación de metamodelos sobre estas, se puede conseguir el objetivo de este proyecto.

A continuación, se muestra el diagrama de interactuación entre los diferentes componentes, las librerías y que saldrá de cada una de ellas.

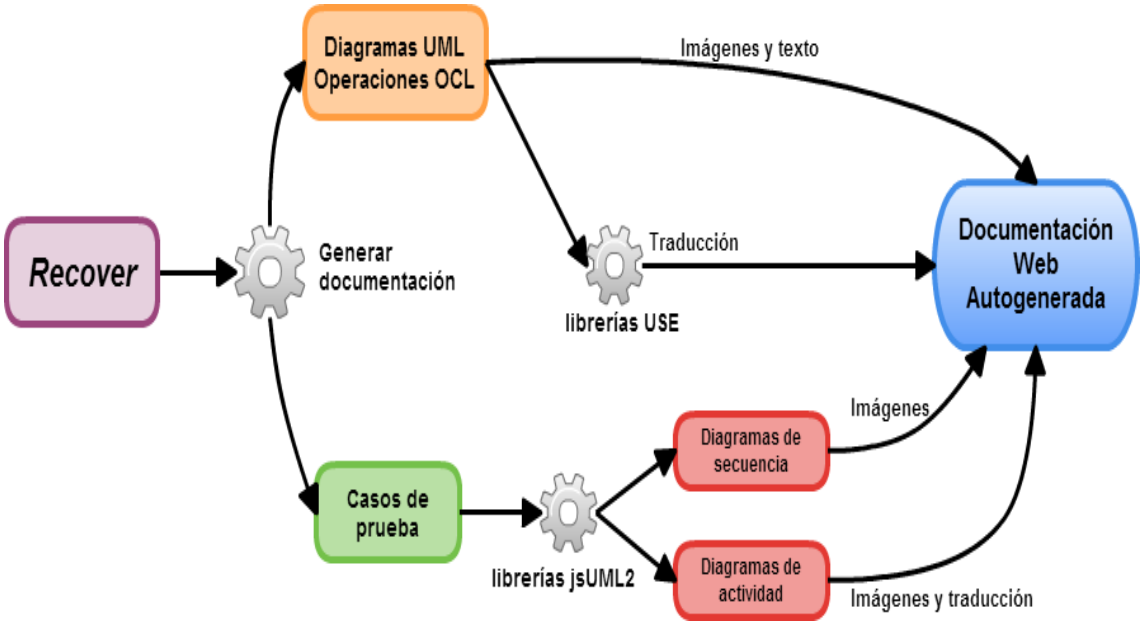


Figura 10.- Funcionamiento de la funcionalidad de documentación web

En el esquema de la Figura 10, se observa que para conseguir la documentación web se originará un proceso principal, que se encargará de crear una documentación base con la información extraída de *Recover* (clases, atributos y operaciones de clases, enumeraciones y casos de uso) y para completarla, se dividirá en dos fases más complejas.

En la primera fase, las imágenes de diagramas de clases de los diagramas *UML* y texto del código *OCL* de las operaciones, se añadirán directamente a la documentación y seguidamente a partir de la librería *USE,* se hará una traducción a lenguaje natural del diagrama *UML* y de las operaciones *OCL* que también será añadido a la documentación.

En cuanto a la segunda fase, corresponde a la generación de diagramas de secuencia y de diagramas de actividad, mediante los casos de prueba asociados a un caso de uso y a partir de la librería *jsUML2*. Los artefactos resultantes que se añadirán a la documentación, consisten en las imágenes de los diagramas y la explicación textual de los diagramas de actividad.

2.2 PARTES INTERESADAS

En este apartado se definirán los principales actores o partes interesadas (*stakeholders*) del proyecto, es decir, las personas que tienen relación alguna con el proyecto y obtienen un beneficio de este.

Desarrollador

El desarrollador del proyecto, es uno de los actores más implicados directamente en el mismo, ya que se encarga de su realización, recayendo sobre él los resultados de su evolución. En este caso, debido a que la naturaleza del proyecto es un "Trabajo de Final de Grado", el desarrollador es un único estudiante el cual se beneficia de la evaluación.

Director y ponente del proyecto

Tanto el director y el ponente del TFG tienen interés en la correcta realización del proyecto, ofrecen soporte y supervisión al proceso del mismo. El director del TFG es Albert Tort, consultor sénior de Software Control & Testing en Sogeti y el ponente es el profesor Ernest Teniente del departamento ESSI (Enginyeria de Serveis i Sistemes d'Informació) de la Universidad Politécnica de Cataluña.

Jefe del proyecto *Recover*

El jefe de proyecto es uno de los actores más importantes del proyecto y forma parte de la empresa *Sogeti*. En este caso el jefe de proyecto es Albert Tort, que será el encargado de guiar y supervisar la dirección que toma el proyecto en cada una de las fases, así como ayudar a establecer los requisitos necesarios. Su interés radica en obtener un producto mucho más completo a ofrecer a los clientes.

Usuarios

Los usuarios de la herramienta *Recover*, son trabajadores de la empresa encargados de utilizarla para especificar el conocimiento del sistema de un cliente en paralelo con el servicio de pruebas realizado, por tanto, estos se beneficiarán gracias a la obtención de más información por parte del sistema, a medida que introducen datos sobre la herramienta.

Empresa

Se refiere a la empresa *Sogeti*, que provee de los servicios de *Recover,* y pretende obtener un beneficio económico, gracias a las mejoras en el proyecto.

Clientes

Los clientes que solicitan los servicios de *testing* con *Recover*, son de carácter diverso y tanto especializados como no en el campo de la informática, gracias a la solución *Recover* con el módulo desarrollado en este proyecto, se beneficiarán de la obtención de una documentación estructurada de su sistema, que puede ser generada cuando lo

deseen, junto con una traducción automatizada a lenguaje natural, que permite la comprensión a un público más amplio y que no requiere de gran especialización. De hecho, en los clientes hay distintos perfiles que requieren saber del conocimiento del sistema para su trabajo (des de analistas funcionales, hasta desarrolladores y profesionales de mantenimiento).

Comercial de *Recover*

Es el encargado de ofertar el producto a los clientes potenciales del sector; al mostrar las nuevas características y eficacia de *Recover,* se incrementará su demanda y con ello también el productor obtendrá mayores beneficios.

2.3 ALCANCE

Para enmarcar este trabajo como TFG se ha limitado el alcance a la integración de tres funcionalidades claramente identificadas:

- Generación de una documentación web navegable y básica formada a partir de diagramas *UML* y especificaciones *OCL* de cada una de sus clases.
- Traducción a lenguaje natural de las partes técnicas de la documentación (*UML* y *OCL*).
- Implementación de un mecanismo automático de generación de diagramas de secuencia y de actividad. Estos diagramas se generan a partir de casos de test y se incluirán en la documentación.

2.3.1 Objetivos

El objetivo final de este proyecto, es la creación de un mecanismo de generación automática de documentación web sobre la herramienta *Recover.* Este mecanismo pretende complementar el proceso de especificación de un sistema software, es decir, dar información al usuario o cliente sobre cómo está hecho el sistema. A partir de la especificación de sus casos de prueba y el esquema conceptual alineado sobre *Recover,* se consigue generar una documentación navegable y accesible a un público más amplio.

2.3.2 Posibles obstáculos

El aspecto más importante a tener en cuenta en un proyecto, es el referente a la **gestión del tiempo**, en este caso, se dispone de un cuatrimestre para su completitud, lo que supone un tiempo bastante limitado, por ese motivo, hará falta elaborar una buena planificación, completa, realista y lo más detallada posible, pero dejando margen de flexibilidad temporal para posibles imprevistos.

Otro problema posible a tener en cuenta, es el **mal diseño de la arquitectura** del mecanismo a integrar, ya que esto nos impediría alcanzar el objetivo principal del proyecto. Para evitar cualquier problema relacionado con este aspecto, se realizarán reuniones con el responsable del proyecto para garantizar que se está tomando la dirección correcta.

La fase de generación de diagramas para la documentación, deberá utilizar herramientas ya existentes, como por ejemplo **librerías de terceros** encargadas de proporcionar un método de representación gráfica, por tanto, habrá que tener en cuenta aspectos como la dificultad de aprendizaje, la fiabilidad y la compatibilidad.

Por último, el proyecto debe tener **una integración progresiva** en una herramienta que está en desarrollo y uso por más integrantes del equipo. Un fallo en la implementación, podría comprometer la estabilidad del sistema y por ello hará falta elaborar un buen plan de pruebas.

2.4 METODOLOGÍA Y RIGOR

2.4.1 Métodos de trabajo

El desarrollo de este proyecto, se ha llevado a cabo mediante la metodología ágil **SCRUM**, debido a que es el tipo de estrategia seguido en el equipo de *Recover*. Esta permite tener un control regular mediante *feedback* del responsable de proyecto, una adecuada gestión de las expectativas y una aplicación de soluciones más eficiente frente a posibles riesgos.

Se incluirán **tres** iteraciones de una duración determinada (4 semanas), obteniendo un producto fiable y funcional al finalizar cada iteración, y por tanto entregables con nuevas características.

Figura 11.- Funcionamiento de la metodología SCRUM

La **primera iteración,** se centrará en obtener un mecanismo de **generación de documentación web básica e integración** sobre la herramienta. El trabajo de esta iteración no es trivial y resulta primordial, ya que las siguientes iteraciones resultan ser funcionalidades incrementales a partir de esta.

Para la **segunda iteración,** se añadirá un módulo capaz de realizar una **traducción a lenguaje natural** de todas las partes técnicas de la documentación (diagramas *UML* y código *OCL*).

Finalmente en la **tercera iteración**, se completará el proyecto incorporando una **generación automática de diagramas de actividad y de secuencia,** complementarios a los casos de uso incluidos en la documentación.

2.4.2 Herramientas de seguimiento

Debido a que el trabajo a desarrollar, forma parte de un proyecto comercial que está en constante desarrollo por parte de un equipo, se utilizará un repositorio en línea alojado en GitHub para el control de versiones y Git como herramienta de gestión del repositorio; esto nos garantizará una disponibilidad del código en cualquier plataforma y facilidad de recuperar código anterior en caso de errores.

El seguimiento del trabajo, se verá reflejado a su vez en el sistema de gestión de tickets (Jira[21]), donde a medida que se vayan completando tareas se irán marcando como 'Completadas', lo que aportará precisión del estado del proyecto.

En cuanto a la comunicación con el director del proyecto, se utilizará el correo electrónico como medio de intercambio de mensajes y sistemas de compartición de ficheros online (Google Drive o Dropbox).

2.4.3 Métodos de validación

Para validar los avances que se van produciendo a lo largo de la realización del proyecto, se utilizarán los siguientes métodos:

- Se establecerán reuniones frecuentes a lo largo de cada iteración con el responsable del proyecto, así como reuniones con el tutor del proyecto al final de cada una para recibir el *feedback* necesario.

- Con el objetivo de comprobar que las funcionalidades desarrolladas responden positivamente, se incluirá una fase de *testing*, la cual nos permitirá saber si una funcionalidad está operativa antes de pasar a la siguiente.

- Para validar que la planificación establecida para el proyecto se ajusta a una línea temporal real, se tomará nota tanto del estado de las tareas como su duración y partir de este método, podremos ver qué fases son más susceptibles a cambios temporales, respecto a la planificación inicial y los posibles motivos.

2.4.4 Entorno de desarrollo

En el proyecto *Recover* se trabaja de manera colaborativa, es por ello que se precisan de herramientas que permitan tanto trabajar a manera individual, para realizar las pruebas pertinentes de las nuevas funcionalidades que pueda añadir un desarrollador, como compartida, para agrupar un conjunto de funcionalidades de interactuación.

En este proyecto, las herramientas principales utilizadas son las siguientes:

- **Eclipse:** Es una *IDE (Integrated Development Environment)* que permite desarrollar código en cualquier tecnología, principalmente Java o relacionados.
- **JBoss:** Servidor que posibilita ejecutar instancias de aplicaciones Java. Permite la integración con Eclipse y la ejecución de este en el ordenador personal. En *Recover* también se utiliza para la versión de producción.
- **Git:** Programa que permite la gestión de repositorios de código. En *Recover,* se utiliza para mezclar las diferentes funcionalidades implementadas por diferentes desarrolladores.
- **Gerrit:** Permite realizar una revisión de los cambios realizados en Git y aceptarlos o denegarlos.
- **Jenkins:** Aplicación web que permite compilar versiones, generar *builds* y ejecutar pruebas automáticas sobre esta.

A continuación, se muestra el flujo normal de trabajo en *Recover,* combinando las herramientas previamente mencionadas:

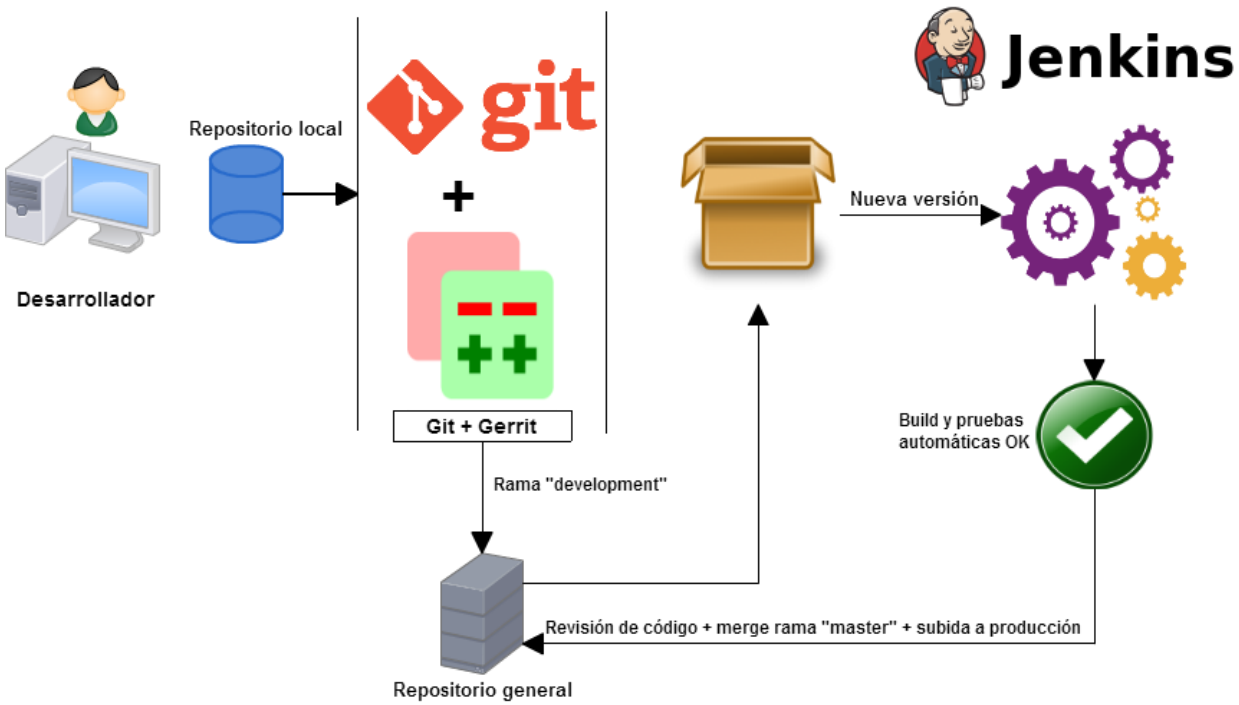


Figura 12.- Flujo de trabajo y entorno de desarrollo en Recover

3 ANÁLISIS DE REQUISITOS

En este apartado, se especifican los requisitos que deberá cumplir el proyecto que se realiza, cuyo objetivo en el ámbito de la ingeniería del software consiste en obtener una descripción detallada de qué características cumplirá el producto final desarrollado. Estos requisitos se clasifican en "requisitos funcionales" y "requisitos no funcionales".

3.1 REQUISITOS FUNCIONALES

Los requisitos funcionales, son los que describen las funcionalidades que ofrecerá el sistema y cómo se comportará ante un conjunto de entradas y cada uno estará descrito por los siguientes campos:

- **Descripción:** describe en que consiste el requisito.
- **Justificación:** indica el motivo por el cual se añade este requisito.
- **Prioridad:** muestra el grado de prioridad del requisito, valorado como Bajo, Medio o Alto.
- **Satisfacción del cliente:** indica el grado de satisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.
- **Insatisfacción del cliente:** indica el grado de insatisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.

Requisito #1	**Generar documentación**				
Descripción	Los usuarios, podrán generar documentación que contenga todos los artefactos sobre el sistema *software* introducido en *Recover.*				
Justificación	Se requiere disponer de una documentación completa a entregar a los clientes, por tanto, la documentación generada deberá contener todos los artefactos disponibles en la herramienta *Recover* generados a partir del sistema *software.*				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	5

Requisito #2	**Visualizar documentación**				
Descripción	Los usuarios, podrán ver la documentación en formato web en un navegador una vez generada.				
Justificación	Los usuarios, pueden tener la necesidad de acceder a la documentación web, sin necesidad de descargarla.				
Prioridad	Alta	**Satisfacción del cliente**	4	**Insatisfacción del cliente**	5

Requisito #3	**Descargar documentación**				
Descripción	Los usuarios, podrán descargar la documentación generada en formato 'zip'.				
Justificación	Los usuarios, pueden querer descargar la documentación para conservarla, transferirla o mostrarla en otros dispositivos sin acceso a la herramienta.				
Prioridad	Media	**Satisfacción del cliente**	2	**Insatisfacción del cliente**	3

Requisito #4	**Cambiar idioma**				
Descripción	El idioma de la documentación, podrá cambiarse previamente a realizar una generación, obteniendo así una documentación en el idioma seleccionado.				
Justificación	Los clientes que demanden los servicios de la herramienta *Recover,* pueden ser tanto nacionales como internacionales, por tanto, se debe tener la posibilidad de generar una versión en español e inglés.				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	3

3.2 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales, son los que describen las restricciones o exigencias de cualidades que el sistema debe cumplir y cada uno estará descrito por los siguientes campos:

- **Descripción:** describe en que consiste el requisito.
- **Justificación:** indica el motivo por el cual se añade este requisito.
- **Criterio de satisfacción:** indica cómo se puede comprobar que el requisito se ha alcanzado con satisfacción.
- **Prioridad:** muestra el grado de prioridad del requisito valorado como Bajo, Medio o Alto.
- **Satisfacción del cliente:** indica el grado de satisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.
- **Insatisfacción del cliente:** indica el grado de insatisfacción que provoca sobre el cliente el cumplimiento del requisito, valorado con una escala ascendente entre 1 y 5.

3.2.1 Requisitos de percepción

Requisito #1					
Descripción	La documentación, debe utilizar los colores e identificativos de la empresa				
Justificación	El aspecto visual de la documentación, no debe discrepar de la imagen corporativa que tienen los clientes				
Criterio de satisfacción	El responsable del proyecto, validará si el diseño es correcto de acuerdo con la imagen corporativa				
Prioridad	Alta	**Satisfacción del cliente**	4	**Insatisfacción del cliente**	4

3.2.2 Requisitos de usabilidad y humanidad

Requisito #2	
Descripción	La interfaz de generación de documentación, es fácil de entender y contiene las mínimas opciones para simplificar su uso.
Justificación	El mecanismo será utilizado con frecuencia, por lo que tiene que ser sencillo de utilizar en cualquier momento.
Criterio de satisfacción	El responsable de proyecto y los usuarios confirman la simplicidad de uso de la interfaz.

Prioridad	Media	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	5

Requisito #3					
Descripción	Las funcionalidades relacionadas con la documentación, se deben poder realizar sin uso previo.				
Justificación	Los usuarios, deben poder generar una documentación, de manera sencilla sin uso previo.				
Criterio de satisfacción	El responsable del proyecto y los usuarios encargados de la utilización de la herramienta, confirman su simplicidad utilizándolo sin uso previo.				
Prioridad	Media	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	5

Requisito #4					
Descripción	El sistema, debe mostrar al usuario, el estado del proceso de generación de documentación.				
Justificación	Los usuarios, deben saber en todo momento, si la documentación se ha generado con éxito o no.				
Criterio de satisfacción	El sistema muestra el estado de la generación, al procesar las operaciones necesarias.				
Prioridad	Alta	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	3

Requisito #5					
Descripción	La documentación y sus funcionalidades, deben tener un registro lingüístico formal y correcto.				
Justificación	Los usuarios, tienen que poder entender en cada momento, el contenido de la documentación.				
Criterio de satisfacción	El responsable del proyecto dará el visto bueno respecto al lenguaje utilizado.				
Prioridad	Media	**Satisfacción del cliente**	1	**Insatisfacción del cliente**	5

Requisito #6					
Descripción	El mecanismo de generación, funcionará completamente en cualquier navegador moderno.				
Justificación	Al ser una aplicación que se ejecuta sobre un navegador web, es importante que funcione en cualquier navegador utilizado actualmente.				
Criterio de satisfacción	La ejecución del mecanismo, será satisfactoria para navegadores como Google Chrome, Mozilla Firefox e Internet Explorer.				
Prioridad	Media	**Satisfacción del cliente**	3	**Insatisfacción del cliente**	3

3.2.3 Requisitos de rendimiento y disponibilidad

Requisito #7					
Descripción	El sistema debe generar la documentación, de la manera más rápida posible.				
Justificación	La generación de documentación se realizará habitualmente, por lo que deberá ser un proceso rápido.				
Criterio de satisfacción	El sistema deberá generar la documentación en menos de un minuto.				
Prioridad	Alta	**Satisfacción del cliente**	2	**Insatisfacción del cliente**	5

Requisito #8					
Descripción	Un fallo en el sistema, no debe comprometer significantemente el servicio proporcionado al usuario.				
Justificación	Una generación de documentación incorrecta, no debe provocar un fallo que afecte al funcionamiento normal de la aplicación.				
Criterio de satisfacción	Se realizarán pruebas de todos los posibles escenarios, garantizando que en ningún caso fallará la generación.				
Prioridad	Alta	**Satisfacción del cliente**	5	**Insatisfacción del cliente**	5

3.2.4 Requisitos legales y de seguridad

Requisito #9					
Descripción	La información utilizada de las empresas, cumplirá con la LOPD (Ley Orgánica de Protección de Datos).				
Justificación	Se debe garantizar que la información utilizada, cumple la legislación actual.				
Criterio de satisfacción	Un asesor de la empresa, garantizará que se cumple con la LOPD.				
Prioridad	Alta	**Satisfacción del cliente**	1	**Insatisfacción del cliente**	5

Requisito #10					
Descripción	Los usuarios, tendrán acceso sólo a las funcionalidades para las que tengan permiso.				
Justificación	Las acciones que realiza el usuario, deben estar controladas por permisos, para evitar usos incorrectos de funcionalidades				
Criterio de satisfacción	Se realizarán pruebas mediante usuarios con distintos permisos, para comprobar que funciona correctamente el sistema de permisos.				
Prioridad	Alta	**Satisfacción del cliente**	1	**Insatisfacción del cliente**	2

3.2.5 Requisitos de escalabilidad

Requisito #11					
Descripción	El mecanismo de generación, debe ser fácilmente ampliable por otros desarrolladores, para añadir nuevas características o realizar modificaciones a las existentes.				
Justificación	En el futuro, ser necesario añadir nuevas características o modificar lasexistentes.				
Criterio de satisfacción	El sistema está diseñado con una planificación y documentación previa, que facilita la ampliación o modificación en un futuro.				
Prioridad	Alta	**Satisfacción del cliente**	4	**Insatisfacción del cliente**	4

4 ESPECIFICACIÓN

A partir del análisis de requisitos realizado en el apartado anterior, se debe elaborar su especificación; mediante esta sección se pretende describir el comportamiento que tendrá el sistema, desde el punto de vista de los actores que lo utilizarán. A lo largo de este capítulo se mostrarán los actores que interactúan con el sistema, diagramas de casos de uso, descripción de los casos de uso, modelo conceptual y modelo de comportamiento.

4.1 ACTORES

A continuación se muestran los actores que interactuarán con el sistema:

Usuario

Es el encargado de utilizar la herramienta *Recover.* Este actor introduce datos sobre los casos de prueba del sistema software a analizar y realiza ejecuciones para validar la alineación de los casos de prueba con el modelo. Utilizará las nuevas funcionalidades a añadir, para generar una documentación una vez el alineamiento es satisfactorio.

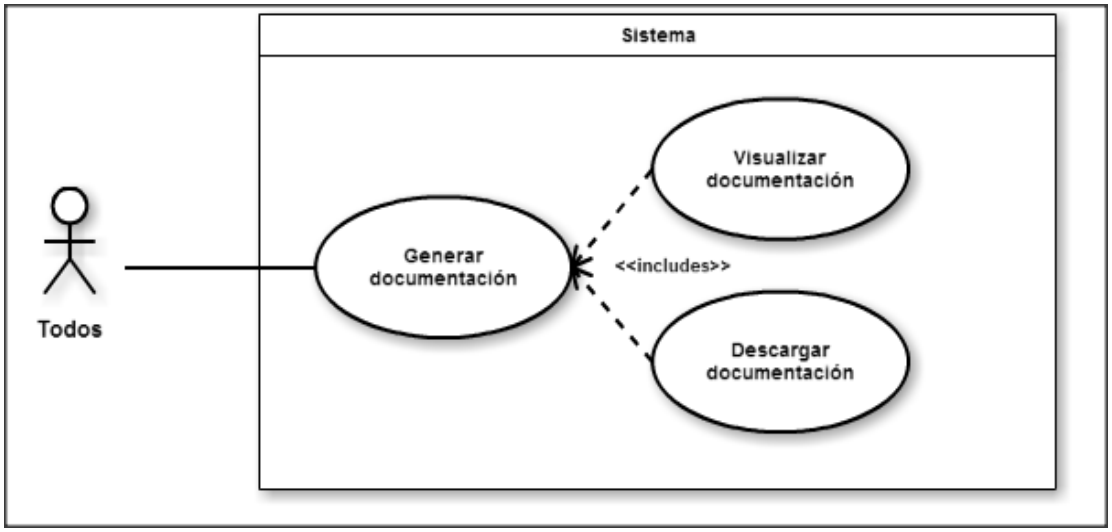
Administrador

La diferencia de este usuario con el usuario convencional, es que tiene más permisos sobre la herramienta y que a efectos prácticos de este proyecto, realizará las mismas acciones que cualquier usuario, pero representa el rol del jefe de proyecto.

Se utilizará "Todos" como nombre de actor, para referirse a los actores descritos previamente como un conjunto único sin diferencias.

4.2 DIAGRAMA CASOS DE USO

El diagrama de casos de uso, muestra visualmente la relación actor-sistema en términos de qué operaciones o procesos, puede realizar un actor mediante la interacción con el sistema.



4.3 DESCRIPCIÓN CASOS DE USO

En este apartado, se realiza la descripción de los casos de uso mostrados de manera visual en el apartado anterior. Esta descripción consiste en la enumeración detallada de eventos, que debe realizar un actor con el sistema y la respuesta de este para alcanzar el objetivo del caso de uso. Estos estarán descritos mediante los siguientes campos:

- **Actor principal:** actor implicado en el caso de uso.
- **Precondición:** condición a satisfacer previamente a la ejecución del caso de uso
- **Disparador:** evento que implica la ejecución del caso de uso.
- **Escenario principal:** secuencia de interacciones a realizar entre el usuario y el sistema con el fin de satisfacer el caso de uso.
- **Extensiones:** secuencias alternativas del escenario principal, debido a algún comportamiento diferente al especificado en el escenario principal.

Caso de uso 1: Generar documentación

Actor principal: Todos.

Precondición: El usuario ha realizado una validación satisfactoria del alineamiento entre pruebas y modelo.

Disparador: El usuario desea generar una documentación en formato HTML sobre el sistema *software* en *Recover.*

Escenario principal:

1. El usuario se dirige a la pantalla de documentación mediante la opción "Documentación".
2. El usuario presiona el botón "Generar especificación HTML".
3. El sistema procesa la solicitud del usuario, generando una documentación HTML en formato *'zip'* y mostrando un diálogo donde descargarla.

Extensiones:

3.a. El sistema sufre algún fallo inesperado durante la generación de la documentación.

3.a.1. El sistema muestra un diálogo al usuario indicando un error en la generación.

3.a.2. Acaba el caso de uso.

Caso de uso 2: Visualizar documentación

Actor principal: Todos.

Precondición: El usuario ha generado una documentación satisfactoriamente (Caso de uso 1)

Disparador: El usuario desea ver la documentación HTML en su navegador.

Escenario principal:

1. El usuario se dirige a la pantalla de documentación mediante la opción "Documentación".
2. El usuario presiona el link "Enlace a la documentación online".
3. Se muestra en el navegador del usuario, el índice de la documentación navegable.

Extensiones: -

Caso de uso 3: Descargar documentación

Actor principal: Todos.

Precondición: El usuario ha generado una documentación satisfactoriamente (Caso de uso 1).

Disparador: El usuario desea descargar la documentación web a su ordenador personal.

Escenario principal:

1. El usuario presiona el link de descarga de la documentación.
2. El sistema procesa la petición y genera el archivo en formato *'zip'* con el contenido de la documentación.

Extensiones: -

4.4 MODELO CONCEPTUAL

El modelo conceptual, pretende representar los conceptos que influyen y son significativos en el dominio del problema a resolver. A través de este, se muestran los conceptos implicados, atributos, relaciones y las restricciones de integridad aplicadas sobre los conceptos.

4.4.1 Esquema conceptual

El esquema conceptual, es la representación gráfica a alto nivel de los conceptos que participan en el sistema, relaciones, atributos y multiplicidades, que en este caso se ha dividido en partes para facilitar su comprensión.

En primer lugar, en la Figura 13, tenemos los conceptos relativos a la lógica de los casos de prueba y casos de uso en *Recover* (*UseCase, TestCase* y *TestProgram)* y los elementos que los componen *(TestElement* y *StateElement).* La información de estos casos de prueba, casos de uso y modelo, es la utilizada para la representación de los diagramas de diferente tipo; todo lo mencionado, se engloba en la documentación (*Documentation).*

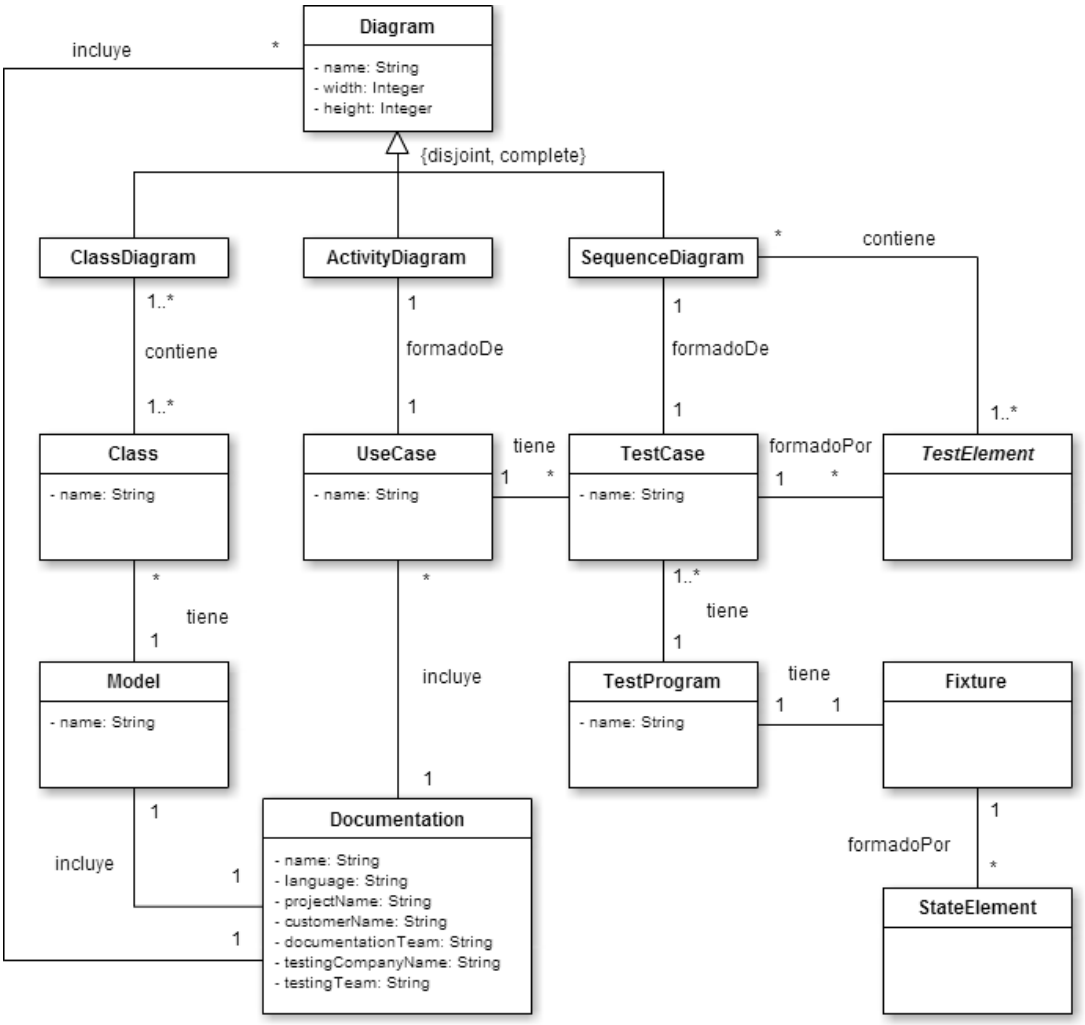


Figura 13.- Esquema conceptual del sistema

La Figura 14, muestra a partir de la clase '*Class*' todos los conceptos relacionados con ella y que intervienen en la representación de un diagrama *UML* sobre la clase *'Model'* (ver Figura 13).

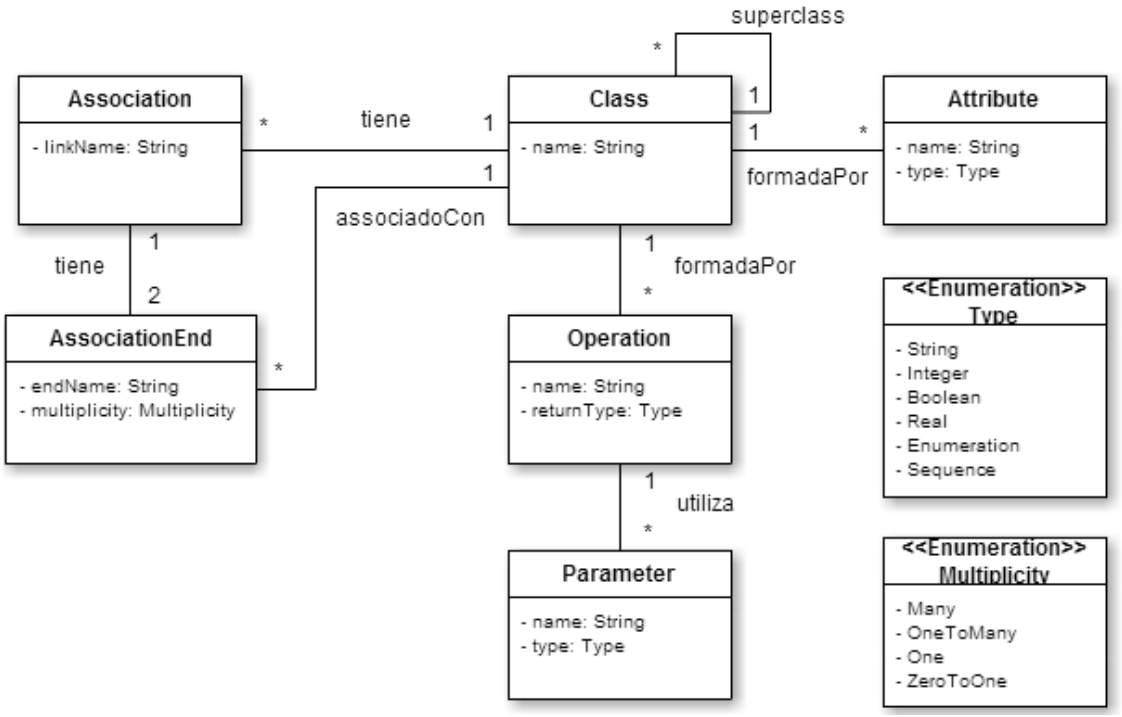


Figura 14.- Esquema conceptual de la clase 'Class'

Finalmente, la jerarquía de la Figura 15, indica los diferentes tipos que pueden adoptar un '*TestElement*' y un '*StateElement*' en relación a su *'TestCase'* y *'TestProgram'* respectivamente.

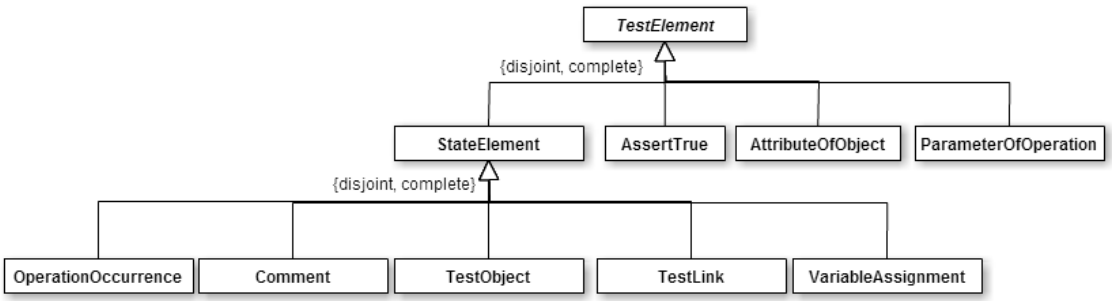


Figura 15.- Jerarquía de los elemetos de los casos de prueba

4.4.2 Restricciones de integridad

Las restricciones de integridad, son aquellas que no pueden ser representadas gráficamente y que son necesarias para garantizar la integridad del esquema.

- El *name* de un *Diagram* tiene que ser único.
- El *name* de una *Class* tiene que ser único.
- El *name* de un *UseCase* tiene que ser único.
- El *name* de un *TestCase* tiene que ser único.
- El *name* de un *TestProgram* tiene que ser único.
- Una *Class* no puede ser superclase de sí misma.

4.4.3 Descripción

Diagram

Representa el tipo genérico de diagrama. Todo tipo concreto de diagrama hereda de esta clase sus características.

- **name:** nombre del diagrama.
- **width:** valor entero que indica la anchura del diagrama.
- **height:** valor entero que indica la altura del diagrama.

ClassDiagram

Representa un diagrama de clases, una instancia concreta de diagrama; cada diagrama de clases, está asociado a un conjunto propio clases que lo forman.

ActivityDiagram

Representa un diagrama de actividad, una instancia concreta de diagrama; un diagrama de actividad, está formado a partir de los casos de prueba de un caso de uso asociado.

SequenceDiagram

Representa un diagrama de secuencia, una instancia concreta de diagrama; este tipo de diagrama, se compone de la información de los elementos de test de un caso de prueba concreto.

Class

Representa una clase de un modelo conceptual.

- **name:** nombre de la clase.

Model

Representa un modelo conceptual.

- **name:** nombre que utiliza el sistema, para diferenciar entre los diferentes modelos existentes (sin espacios).

Documentation

Representa la documentación del sistema *Recover*; esta documentación contiene información de todos los elementos en el sistema a partir de un modelo, casos de uso y diagramas de un proyecto.

- **name:** nombre de la documentación (formado añadiendo del nombre del proyecto).
- **languaje:** lenguaje en que se muestra la documentación.
- **projectName:** nombre del proyecto asociado a la documentación.
- **customerName:** nombre del cliente que solicita los servicios de *Recover.*
- **documentationTeam:** nombre del equipo que genera la documentación (en este caso Sogeti).
- **testingCompanyName:** nombre de la compañía que ofrece los servicios de *testing* (en este caso Sogeti).
- **testingTeam:** nombre del equipo que realiza el *testing* mediante la herramienta *Recover* (en este caso Sogeti).

UseCase

Representa un caso de uso y está compuesto de diversos casos de prueba.

- **name:** nombre del caso de uso

TestCase

Representa un caso de prueba y está compuesto de diversos elementos de test.

- **name:** nombre del caso de prueba

TestProgram

Representa un programa de test, tiene un estado inicial (*Fixture)* y está asociado a diversos casos de test.

- **name:** nombre del programa de test

Fixture

Representa el estado inicial de un programa de test y consiste en un conjunto de elementos de estado (*StateElement),* que indican la situación por defecto, previa a realizar un caso de prueba de un programa de test.

TestElement

Representa un elemento de test; estos elementos constituyen las acciones que se realizan en un caso de prueba y pueden ser de diversos tipos:

- **AssertTrue:** Indica la acción de comprobación de una certeza.
- **AttributeOfObject:** Representa un atributo de un objeto del test.
- **ParameterOfOperation:** Representa un parámetro de una operación.
- **StateElement:** Engloba los elementos que modifican el estado.

StateElement

Representa un elemento de estado (subconjunto de elemento de test); mediante estos elementos, se representan las acciones que modifican el estado de ejecución de un caso de prueba y una *fixture* y los tipos disponibles son:

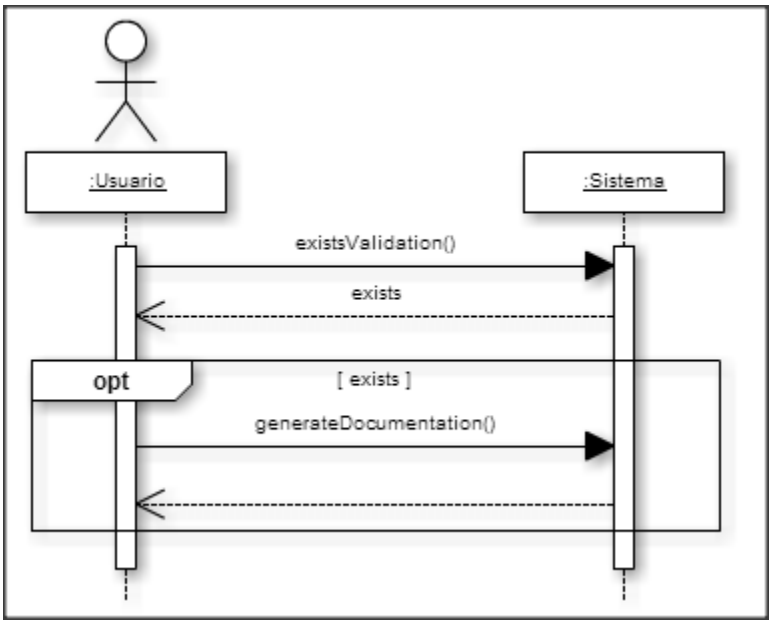
- **OperationOccurrence:** Indica la ejecución de la operación de una clase del modelo.
- **TestLink:** Indica la creación de una asociación entre dos objetos de la prueba.
- **VariableAssignment:** Indica la asignación de un valor a una variable de una clase del modelo.
- **TestObject:** Indica la creación de un objeto en la prueba.
- **Comment:** Representa un comentario de la prueba.

4.5 MODELO DE COMPORTAMIENTO

En este apartado, se definen las operaciones actor-sistema que se realizan en base a los casos de uso del apartado 8.3 y sus contratos. En un sistema orientado a objetos, las diferentes acciones a realizar, se hacen efectivas mediante operaciones invocadas sobre estos objetos.

A continuación, se muestra el modelo de comportamiento, que muestra la secuencia de operaciones que debe realizar un actor contra el sistema para conseguir un resultado de los casos de uso.

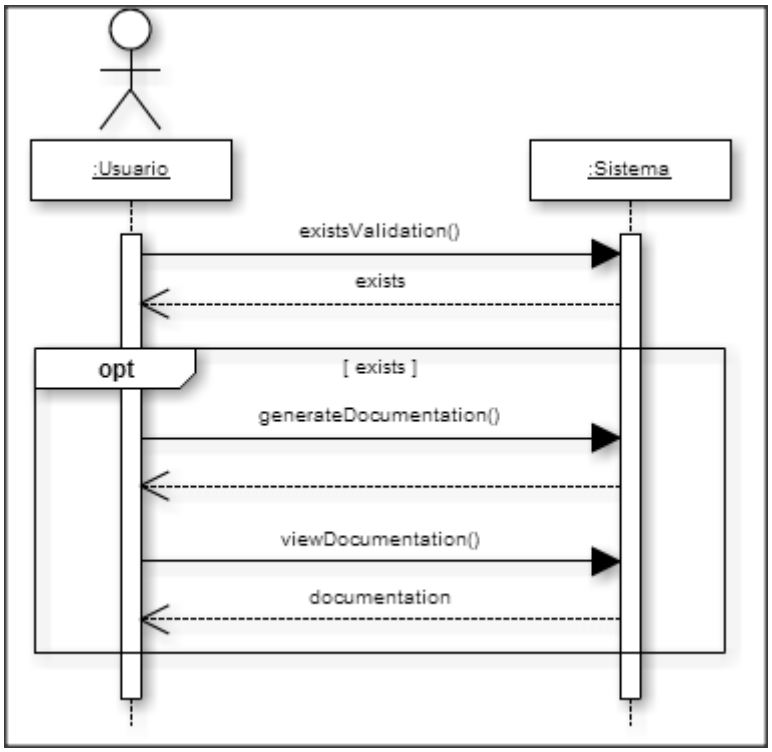
4.5.1 Generar documentación



existsValidation()	
Pre	El usuario ha hecho *login*
Post	Se retorna *cierto* si existe una validación o *falso* si no existe

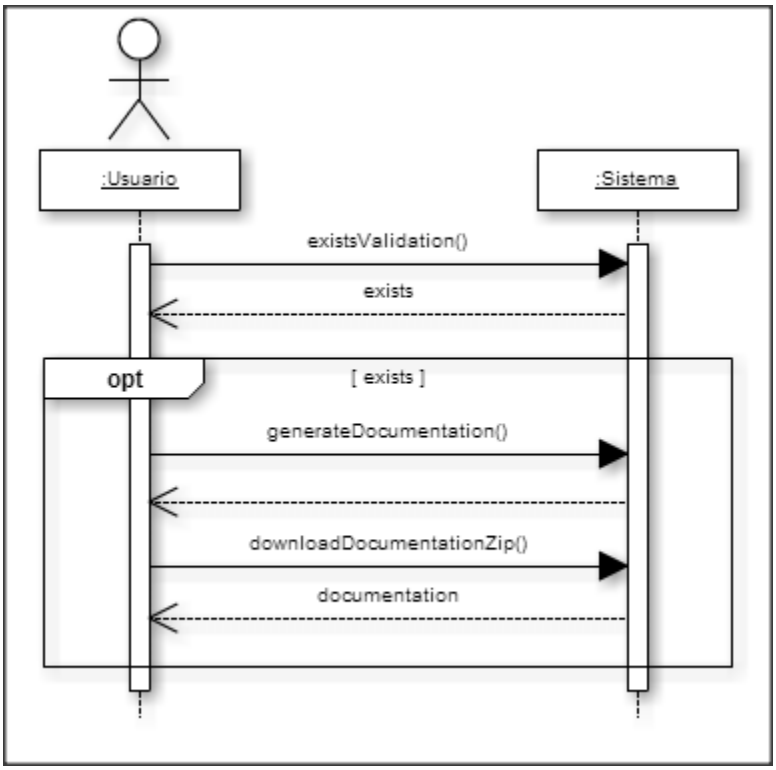
generateDocumentation()	
Pre	El usuario ha realizado una validación satisfactoria
Post	Se genera una documentación y se guarda en el sistema

4.5.3 Visualizar documentación



viewDocumentation()	
Pre	Existe una documentación previamente generada
Post	Se retorna la documentación

4.5.4 Descargar documentación



downloadDocumentationZip()	
Pre	El usuario ha hecho *login* y se ha generado una documentación correctamente
Post	Se retorna la documentación web en formato 'zip'

5 DISEÑO

En esta sección, se muestran todos los aspectos del diseño de *Recover*, pasando por la arquitectura lógica/física utilizada por la herramienta y el diseño de los nuevos componentes a añadir para cumplir con las funcionalidades especificadas en la fase de análisis de requisitos (ver sección 7).

5.1 ARQUITECTURA LÓGICA

Cuando hablamos de arquitectura lógica, nos referimos a la descripción de los componentes lógicos y cómo interactúan entre ellos sobre la arquitectura física, para lograr el funcionamiento deseado de una solución o herramienta.

En el caso de *Recover* la arquitectura lógica que se sigue es la 3 capas, una arquitectura clásica donde se pretende diferenciar en niveles separados los diferentes aspectos de la aplicación; por tanto, todo el diseño de las nuevas funcionalidades deberá basarse en esta arquitectura.

En la Figura 16, se muestra el esquema principal de la arquitectura.

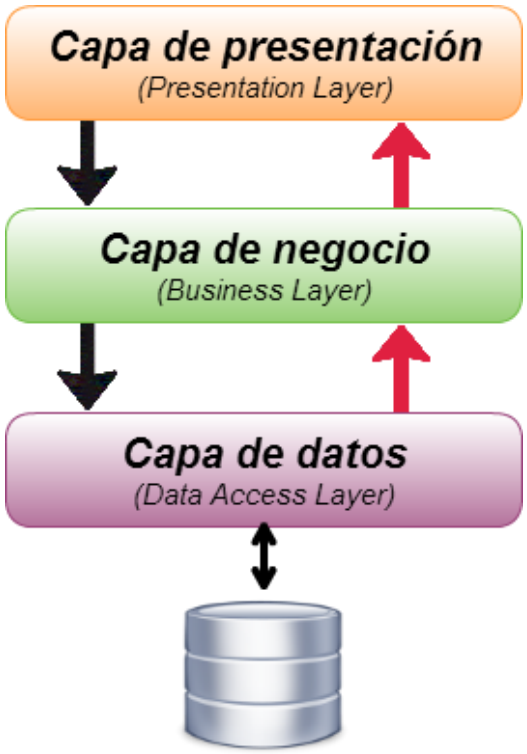


Figura 16.- Arquitectura lógica en 3 capas

El hecho de usar este tipo de arquitectura, tiene una serie de beneficios notables a la hora de desarrollar *software* y por ello es una de las más utilizadas, sin embargo, aun teniendo numerosas ventajas, los inconvenientes también están presentes (ver Tabla 1).

Ventajas
• **Un cambio en la interfaz de usuario, lógica o base de datos solo afecta a su capa concreta, sin necesidad de modificar las demás.** • **Una capa es completamente reemplazable, sin afectar al funcionamiento de las demás.** • **Las capas son reutilizables.** • **A nivel de datos, en caso de aumentar la complejidad o necesidad de procesado, permite escalabilidad sin afectar el funcionamiento general.**
Inconvenientes
• **Pérdida de eficiencia por la comunicación entre capas.** • **Dificultad para probar el funcionamiento correcto, debido a la cantidad de elementos que se comunican** • **Funcionamiento complejo**

Tabla 1.- Ventajas e inconvenientes de la arquitectura de 3 capas

A continuación se describe cada una de las capas de la arquitectura:

Capa de presentación

Es la capa encargada de la interacción usuario-aplicación y sus principales funciones, consisten en mostrar la información de la aplicación al usuario, realizar validaciones de datos de entrada de éste sobre la aplicación e interpretar sus acciones; su comunicación se produce con la capa de negocio, a la que le transmite información de la petición proporcionada por el usuario para fines concretos de la aplicación y negocio le devuelve los datos de respuesta, que se mostrarán al usuario en base a la petición.

Capa de negocio

En ocasiones también llamada dominio, es la capa encargada de la lógica de la aplicación y de la implementación de las funcionalidades concretas que esta ofrece; situada en medio de la capa de presentación y de datos, hace de intermediario entre estas dos, realizando los cálculos necesarios para la aplicación a partir de datos guardados y los introducidos por el usuario, validados por la capa de presentación. Esta capa controla totalmente el acceso a la capa de datos.

Capa de datos

Esta es, la capa encargada de la comunicación con los sistemas contenedores de datos de la aplicación o servicios; proporciona a la capa de negocio una abstracción a la hora de acceder a las fuentes de datos (sistemas de mensajería, bases de datos, etc).

5.2 ARQUITECTURA FÍSICA

La arquitectura física, muestra cuales son los componentes y cómo interactúan físicamente en una solución, es decir, la parte *hardware* o física que se encarga de alojar la aplicación y hacerla disponible a los usuarios finales.

En el contexto de *Recover,* contamos con arquitectura física típica de aplicaciones web basada en cliente-servidor (Figura 17). En esta arquitectura, se cuenta con varios dispositivos que se conectan a través de la red (clientes) a la aplicación en cuestión alojada en el servidor.

Figura 17.- Arquitectura física cliente-servidor

La Figura 18, presenta el esquema completo y detallado de la distribución de Recover; a la izquierda aparece el cliente, que a través de peticiones en internet se comunica con el servidor de producción de *Recover*. A nivel de cliente, éste tan solo ve la herramienta a partir de las tecnologías *HTML, CSS, JavaScript* con *JSP* y hace peticiones a partir de *AJAX*. Seguidamente en la parte derecha, se representa el servidor de producción, que contiene la aplicación de *Recover,* cuya lógica está implementada en *Java* y conecta a la base de datos de producción (en la imagen 'BD Recover'), que utiliza tecnología *MySQL*; Además, se cuenta con un servidor de *backup,* donde se realizan las copias de seguridad de la base de datos del servidor de producción, para evitar posibles problemas.

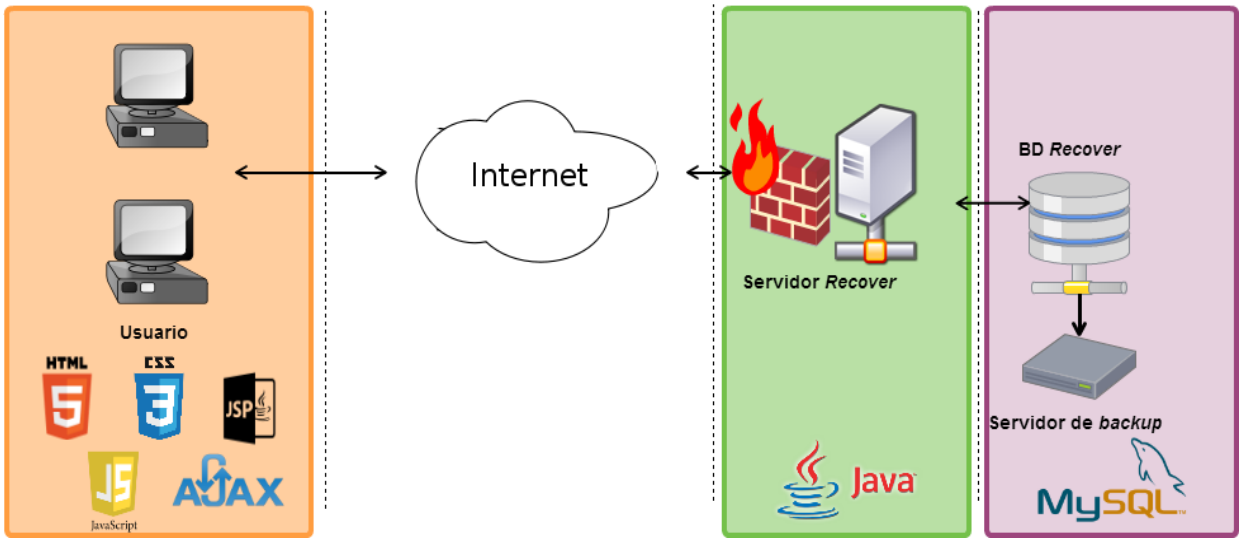


Figura 18.- Arquitectura física de Recover

5.3 CAPA DE PRESENTACIÓN

Contiene la parte visible para el usuario, es decir, la interfaz gráfica; mediante esta capa, se muestra la información al usuario y le permite ejecutar las acciones que desee sobre la herramienta; encargándose de intercambiar información con la capa de negocio.

En el caso de *Recover,* la capa de presentación esta implementada mediante la tecnología *JSP*, *HTML, CSS, JavaScript* y *AJAX* para la comunicación con el servidor.

5.3.1 Diagrama de navegación (WAE2)

A continuación, se muestra el diagrama principal de navegación entre las páginas de cliente y del servidor, así como la forma de interactuar entre ellas para cumplir con la funcionalidad de generación de documentación.

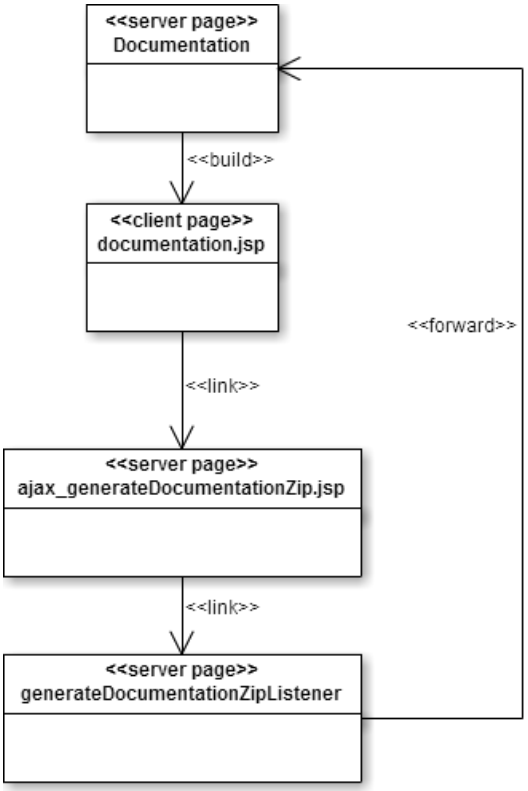


Figura 19.- Diagrama de navegación WAE2

5.3.2 Diagramas de secuencia

El diagrama de secuencia Figura 20, muestra el flujo completo que se realiza internamente entre los componentes del diagrama de clases de la Figura 19, a partir de la acción del usuario.

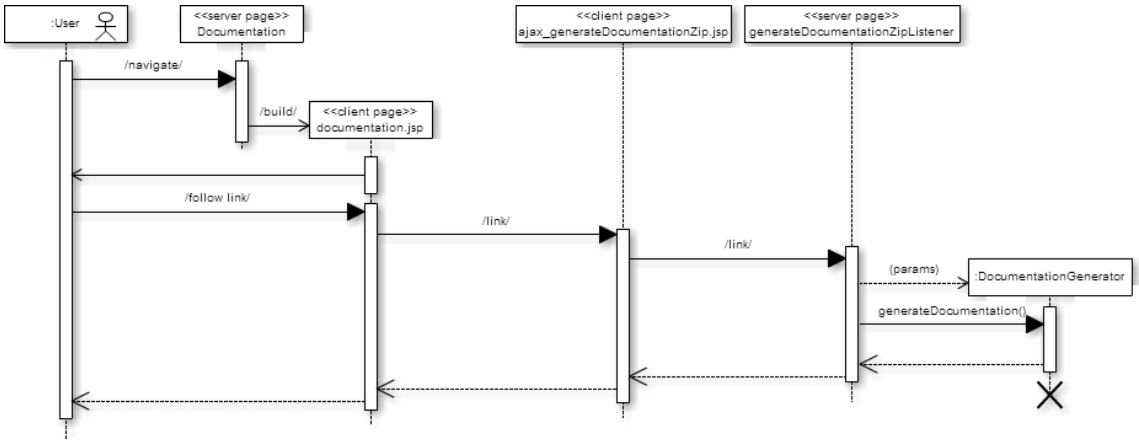


Figura 20.- Diagrama de secuencia de interacción la cliente-servidor

Se puede observar el funcionamiento, dónde un usuario del sistema navega a la página de documentación, se le muestra dicha página y una vez seleccionada la opción de generar documentación, se inicia la petición *AJAX* con su *Listener* correspondiente; esta acción desencadena en una serie de operaciones en el dominio de la aplicación, que una vez finalizadas, retornan el resultado al usuario.

5.3.3 Diseño de la interfaz de generación

En cuanto a la interfaz de usuario, se deberá incluir en la pestaña actual de documentación de *Recover,* un apartado que permita al usuario generar la documentación web. En la Figura 21, se muestra el diseño de este apartado, donde se ha enfocado principalmente a la usabilidad, añadiendo el mínimo número de opciones, cuya funcionalidad es claramente intuitiva.

Figura 21.- Diseño del apartado de generación de documentación web

Todas las funcionalidades mencionadas a añadir, parten de este apartado (Generar, visualizar y descargar documentación). Una vez realizada una validación de los casos de prueba en *Recover*, este apartado será visible en la pestaña documentación, donde a partir del botón "Generar especificación HTML" se ejecutará el proceso interno que genera la documentación web íntegra (ver Figura 19 y Figura 20).

Una vez acabado el proceso, se mostrará al usuario un mensaje que le informará sobre el estado de la documentación, permitiéndole descargarla en formato '*zip*' en caso de desearlo (Figura 22).

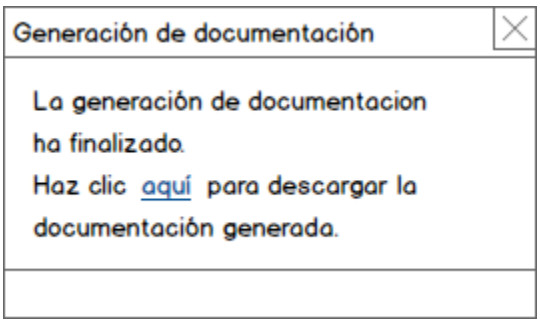


Figura 22.- Aviso de documentación generada correctamente

También se puede observar en la Figura 21, el enlace a la documentación web *on-line,* mediante este se pretende tener un método rápido de visualizar la documentación una vez generada, sin necesidad de descargarla y a partir de esta opción, se obtiene la documentación directamente del servidor sobre la última documentación generada.

5.3.4 Diseño de la documentación web

En este apartado, se detalla la interacción entre las páginas generadas por el proceso automático, así como el aspecto visual mediante *mockups* del resultado que tendrá cada una de ellas.

Mapa navegacional

El siguiente mapa navegacional, muestra el flujo que el usuario puede realizar al visualizar la documentación autogenerada, partiendo desde la página principal (index.html). Por simplicidad del diagrama no se incluyen las navegaciones hacia la página anterior desde cualquier página y tampoco las navegaciones a los índices (coloreados en azul), ya que se puede acceder a estos, desde cualquiera de las demás páginas mostradas en el diagrama.

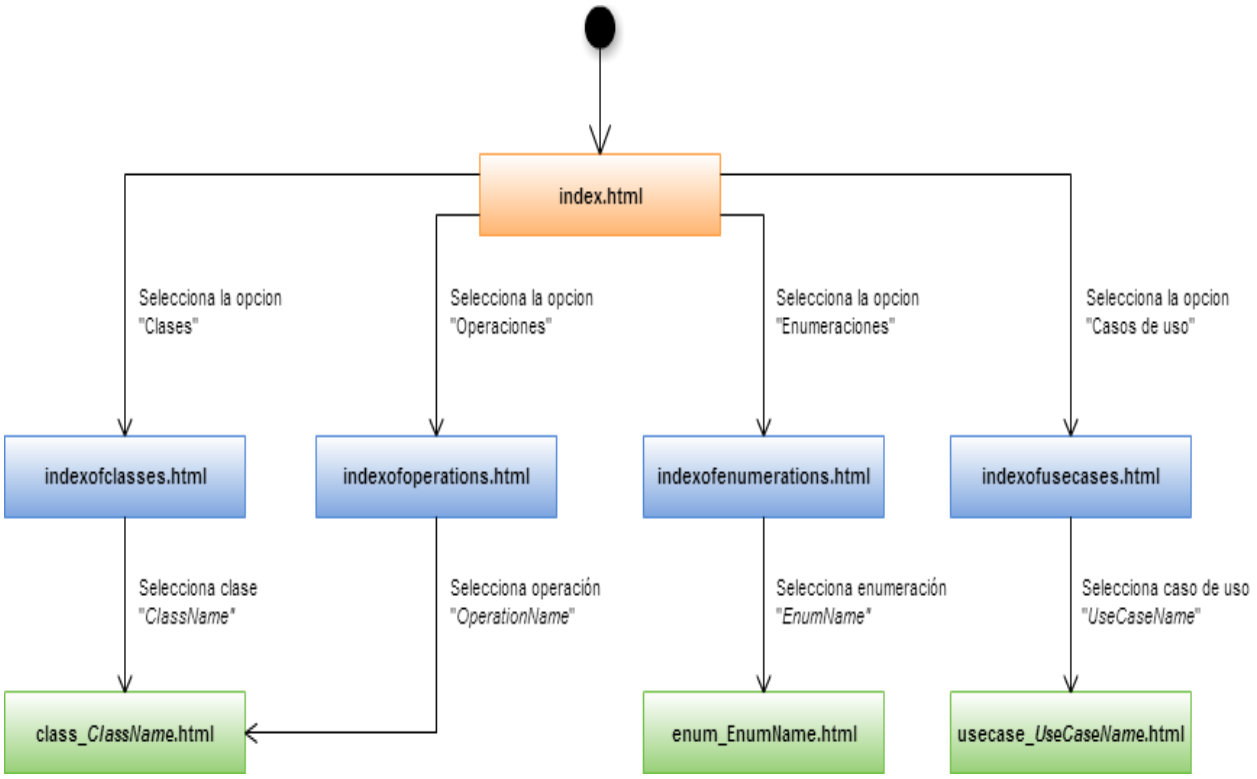


Figura 23.- Mapa navegacional de la documentación web

A continuación, se muestra el diseño realizado para las diferentes páginas de la documentación web:

Página principal

Esta es la pantalla principal que se muestra al entrar a la documentación, se puede observar en primer lugar la sección "Información del proyecto", donde se dispondrá de diferentes datos informativos referentes al proyecto, cliente y hora de generación entre otros; además, se incluye una tabla de contenidos, con esta se permite la navegación hacia los diferentes apartados de la documentación a partir de sus enlaces, los cuales contienen toda la información de los artefactos en *Recover*.

Figura 24.- Página principal de la documentación web

Índice de clases

En la página de índice de clases, se muestran todas las clases que contiene *Recover* con su respectivo enlace, fruto del alineamiento automático del modelo con los casos de prueba, también se muestra información simplificada de la documentación y enlaces a las diferentes secciones de esta.

Figura 25.- Página del índice de clases

Índice de casos de uso

La vista siguiente, contiene los diferentes casos de uso que los usuarios han introducido sobre la herramienta *Recover*.

Figura 26.- Página del índice de casos de uso

Índice de enumeraciones

Esta vista, contiene las diferentes enumeraciones que se han añadido sobre el modelo de la herramienta *Recover* al aplicar las acciones de alineamiento.

Figura 27.- Página del índice de enumeraciones

Índice de operaciones

En la siguiente vista, se pueden observar todas las operaciones que contiene el modelo de *Recover* en cada una de las clases.

Figura 28.- Página del índice de operaciones

Página de clase

Una vez consultamos la información de una clase concreta desde el índice de clases, obtenemos la siguiente página. Al principio, muestra la información simplificada del proyecto y enlaces a otras secciones de la documentación, pero además, contiene una descripción completa de la clase: diagrama de clases filtrado, atributos, asociaciones y operaciones. Esta página incluye la descripción autogenerada en lenguaje natural del diagrama de clases y el código *OCL* de las operaciones.

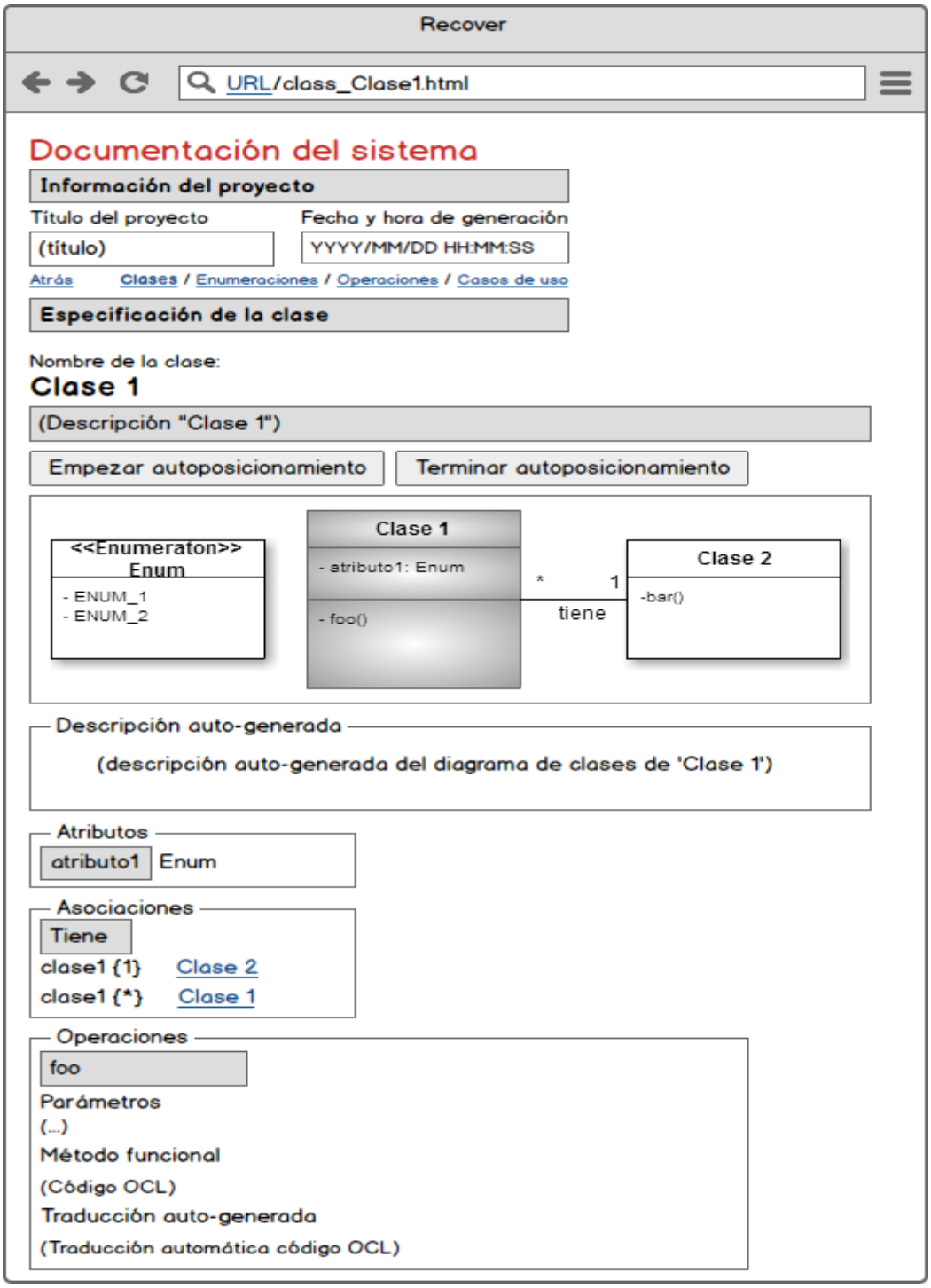


Figura 29.- Página de clase específica

Página de caso de uso

Cuando se utiliza un enlace del índice de casos de uso nos aparece la página siguiente. En ella se puede apreciar el resumen del proyecto y enlaces a las diferentes secciones, así como la descripción del caso de uso y los diagramas de actividad y de secuencia autogenerados.

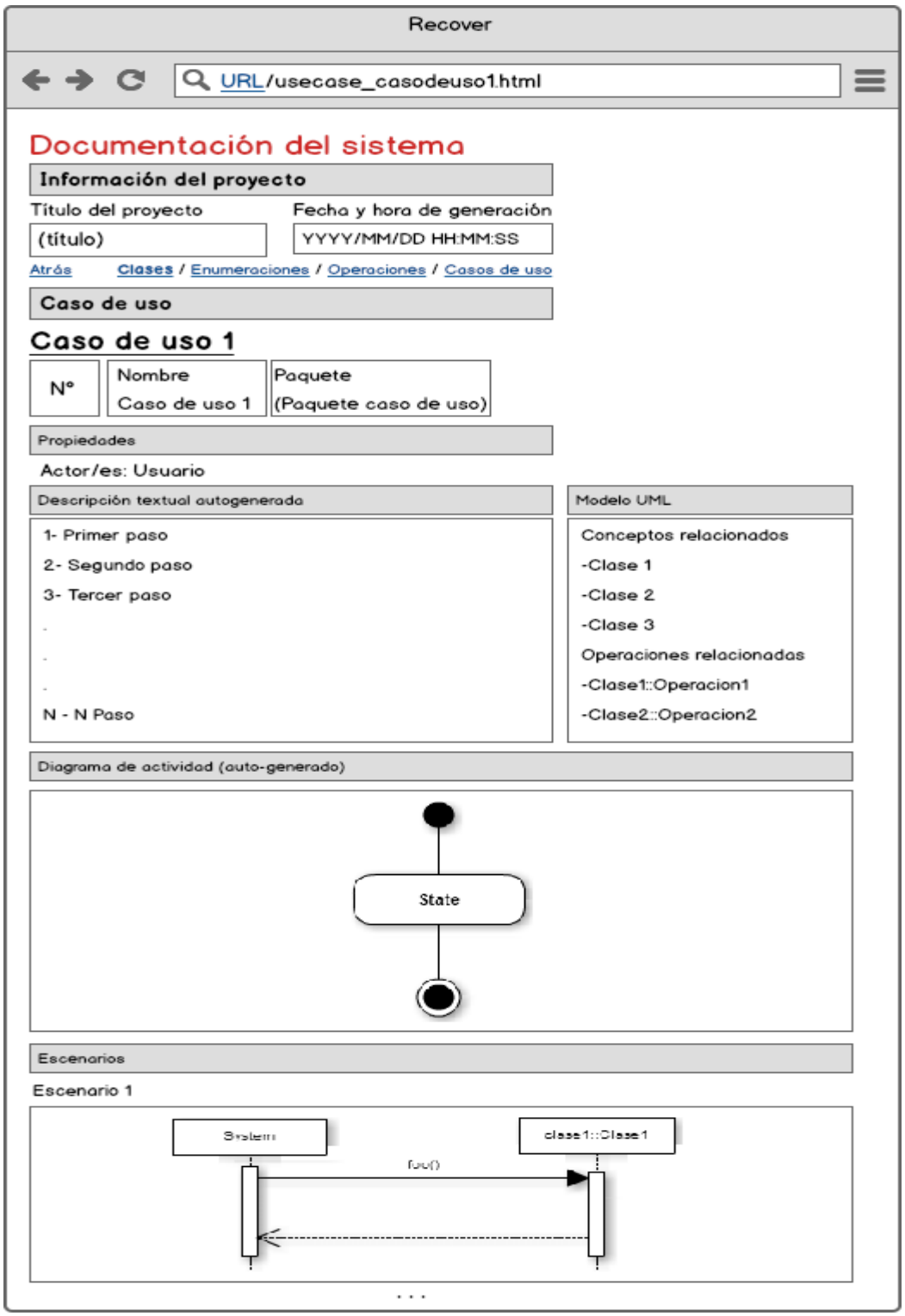


Figura 30.- Página de caso de uso específico

Página de enumeraciones

Al acceder a una enumeración en concreto desde el índice de enumeraciones, nos aparece la siguiente pantalla. En esta se muestra la información básica del proyecto, los enlaces a las diferentes secciones de la documentación y una lista de los diferentes valores que puede adoptar la enumeración seleccionada.

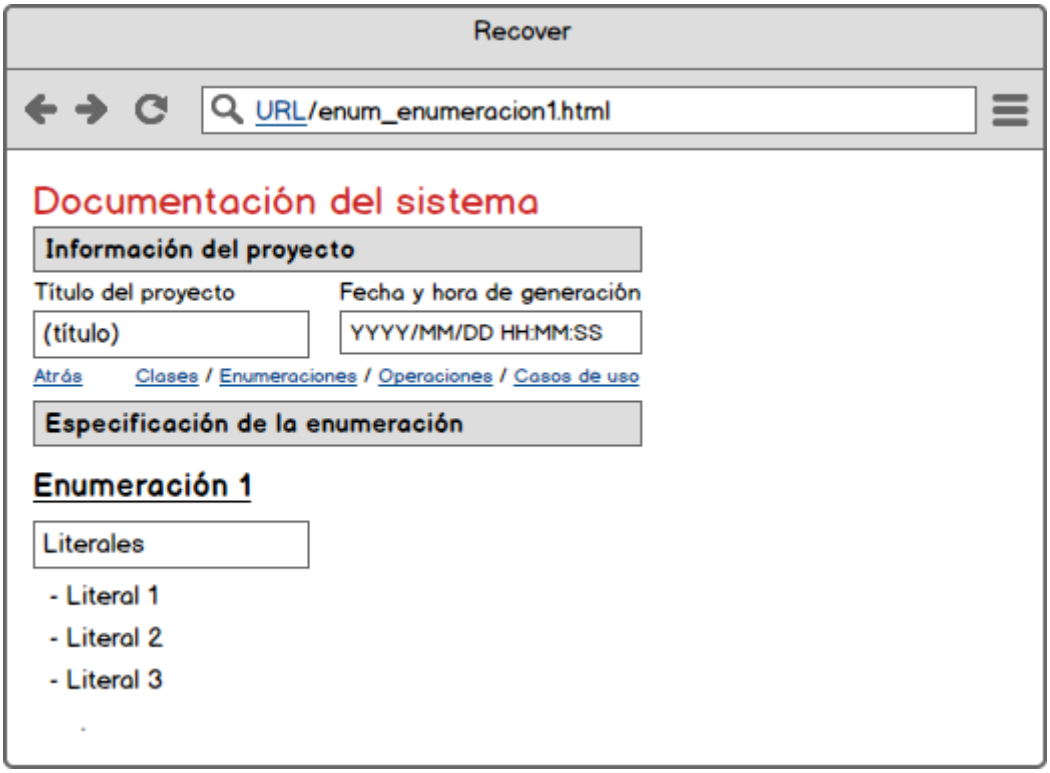


Figura 31.- Página de enumeración específica

5.4 CAPA DE NEGOCIO

Esta capa, es la encargada de manejar la lógica de la aplicación *Recover*; trata las acciones a realizar mediante el conjunto de entradas, que proporciona el usuario desde la capa de presentación y las ejecuta, además contiene el domino de la aplicación y hace de intermediario entre las capas de presentación y datos.

La tecnología utilizada para la implementación del dominio es Java y las peticiones de usuario por parte de la capa de presentación se reciben mediante *Listeners*.

5.4.1 Documentación web autogenerada

La generación automática de la documentación web, comienza cuando se produce una petición AJAX por parte del usuario desde la capa de presentación (ver sección 9.2.1.2), una vez realizada, el servidor procesa la petición en la capa de negocio.

5.4.1.1 Diagramas de clases

Figura 32.- Clase principal de la documentación web

5.4.1.2 Diagramas de secuencia

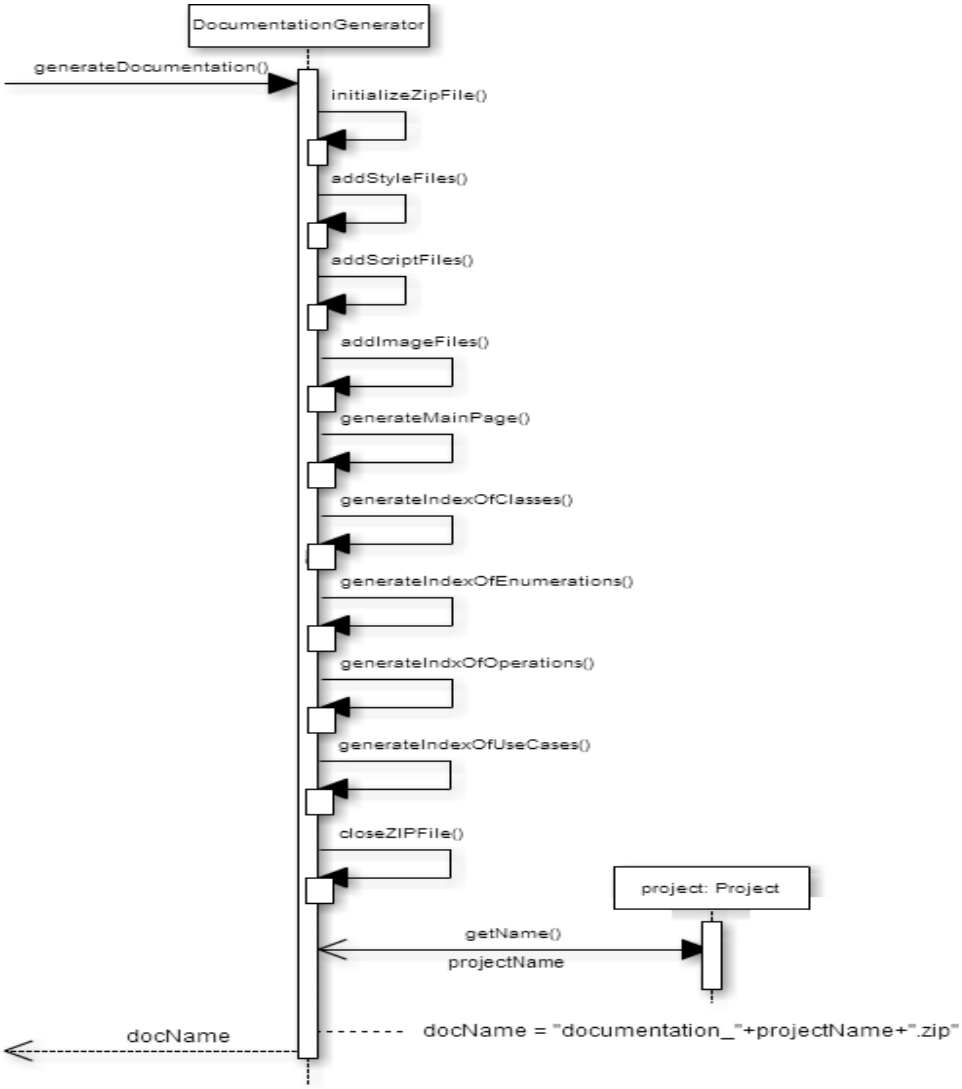
A continuación, se muestra mediante diagramas de secuencia el comportamiento que tendrán las operaciones de la clase "DocumentationGenerator", las cuales son las encargadas de generar cada una de las páginas de la documentación.

En esta sección, se han incluido los diagramas de las operaciones más complejas y que requieren una mayor lógica sobre el proceso de generación automático (para ampliar información sobre los diagramas ver el Anexo B).

5.4.1.2.1 Generar Documentación

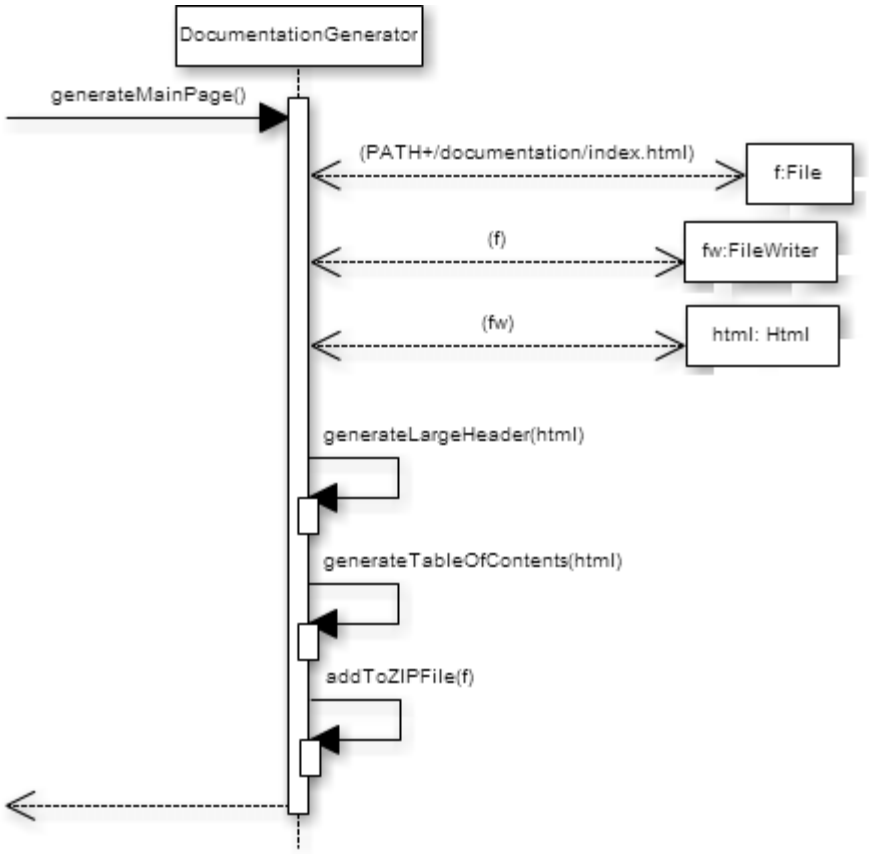
GenerateDocumentation

El siguiente diagrama, muestra el flujo general, mediante el cual se genera la documentación web. En primer lugar, se inicializa el fichero *'zip'* donde se guardarán todos los ficheros generados, se añaden las hojas de estilos, *scripts* e imágenes. Seguidamente se generan las páginas principal y de índices con sus respectivas páginas específicas y finalmente, se guarda el fichero *'zip'* y se retorna el nombre del fichero creado.



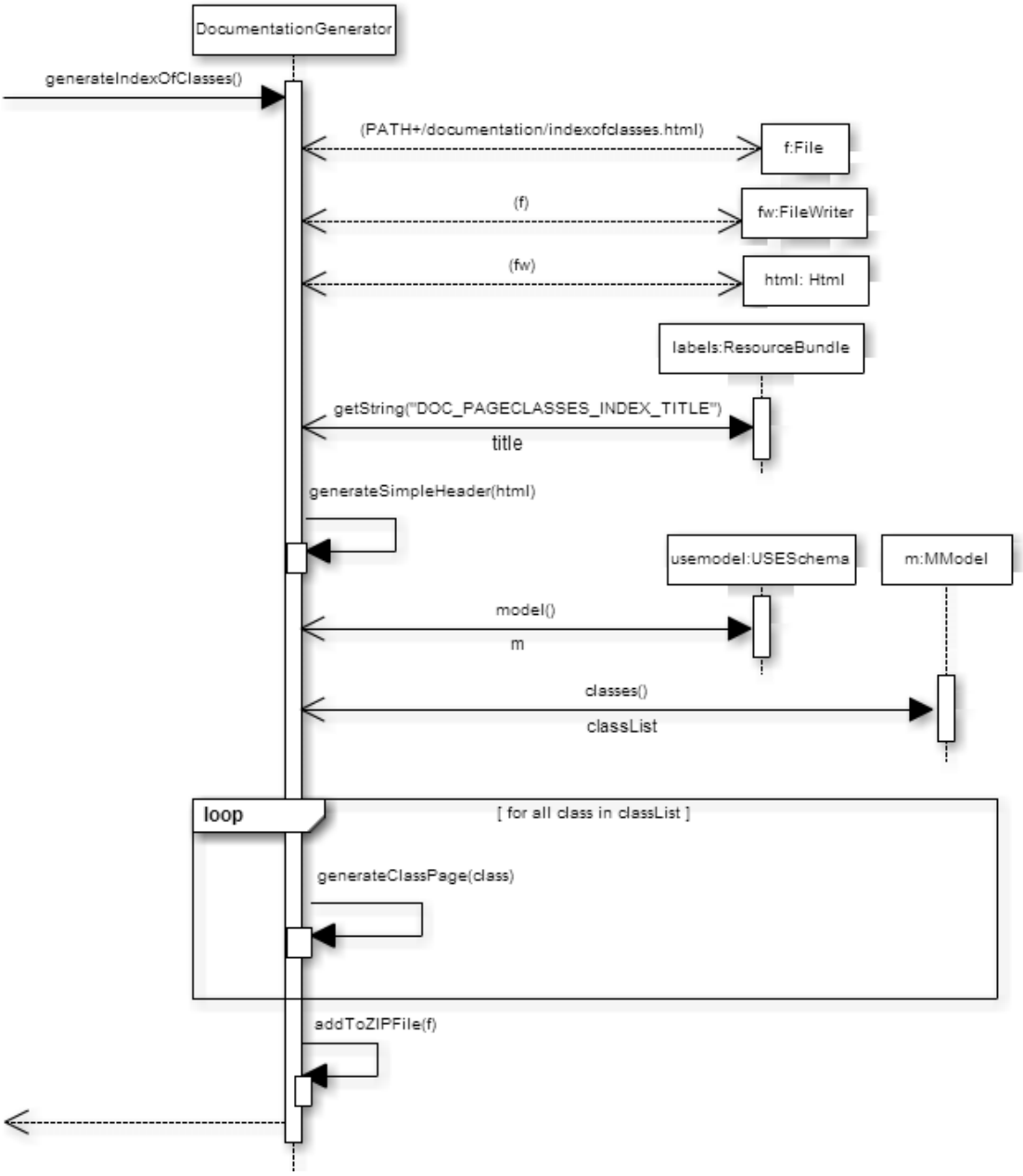
GenerateMainPage

Esta función, se encarga de generar la página principal de la documentación; para ello, genera un fichero al cual le añadirá un encabezado con la información detallada y una tabla de contenidos para navegar por los diferentes apartados (ver Figura 24). Al acabar, añade la página generada con nombre "index.html" al fichero *'zip'.*



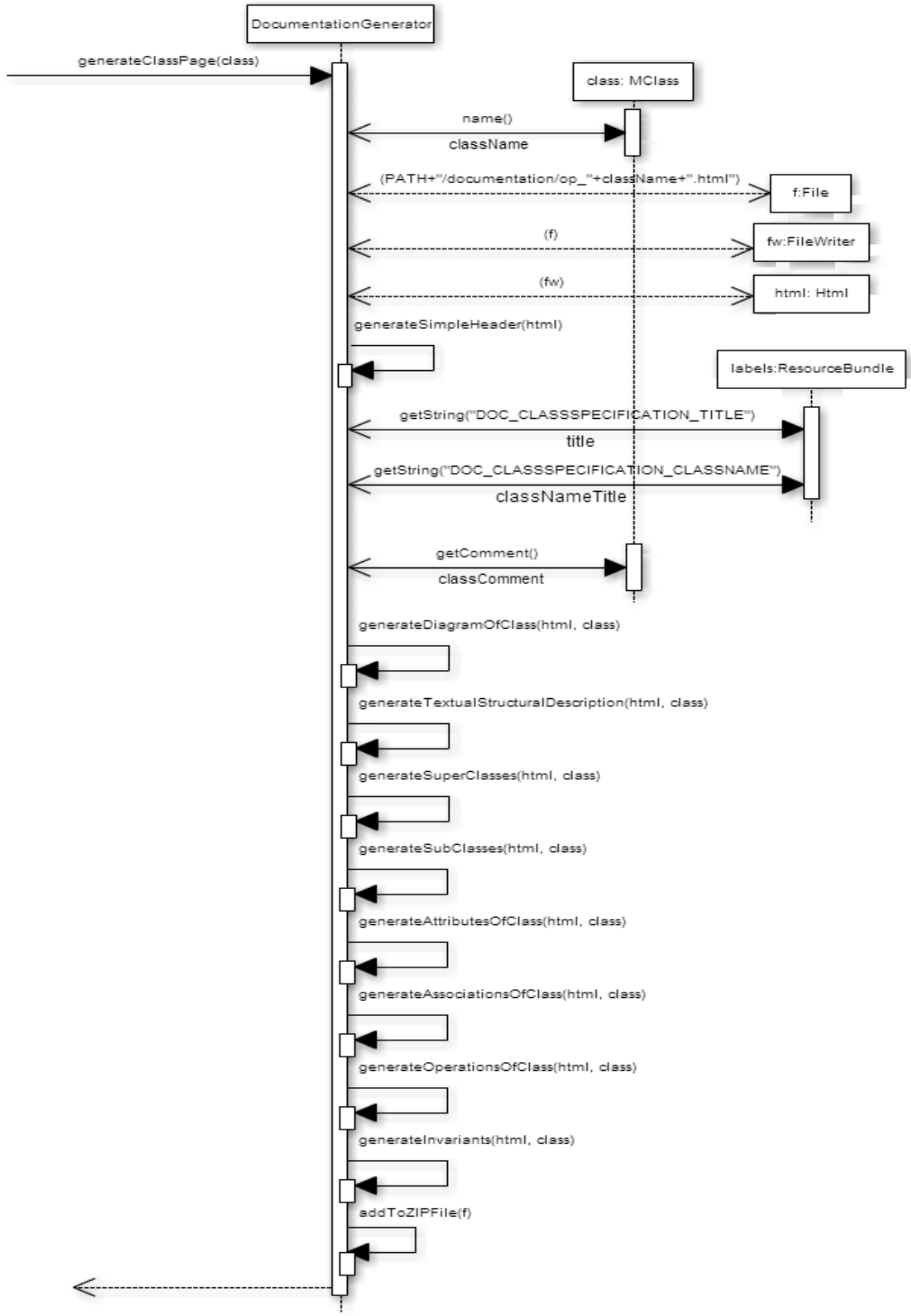
GenerateIndexOfClasses

En este diagrama, se muestra el proceso de generación del índice de clases. Primero se genera el fichero "indexofclasses.html", que contendrá un encabezado simple con la información básica de la documentación. Seguidamente, se incluye una lista de enlaces a cada una de las páginas de las clases del modelo en *Recover,* que se generan a partir de la llamada a la función *generateClassPage* y finalmente, se añade el fichero creado al fichero *'zip'*.



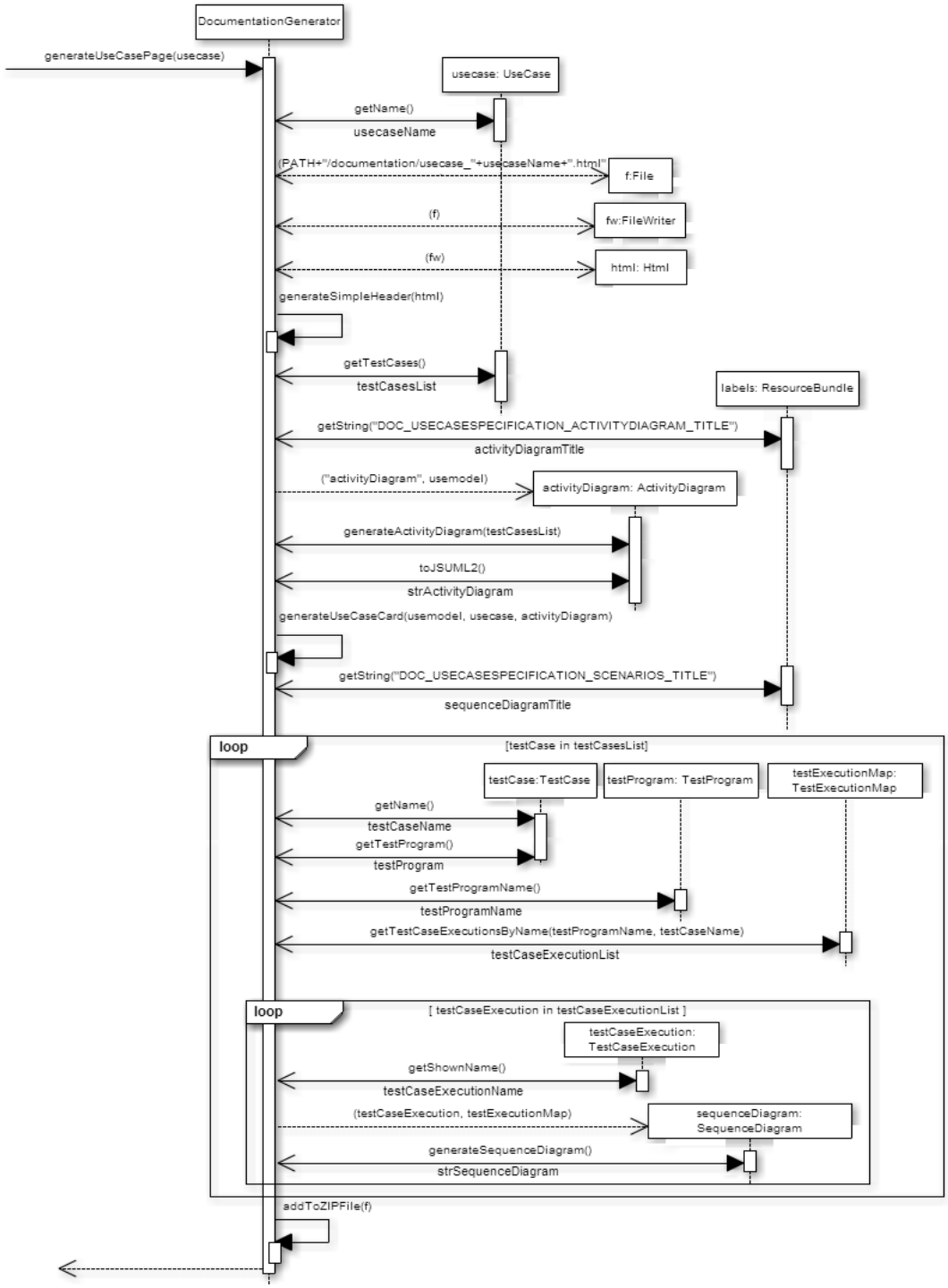
GenerateClassPage

La siguiente función, es la encargada de generar la página de una clase específica con su correspondiente (ver Figura 29); para ello, en el fichero que crea con el nombre "op_className.html" (donde *'className'* es el nombre de la clase específica), añade un encabezado simple con la información del proyecto, el diagrama de clases con los conceptos directamente relacionados con la clase, la traducción a lenguaje natural del diagrama de clases y la descripción de los atributos, relaciones y operaciones de la clase concreta y finalmente, se añade el fichero creado al *'zip'.*



GenerateUseCasePage

Esta operación, se encarga de generar la página de un caso de uso concreto (ver Figura 30); primero se crea el fichero web con el nombre "usecase_usecaseName.html" (donde *'usecaseName'* es el nombre del caso de uso específico); a continuación, se genera el diagrama de actividad a partir de los casos de prueba y se añade la descripción del caso de uso mediante la función *generateUseCaseCard.* Finalmente, con los casos de test de un caso de uso y el resultado la última ejecución realizada en *Recover, se* genera el diagrama de secuencia de los casos lógicos/físicos. Al acabar se añade el fichero web al *'zip'.*



5.4.2 Traducción de UML y OCL a lenguaje natural

Actualmente, en *Recover* se está utilizando la librería *USE (UML-based Specification Environment),* basada en la versión 1.3 del *metamodelo* de *UML* del *OMG* para la representación interna de los diagramas *UML* y para la sintaxis *OCL*. Por tanto, la funcionalidad de traducción automática, se plantea como una ampliación sobre esta librería.

Para conseguir entender en profundidad el funcionamiento de la librería, se ha realizado un estudio previo, para saber por dónde abordar el problema e ir directamente al diseño de las partes que pueden influir en nuestro proceso de traducción (ver Anexo D). Dado que el diseño o documentación técnica de esta librería no está disponible, se ha tenido que investigar la organización de la estructura de clases completa y sus relaciones, para entender el papel de cada clase en la representación *UML* y *OCL.*

Concretamente, para los diagramas *UML,* se pretende hacer una representación textual completa del diagrama, mostrando para cada clase sus atributos, operaciones, asociaciones, superclases y subclases.

En cuanto a la traducción de *OCL,* la intención es realizarla sobre un subconjunto de expresiones, que permiten representar cualquier tipo de operación deseada y que serán las utilizadas para especificar las operaciones por parte de los usuarios de *Recover*.

5.4.2.1 Traducción UML

La traducción a lenguaje natural de *UML* cubre el área de la representación textual del diagrama de clases que utiliza *Recover.* Para realizar esta funcionalidad, se parte de la base del estudio realizado sobre la librería use *USE; a*l analizar el paquete encargado de la representación interna de los diagramas *UML,* se observa, que cuenta con todos los aspectos necesarios para realizar la traducción a lenguaje natural, por tanto, no hará falta ninguna modificación de las clases del paquete.

Sin embargo, sí será necesario diseñar los diagramas de secuencia de la navegabilidad entre clases, que permita extraer y transformar toda la información que queramos mostrar de la jerarquía de clases de la librería.

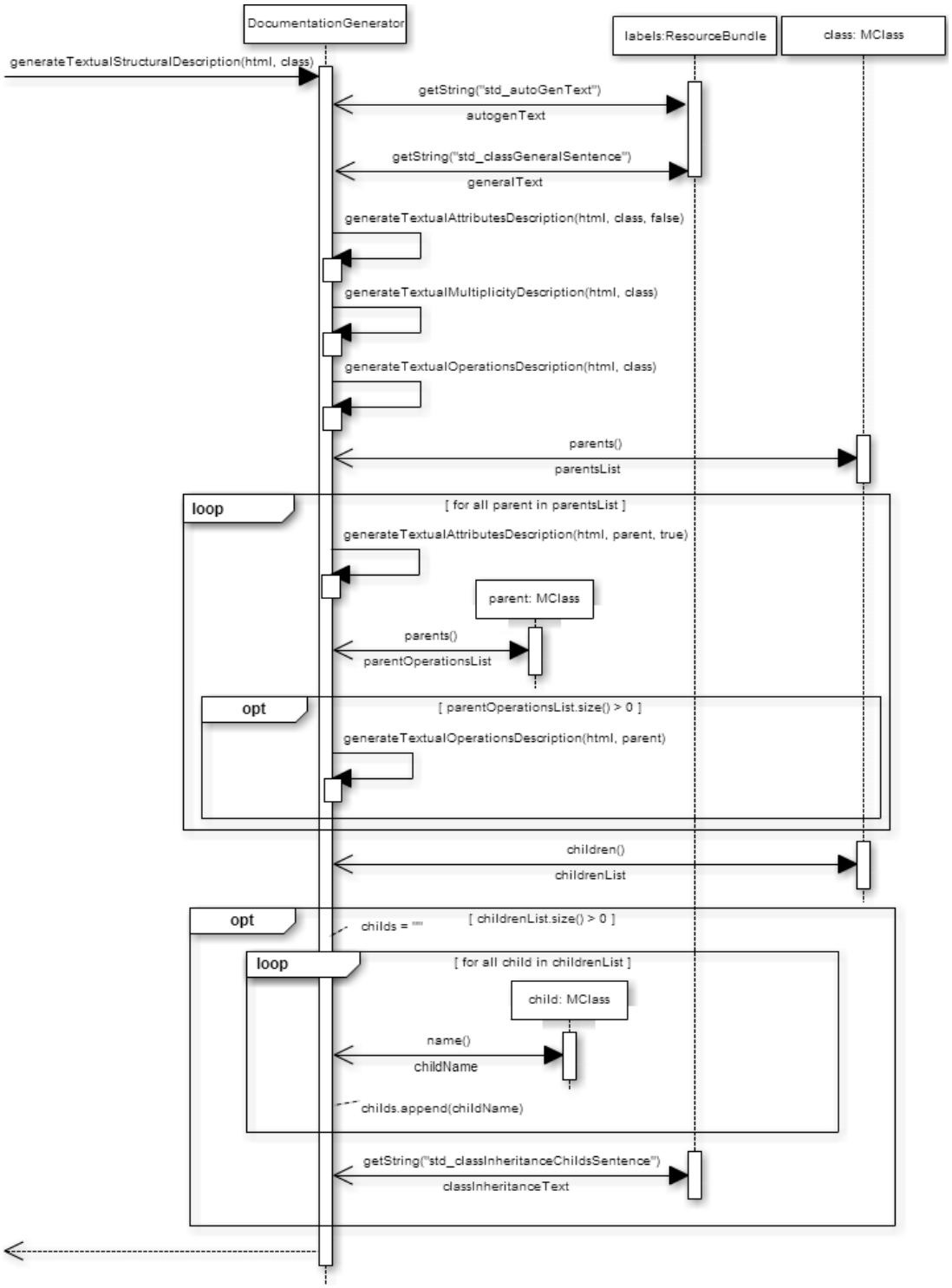
Esta funcionalidad, estará disponible en cada una de las páginas específicas de cada clase en la documentación (ver Figura 29 en la sección 'Descripción auto-generada'), destacando de cada una las siguientes características (en el mismo orden descrito a continuación):

- Atributos de la clase
- Asociaciones directas con otras clases y su tipo de multiplicidad
- Operaciones propias que puede realizar
- Operaciones que puede realizar como consecuencia de herencia de otra clase
- Tipos que puede adoptar como consecuencia del polimorfismo por ser padre de otra clase.

Diagramas de secuencia

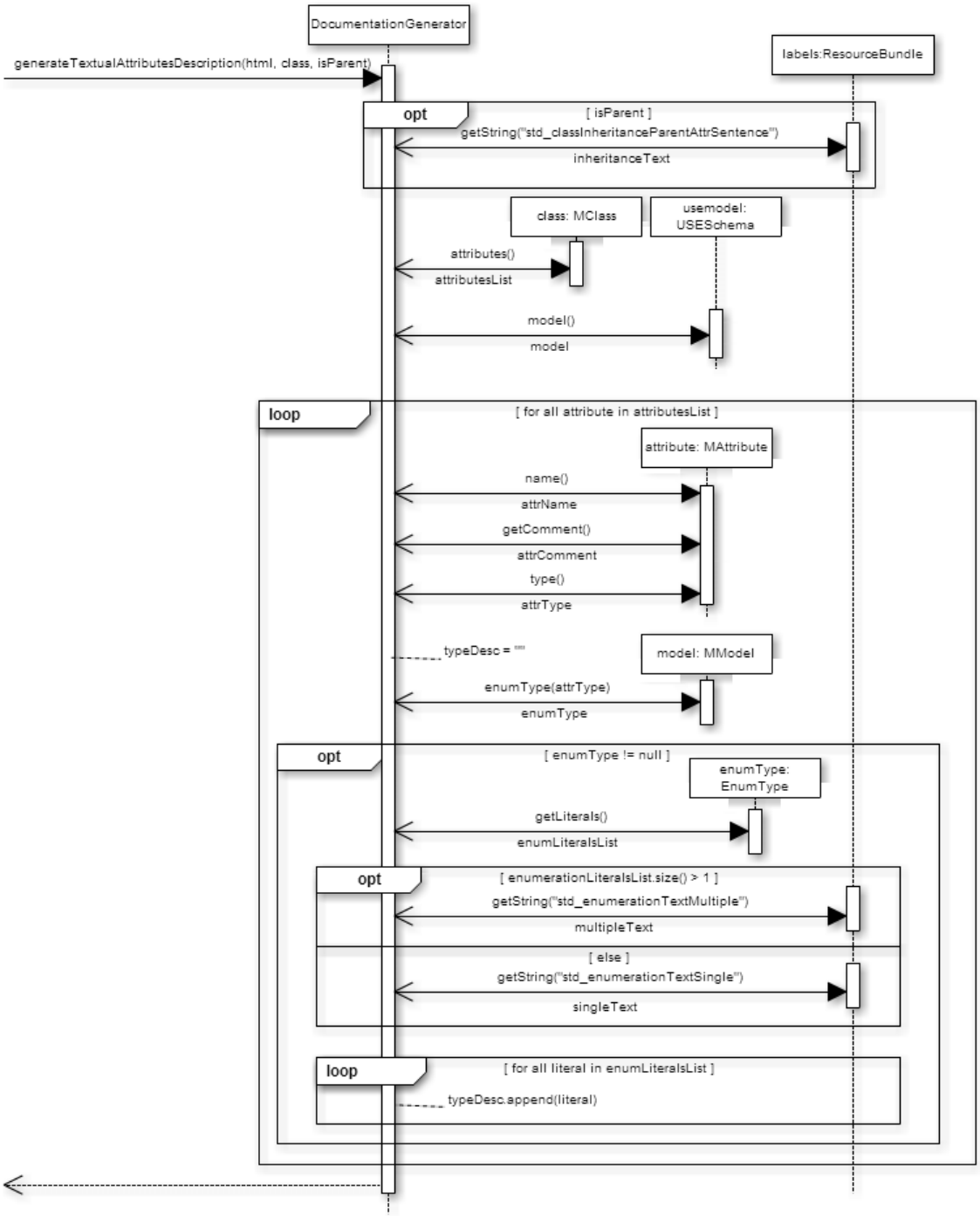
GenerateTextualStructuralDescription

En el siguiente diagrama, se puede ver el flujo de la función encargada de generar cada una de las partes de la traducción y observar las diferentes llamadas a funciones que traducen las partes de la clase concreta *class* (atributos, relaciones/multiplicidades y operaciones), las clases padre (atributos y operaciones) y una enumeración de las clases hijo, para indicar, qué tipos de instancia puede ser la clase *class*.



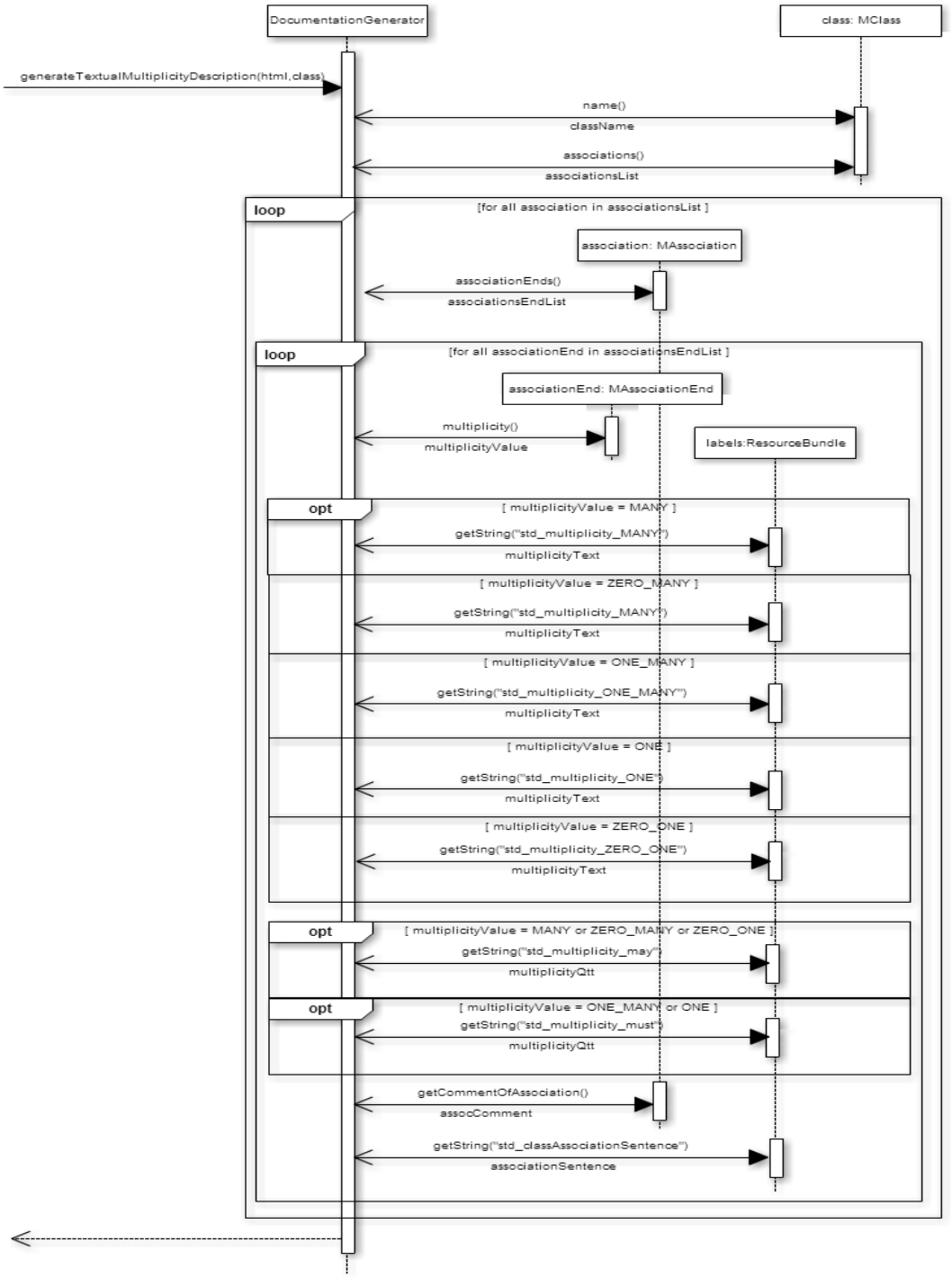
GenerateTextualAttributesDescription

Esta función, realiza la tarea de traducir todos los atributos de una clase concreta.; para ello, recupera del modelo la lista de todos los atributos de la clase y sobre cada uno de ellos extrae el nombre, comentario y tipo. Si el tipo es enumeración, se extraen y se muestran los valores literales que puede adoptar el atributo y se realiza su traducción en función de si contiene más de un valor posible o no.



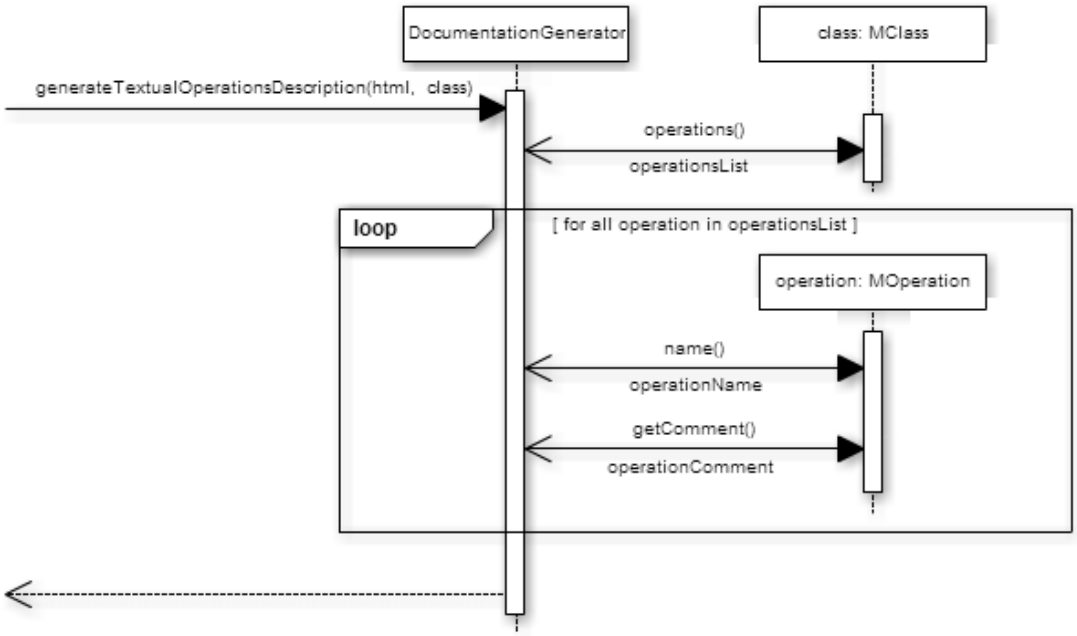
GenerateTextualMultiplicityDescription

Para saber la cantidad de instancias de una clase, relacionadas con otras instancias del diagrama de clases, habrá que recorrer cada una de sus relaciones y extraer la información necesaria de ella. La función *generateTextualMultiplicityDescription,* se encarga de traducir las multiplicidades de las relaciones de una clase concreta *class* y a partir de las asociaciones (*MAssociation),* se extraen los extremos (*MAssociationEnd)* de cada una de las relaciones y en función de su multiplicidad (*, 1..*, 1, 0..1) se traduce de una manera u otra según convenga; el resultado de la traducción se complementa con el nombre de la asociación y sus comentarios.



GenerateTextualOperationsDescription

Dada una clase concreta *class,* esta operación se encarga de mostrar la representación textual de sus operaciones, recorriendo una a una (*MOperation)* y representándolas a partir de su nombre y comentarios asociados.



5.4.2.2 Traducción OCL

La jerarquía de clases, implicada en la representación del lenguaje *OCL* de la librería *USE,* es muy extensa, debido al gran número de operaciones que contiene este lenguaje.

En el caso de la traducción de *OCL,* aunque la arquitectura de la librería es completa y soporta todas las expresiones del lenguaje *OCL*, es necesario añadir modificaciones, para que la librería soporte la traducción a lenguaje natural de estas expresiones.

Las operaciones de este paquete sintáctico se dividen en dos grupos:

- *SOIL Statements*
- *OCL Expressions*

La diferencia entre estos dos grupos, consiste en que las sentencias *SOIL* son un tipo de operaciones implementadas en *ActionSemantics,* que se utilizan para modificar el estado de los objetos de un modelo; por el contrario, las expresiones *OCL,* son operaciones del tipo *query* que al evaluarse retornan un valor.

A continuación, se muestran para cada uno de los grupos, las expresiones sobre las que se centrará el diseño de este trabajo:

SOIL Statements		
Tipo	**Clase**	**Ejemplo de uso**
Creación de objeto	MNewObjectStatement	obj := **new** Tipo
Destrucción de objeto	MObjectDestructionStatement	**destroy** nombreObjeto
Creación de asociación	MLinkInsertionStatement	**insert**(obj1,obj2) **into** nombreAsociación
Borrado de asociación	MLinkDeletionStatement	**delete**(obj1,obj2) **from** nombreAsociación
Asignación a atributo	MAttributeAsignmentStatement	obj.atr **:=** *expresión*
Asignación a variable	MVariableAsignmentStatement	var **:=** *expresión*
Llamada a operación	MOperationCallStatement	obj.operación(arg1, …)
Condicional	MConditionalExecutionStatement	**if** *expresión* **then** x [else] **end**
Bucle	MWhileStatement	**for** var **in** *expresión* **do** x **end**

Tabla 2.- Listado de sentencias SOIL a traducir

OCL Expressions		
Operaciones sobre objetos o atributos		
Expresión	**Clase**	**Ejemplo de uso**
isDefined	Op_isDefined	x**.isDefined()**
isUndefined	Op_isUndefined	x**.isUndefined()**
Operaciones sobre colecciones		
Expresión	**Clase**	**Ejemplo de uso**
select	ExpSelect	*col*-**>select(x |** *expresión booleana***)**
reject	ExpReject	*col*-**>reject(x |** *expresión booleana***)**
union	Op_bag_union Op_bag_union_set Op_or-deredSet_union Op_se-quence_union Op_set_union Op_set_union_bag	*col*-**>union(***expresión colección***)**
intersection	Op_bag_intersection Op_bag_intersection_set Op_set_intersection Op_set_intersection_bag	*col*-**>intersection(***expresión colección***)**
asSet	Op_collection_asSet	*col*-**>asSet(***colección***)**
size	Op_collection_size Op_string_size	*col*-**>size()** *cadena*-**>size()**
count	Op_collection_count	*col*-**>count(***objeto***)**
includes	Op_collection_includes	*col*-**>includes(***objeto***)**
excludes	Op_collection_excludes	*col*-**>excludes(***objeto***)**
includesAll	Op_collection_includesAll	*col*-**>includesAll(***colección***)**
excludesAll	Op_collection_excludesAl	*col*-**>excludesAll(***colección***)**
isEmpty	Op_collection_isEmpty	*col*-**>isEmpty()**
notEmpty	Op_collection_notEmpty	*col*-**>notEmpty()**
sum	Op_collection_sum	*col*-**>sum()**
exists	ExpExists	*col*-**>exists(x |** *expresión booleana***)**
forAll	ExpForAll	*col*-**>forAll(x |** *expresión booleana***)**
isUnique	ExpIsUnique	*col*-**>isUnique(***expresión***)**
Operaciones booleanas y aritméticas		
Expresión	**Clase**	**Ejemplo de uso**

or	Op_boolean_or	x **or** y
and	Op_bolean_and	x **and** y
xor	Op_boolean_xor	x **xor** y
not	Op_boolean_not	**not** x
+, -, *, /, %	Op_number_add Op_number_sub Op_number_mult Op_number_div Op_integer_mod Op_integer_idiv	x **+** y
=, <>, <=, >=, <, >	Op_equal Op_notequal Op_number_less Op_number_greater Op_number_lessequal Op_number_greaterequal Op_string_less Op_string_greater Op_string_lessequal Op_string_greaterequal	x **=** y
implies	Op_boolean_implies	x **implies** y
if x **then** y [**else** z] **endif**	ExpIf	**if** x **then** y **[else** z**] endif**
Operaciones sobre clases		
Expresión	**Clase**	**Ejemplo de uso**
oclIsTypeOf	ExpIsTypeOf	*obj***.oclIsTypeOf**
oclAsType	ExpAsType	*obj***.oclAsType**
Operaciones sobre variables		
Expresión	**Clase**	**Ejemplo de uso**
let	ExpLet	**let** *var:Type = expresión* **in** *expresión (usando la variable var)*

Tabla 3.- Listado de expresiones OCL a traducir

Diagramas de clase

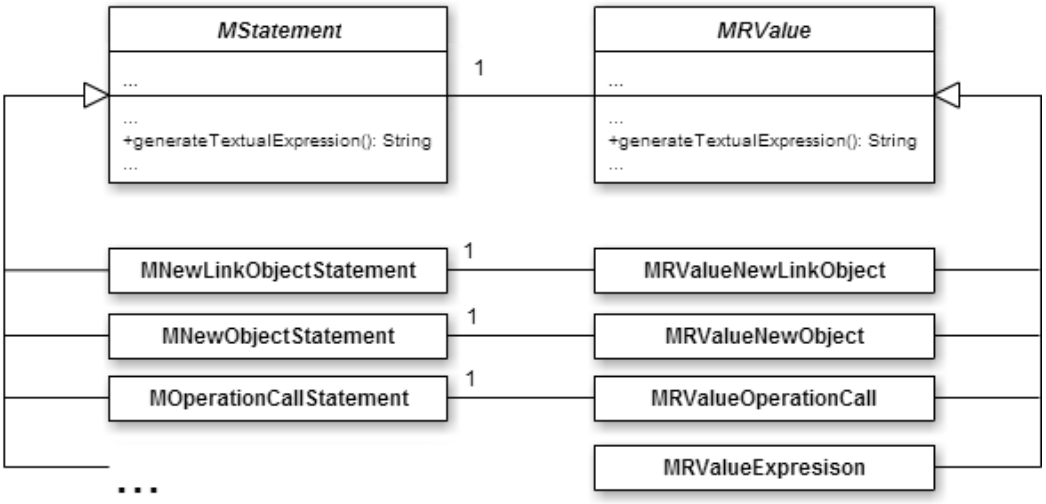
Una vez definidas las clases que afectan a las sentencias *SOIL* y a las expresiones *OCL*, se puede proceder a realizar las modificaciones y el nuevo diseño sobre el diagrama de clases de la librería, en caso de ser necesario.

En primer lugar, la función privada *generateTextualOperationTranslation* añadida a la clase *DocumentationGenerator,* es la principal encargada de empezar el proceso de traducción; al ejecutarse (desde *generateOperationsOfClass),* se inicia el proceso de interacción entre la documentación y las clases de la librería.

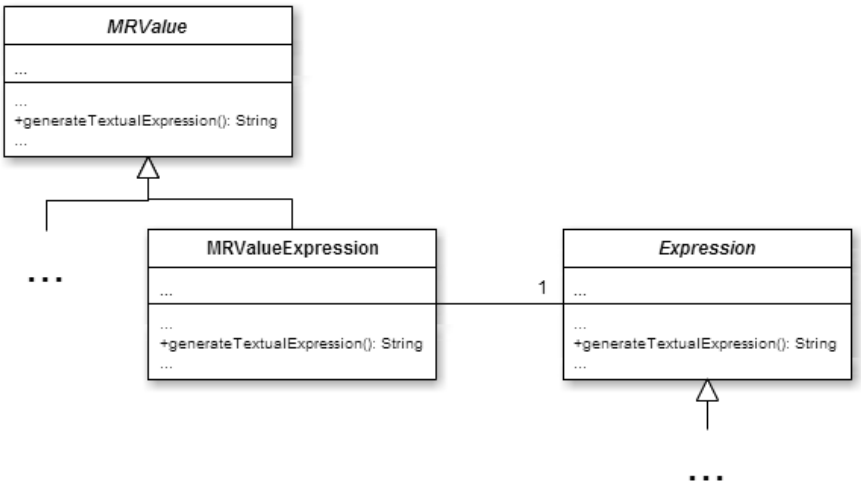
Dado que todas las sentencias *SOIL* de la librería, son hijos de la superclase abstracta *MStatement,* se ha decidido añadir sobre esta clase, una operación pública llamada *generateTextualExpression* y así de esta manera, todas las subclases de *MStatement*, heredarán la operación y tendrán su funcionamiento propio o por defecto de la superclase, dependiendo del tipo de sentencia que se trate y si se traduce o no.

Examinando detenidamente las clases de las sentencias *SOIL,* se puede observar, que para representar la expresión, algunas dependen del tipo abstracto *MRValue*, concretamente de *MRValueNewObject, MRValueNewLinkObject* y *MRValueOperationCall,* que a su vez, incluyen un atributo del tipo *MStatement*.

Por este motivo, también se debe añadir la operación *generateTextualExpression* en la superclase *MRValue*, para que todas las subclases hagan su traducción correspondiente y una vez realizado todo esto, habría cobertura sobre todas las clases mencionadas en la Tabla 2.



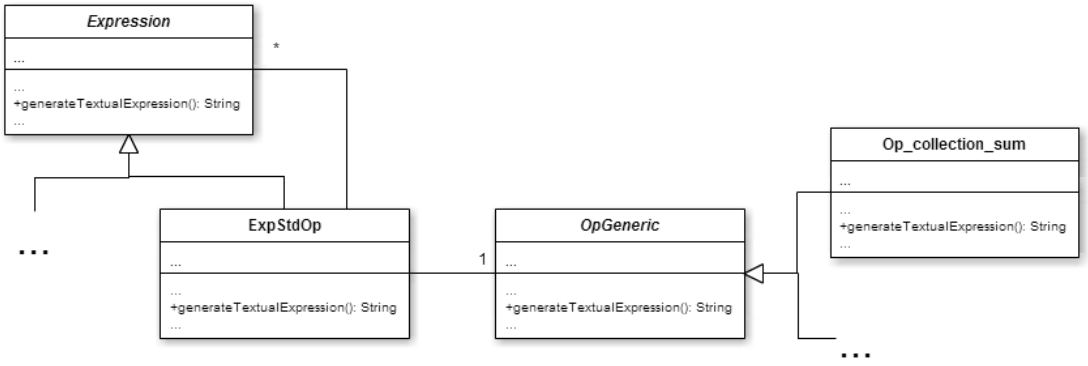
En cuanto a las modificaciones necesarias para las expresiones *OCL,* parten de *MRValueExpression,* la única subclase de *MRValue* que no contiene un atributo del tipo *MStatement,* sino del tipo abstracto *Expression.*



Como la clase abstracta *Expression,* engloba todas las expresiones *OCL,* la solución a la traducción de *OCL* es análoga a la realizada con *MStatement*, teniendo que añadir la operación *generateTextualExpression* en la superclase y heredándose en sus subclases con un comportamiento determinado. Cabe destacar que algunos tipos de expresiones, están formados por otros tipos de expresiones más pequeñas y que con esta solución, tan solo haría falta hacer una llamada a la misma operación.

Ahora ya se tiene cobertura para todas las expresiones *OCL,* pero en relación a la Tabla 3, esto tan solo es un subconjunto de expresiones (el nombre de clase empieza por '*Exp*'); el motivo de esto, consiste en que la librería tiene el otro subconjunto, representado como operaciones estándar, mediante la jerarquía de la clase abstracta *OpGeneric,* a partir de un atributo de la clase de expresión *ExpStdOp*.

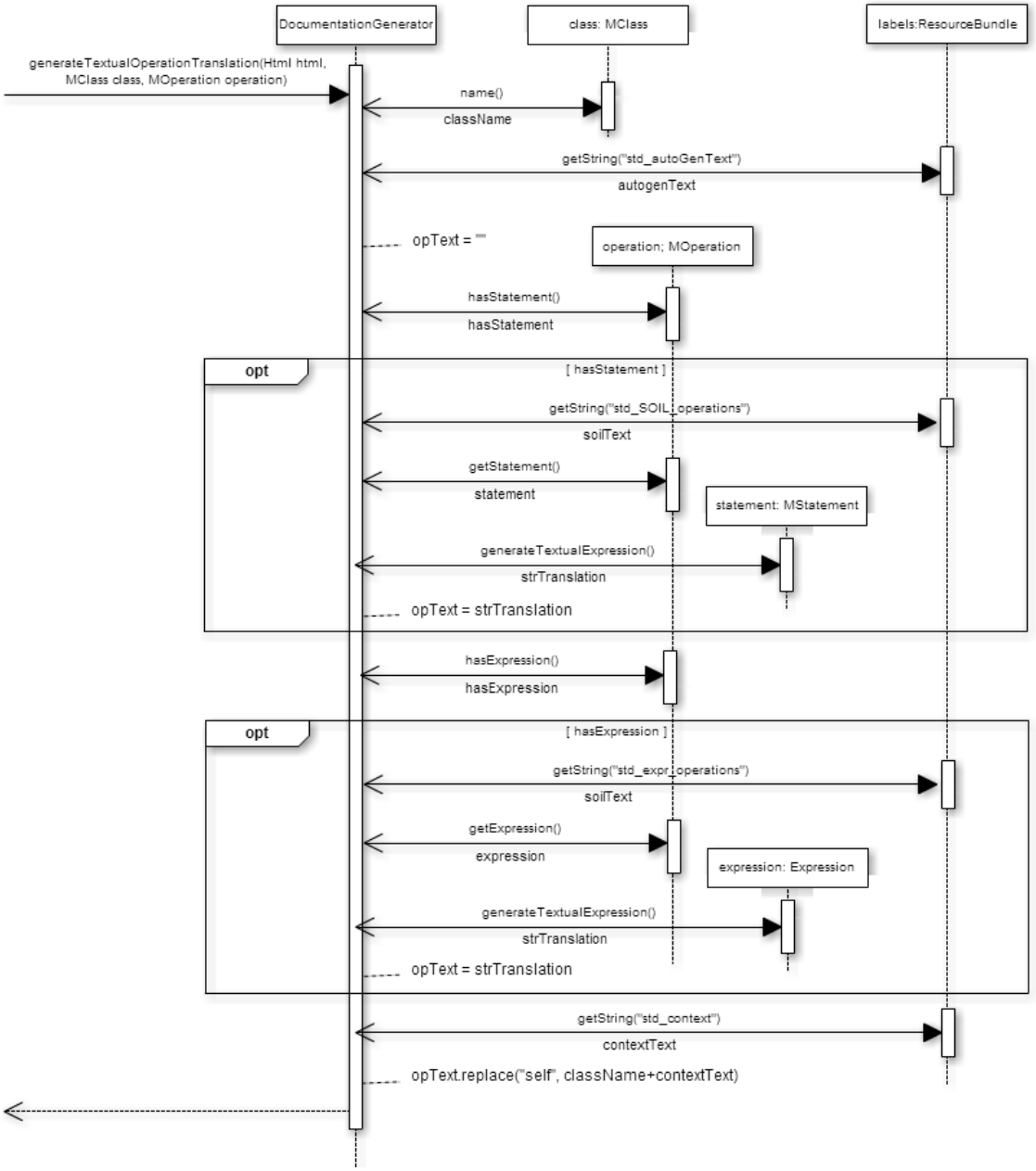
Una vez más, para abarcar todo el abanico de posibilidades de operaciones estándar, se añade sobre la clase abstracta *OpGeneric y* a la traducción del atributo del mismo tipo sobre *ExpStdOp* la llamada a la función *generateTextualExpression,* teniendo cobertura para cada una de las operaciones estándar.



Diagramas de secuencia

GenerateTextualOperationTranslation

Esta es la función principal de la traducción *OCL,* incluida en la clase de la documentación. A partir de una operación de una clase (*MOperation),* se comprueba si se trata de un tipo *statement* o *expression* y se llama a método *generateTextualExpresion* de la clase *M*Statement o *Expression* según corresponda. En ese momento y mediante el polimorfismo, se inicia el proceso de traducción navegando por la jerarquía de clases, que forman la operación a partir de llamadas sucesivas al método *generateTextualDescription (*para ver los diagramas de cómo se genera la traducción de cada expresión ir al Anexo C), que van procesando desde las unidades textuales de mayor tamaño, hasta llegar a unidades de texto simple e indivisible. Finalmente, al acabar de procesar la operación, se devuelve el texto traducido listo para añadir a la documentación.



5.4.3 Generación de diagramas de actividad y de secuencia

Para la generación automática de diagramas de actividad y secuencia, se va a utilizar la librería *jsUML2,* que nos permite hacer las representaciones gráficas deseadas para esta funcionalidad. Esta librería, cuenta con una documentación[22] amplia de cómo debe utilizarse que se ha estudiado a fondo, pero tan solo se puede usar *JavaScript* para mostrar los diagramas.

Debido a que las tecnologías utilizadas por *Recover* y la librería son distintas, se debe realizar un metamodelo a partir del estudio de la documentación existente; la función de este metamodelo consiste en crear una capa de abstracción sobre la librería, que permita trabajar con sus elementos y representarlos en *Recover,* para crear un algoritmo de generación automática, manipularlos con facilidad y finalmente representar el diagrama final con los objetos del metamodelo traducidos a *jsUML2*.

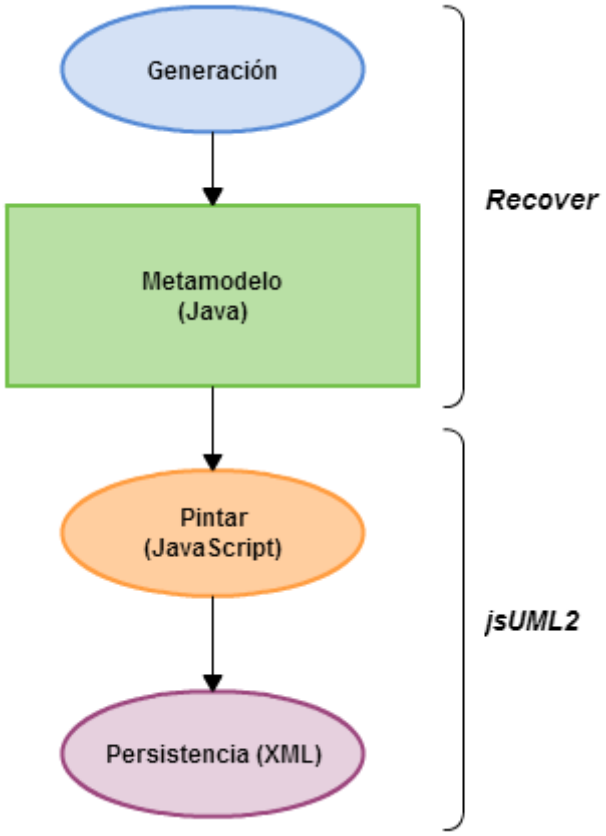


Figura 33.- Diagrama del metamodelo sobre jsUML2

En la Figura 33, se muestra la estructura de funcionamiento propuesta y el papel del metamodelo sobre esta, de tal manera que cada una de las fases tiene una función y se ejecutan secuencialmente para cada uno de los diagramas. La descripción concreta de cada una de las fases es la siguiente:

- **Generación:** Se extrae la información necesaria de *Recover,* para generar el tipo concreto de diagrama.

- **Metamodelo:** La información de la fase anterior, se mapea con su clase correspondiente en el metamodelo, teniendo así una representación de los elementos del diagrama.
- **Pintar:** En esta fase, se procede a representar el diagrama en la documentación web; para ello, el metamodelo cuenta con una forma de sacar toda la información, de manera que la librería *jsUML2* pueda interpretarlo y pintarlo en esta fase.
- **Persistencia:** Una vez el resultado de las fases anteriores ha sido satisfactorio, en esta última se utiliza la funcionalidad de la propia librería *jsUML2,* para guardar sobre *Recover* el resultado obtenido de los diagramas.

5.4.3.1 Generación de diagramas de actividad

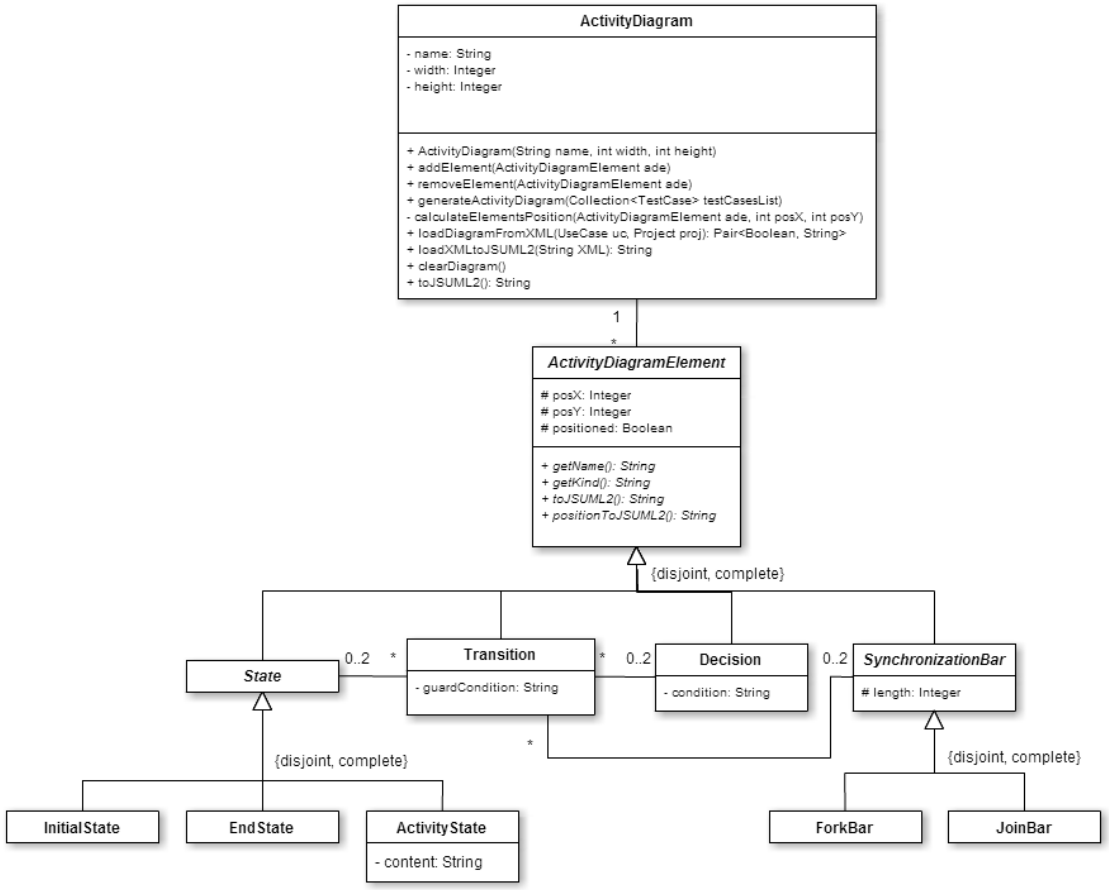
Diagramas de clases

En la Tabla 4, se muestra la relación entre las clases del metamodelo y los elementos de la librería *jsUML2,* que se necesitan para representar un diagrama de actividad.

jsUML2	Clase metamodelo
UMLActivityDiagram	ActivityDiagram
UMLInitialNode	InitialState
UMLActivityFinal	EndState
UMLAction	ActivityState
UMLFork_JoinNode	ForkBar / JoinBar
UMLFlow	Transition
UMLDecision_MergeNode	Decision

Tabla 4.- Relación jsUML2-metamodelo para los diagramas de actividad

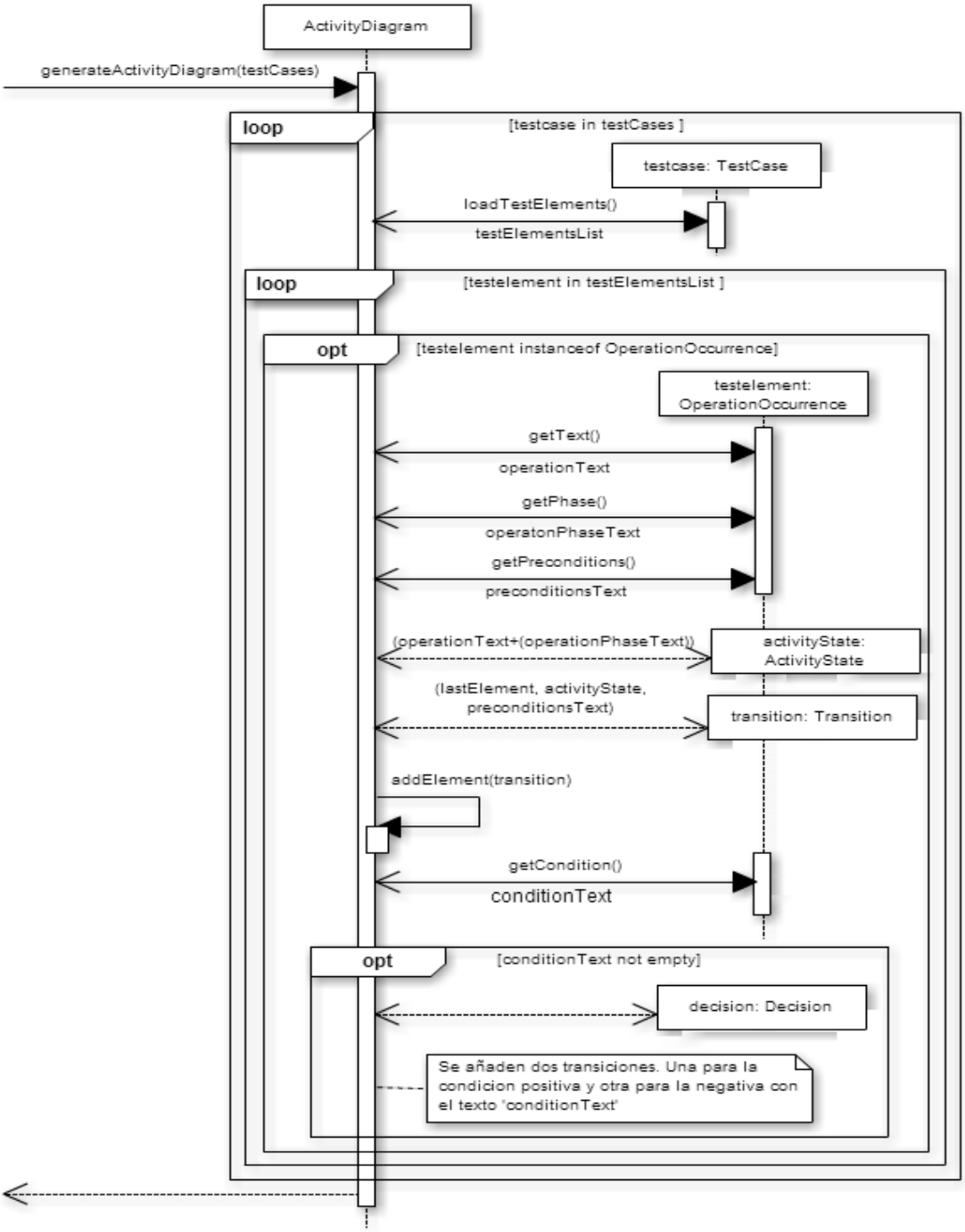
A continuación, se ha diseñado la siguiente jerarquía de clases sobre los elementos de la tabla anterior para el paquete de diagramas de actividad.



Diagramas de secuencia

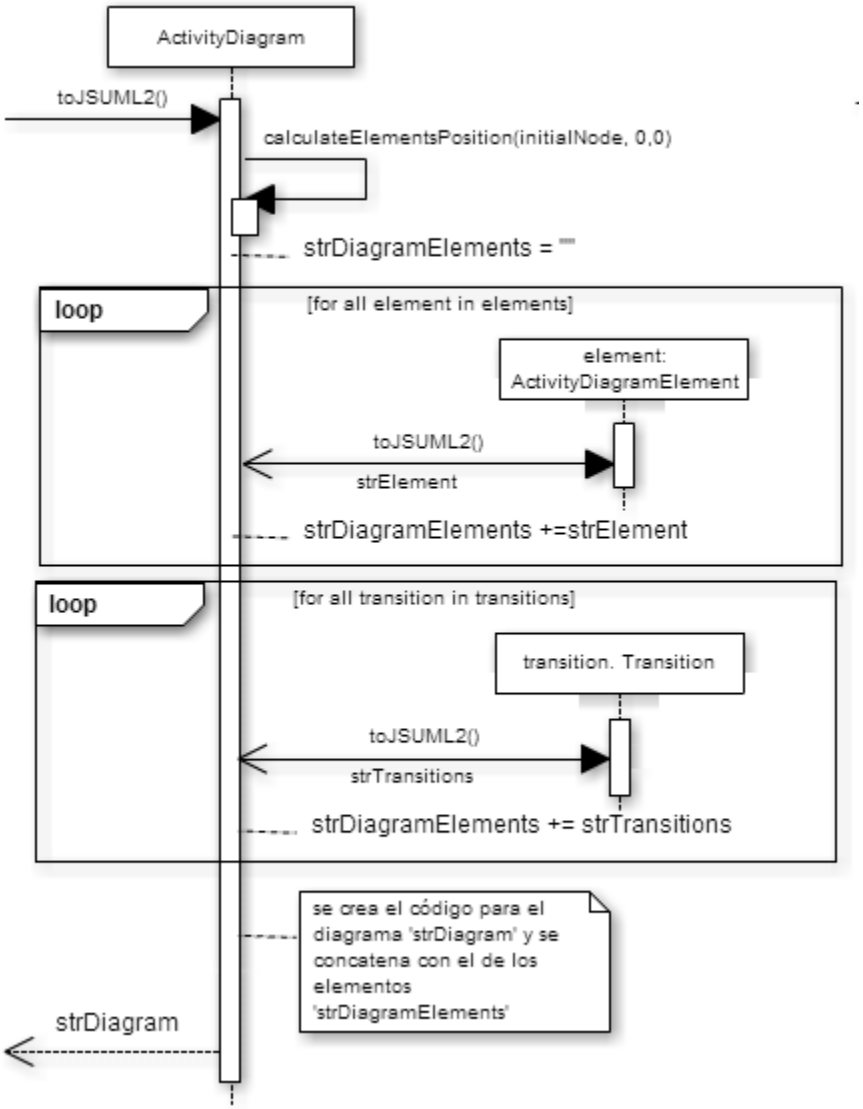
GenerateActivityDiagram

Esta función, recoge toda la información relativa a los casos de prueba de la herramienta *Recover* y la trata para convertirla en elementos del metamodelo, representando así el diagrama de actividad completo internamente.



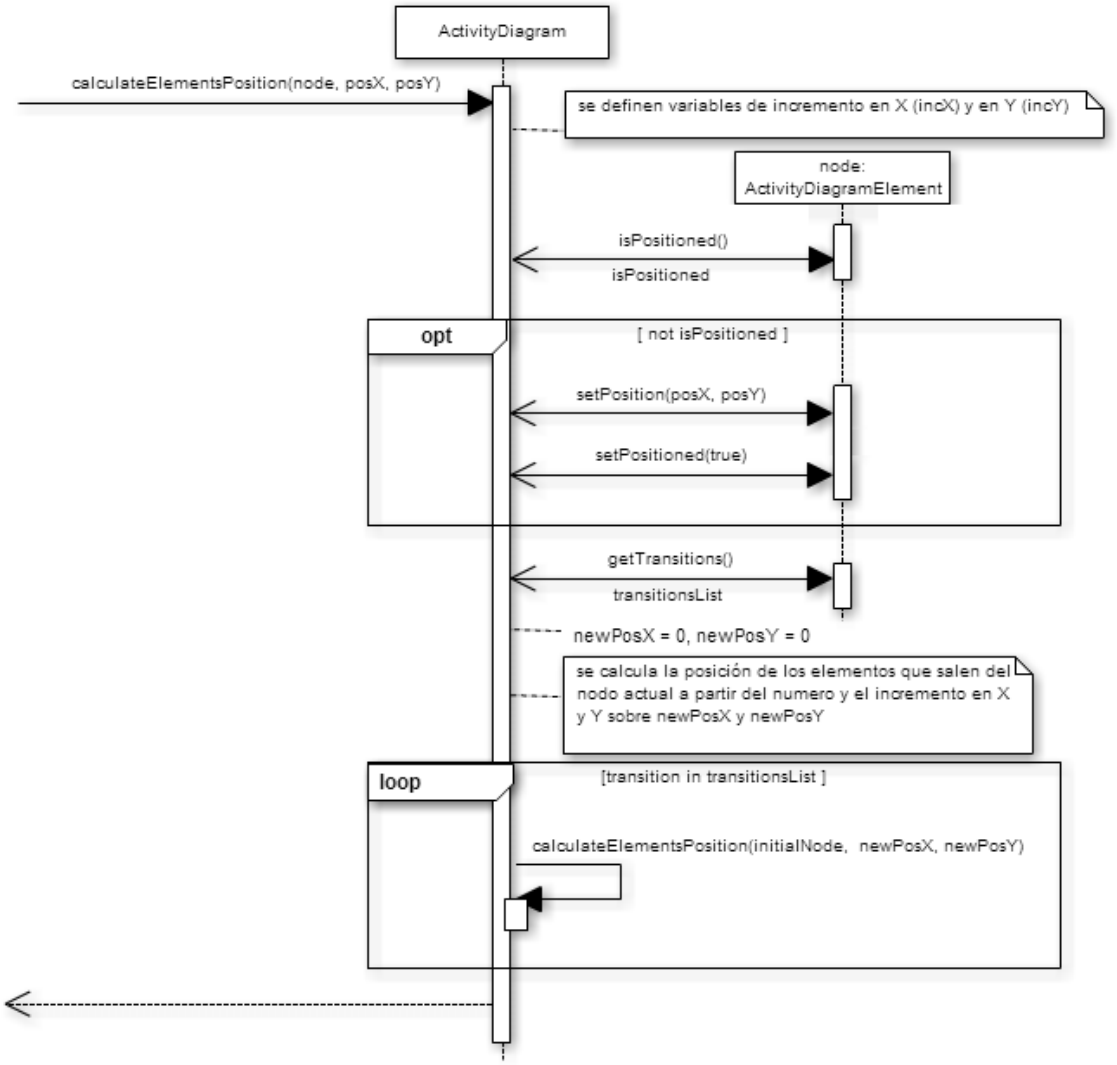
ToJSUML2

Mediante esta función, se transforma calcula la posición de todos los elementos del metamodelo (*calculateElementsPosition)*, dejándolo bien formado para la representación gráfica; seguidamente, recorriendo todos los elementos y transiciones, utilizando su función *toJSUML2* concreta, se forma el código JavaScript completo sobre la variable *strDiagram*, listo para mostrar en la documentación.



CalculateElementsPosition

Calcula automáticamente la posición de cada elemento del diagrama de actividad recursivamente, situando uno a uno y extendiéndose por cada transición de este.



5.4.3.2 Generación de diagramas de secuencia

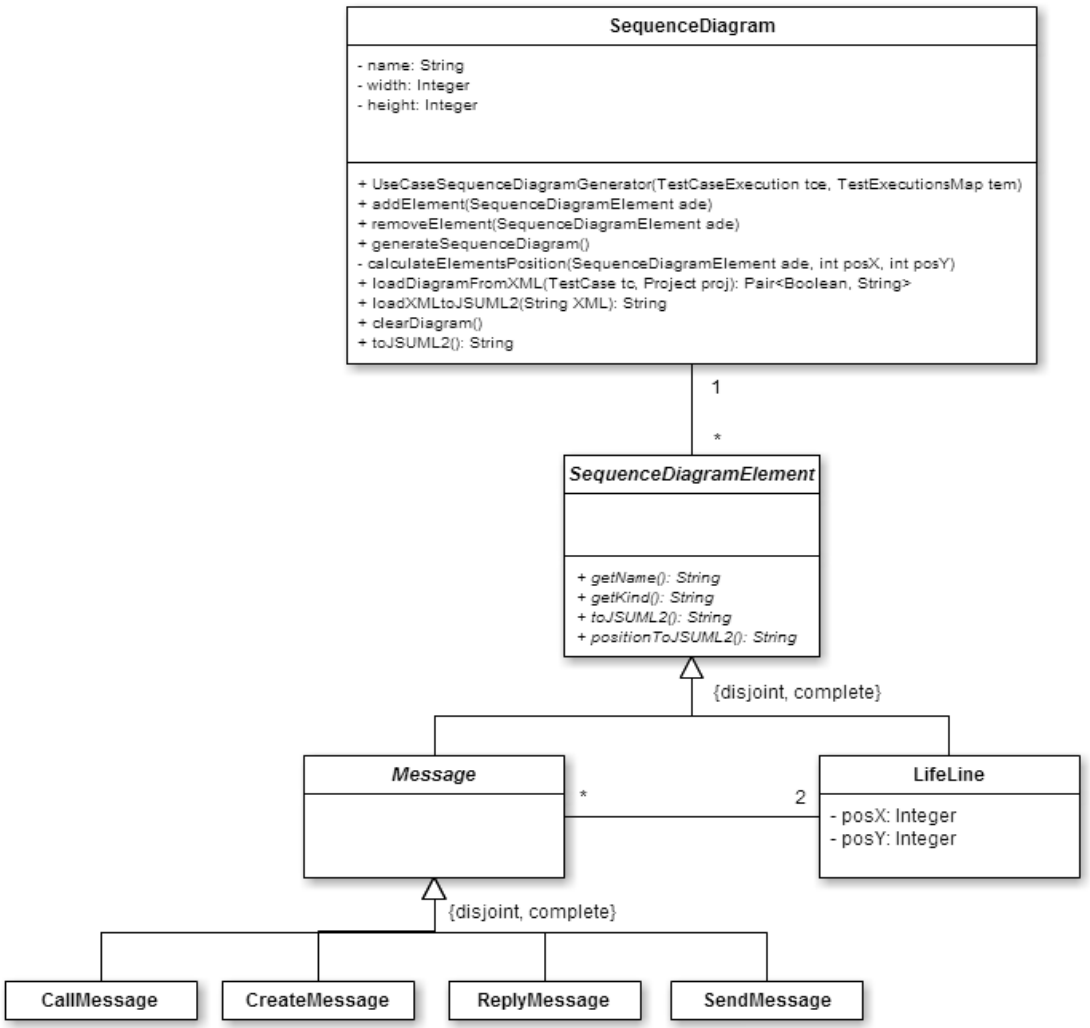
Diagramas de clases

En la Tabla 5, se muestra la relación entre las clases del metamodelo y los elementos de la librería *jsUML2,* que se necesitan para representar un diagrama de secuencia.

jsUML2	Clase metamodelo
UMLSequenceDiagram	SequenceDiagram
UMLLifeLine	Lifeline
UMLCallMessage	CallMessage
UMLCreate	UMLCreateMessage
UMLReplyMessage	ReplyMessage
UMLSendMessage	SendMessage

Tabla 5.- Relación jsUML2-metamodelo para los diagramas de secuencia

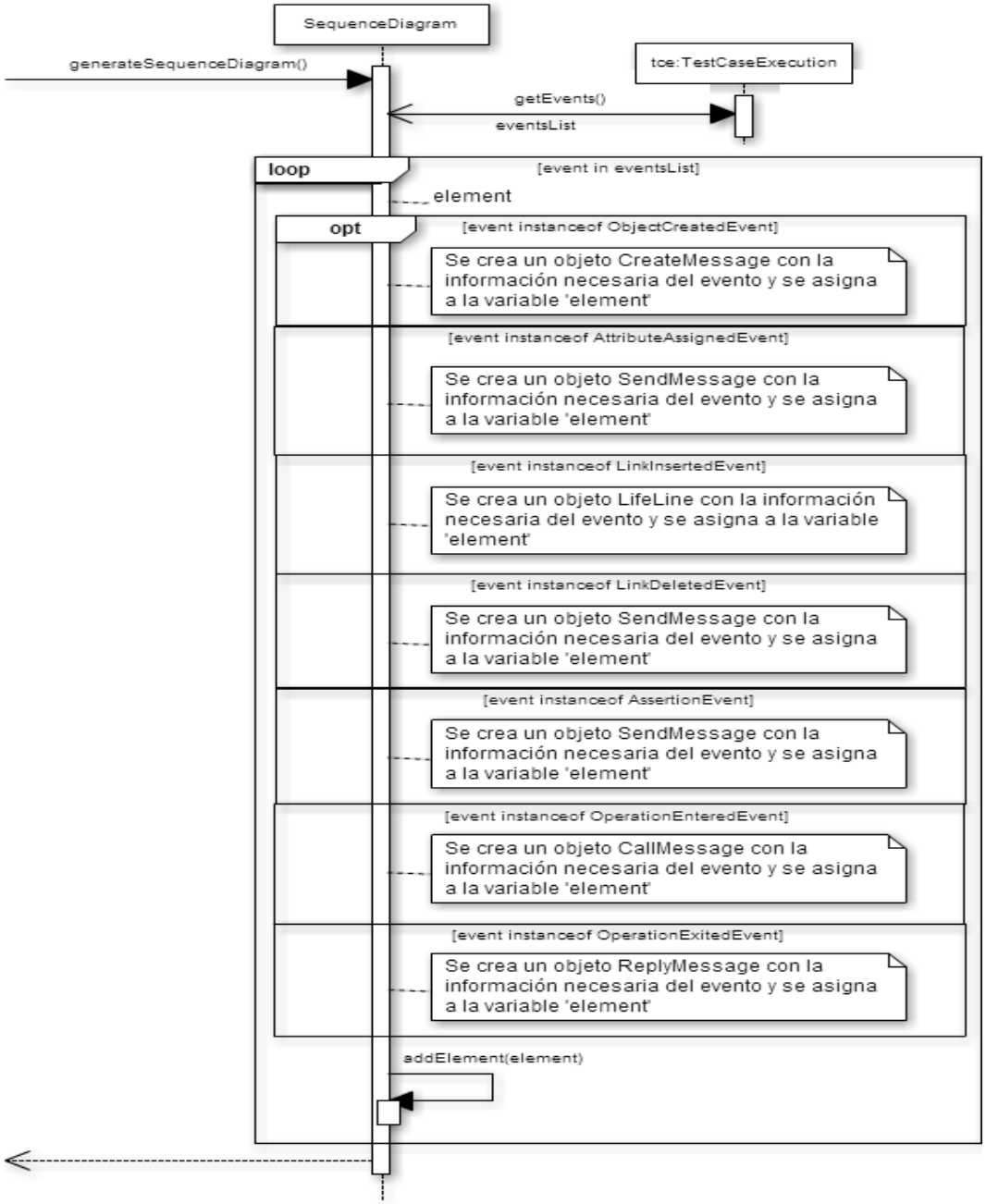
A partir de la tabla anterior, se ha diseñado como metamodelo la siguiente jerarquía de clases para el paquete de diagramas de secuencia.



Diagramas de secuencia

La generación de diagramas de secuencia, se procesa de manera igual que los diagramas de actividad, por este motivo, las funciones de generación de código JavaScript (*toJSUML2)* y la de cálculo de posición de los elementos (*calculateElementsPosition)*, funcionan de manera similar; la única diferencia notable en estas funciones, tiene que ver con la segunda, que realiza incrementos de cálculo distinto ya que los elementos y el diagrama son diferentes; sin embargo, la función *generateSequenceDiagram*, sí que cambia al utilizar otros datos de *Recover* (eventos de la validación pruebas-modelo), para la representación del diagrama en el metamodelo.

GenerateSequenceDiagram



6 IMPLEMENTACIÓN

La etapa de implementación, es en la que se aplican los requisitos definidos en la especificación y el diseño para la obtención de las funcionalidades establecidas como objetivo. En este apartado se detalla el resultado y proceso de implementación de las funcionalidades, así como los detalles técnicos de la misma.

6.1 DOCUMENTACIÓN WEB AUTOGENERADA

En esta sección, se explica la parte principal y más importante del proyecto, la documentación web autogenerada (para ver las imágenes completas de las páginas implementadas ir al Anexo A). Para la implementación se han utilizado diversas tecnologías, en concreto para crear el apartado de la Figura 34, se ha utilizado *JSP*, ya que es la tecnología en que esta implementado *Recover*. La página de 'Documentación' ya existía previamente en la herramienta, debido a que contenía otras funcionalidades, así que solo se debía añadir la sección de documentación autogenerada a la página *JSP* correspondiente mediante *HTML* y *CSS.*

Figura 34.- Página de documentación en Recover

Una vez implementada la sección de autogeneración, desde la cual empezará la creación automática de la documentación web, se necesita conectar al servidor para que se inicie proceso; para ello, mediante una llamada *AJAX,* al presionar el botón "Generar especificación HTML", el servidor comienza a trabajar en la generación y una vez finalizada correctamente, se muestra el mensaje de

Figura *35*, donde podrá descargarse la documentación en formato *'zip'.*

Figura 35.- Mensaje de generación HTML correcta

Como se ha podido ver en el apartado de diseño, la documentación se forma a partir de las llamadas a funciones de la clase *DocumentationGenerator,* las cuales contienen cada una su código *HTML* correspondiente. Debido a que la capa de negocio de *Recover,* está implementada en Java y generar código *HTML* a partir de cadenas de caracteres puede ser más complicado y difícil de mantener, se ha recurrido a utilizar la librería Java Anti-Template Language (JATL) en la mayoría de los casos donde es posible; de esta manera, es posible programar las páginas de la documentación en lenguaje Java directamente y la librería se encarga de mapear cada función con su *tag HTML* así como generar el código interpretable por el navegador.

Figura 36.- Página principal de la documentación

En la Figura 36, se muestra el resultado obtenido en la implementación de la página principal de la documentación con la tecnología mencionada y en la Figura 37, la implementación en *JATL* de la tabla de contenidos.

```
private void generateTableOfContents(Html html) {
    html
        .h2().classAttr("section").text(labels.getString("DOC_TABLE_OF_CONTENTS")).end()
        .ul().classAttr("tableofcontents")
            .li().a().href("indexofclasses.html").text(labels.getString("DOC_INDEX_CLASSES")).end().end()
            .li().a().href("indexofenumerations.html").text(labels.getString("DOC_INDEX_ENUMERATIONS")).end().end()
            .li().a().href("indexofoperations.html").text(labels.getString("DOC_INDEX_OPERATIONS")).end().end()
            .li().a().href("indexofusecases.html").text(labels.getString("DOC_INDEX_USECASES")).end().end()
        .end()
    .end();
}
```

Figura 37.- Código en JATL de la tabla de contenidos

Accediendo a un enlace de la tabla de contenidos, aparece el índice de la sección seleccionada y al seleccionar el enlace con la opción "Clases", la documentación nos lleva a la página de la Figura 38, cuyo contenido es el índice de clases (*indexofclasses.html*).

Figura 38.- Página del índice de clases

Esta página, contiene un encabezado simple y un listado de clases que se crea a partir de la iteración sobre las clases del modelo en *Recover,* que a su vez generan su página específica de clase (ver Figura 39 y Figura 40).

```
private void generateIndexOfClasses() throws IOException, SQLException, NamingException {
    File f = new File(mainFullPath+"./documentation/indexofclasses.html");
    FileWriter fw = new FileWriter(f);
    Html html = new Html(fw);
        generateSimpleHeader(html);
        html.h2().classAttr("section").text(labels.getString("DOC_PAGECLASSES_INDEX_TITLE")).end()
            .ul().classAttr("tableofcontents");
            for (MClass cls : usemodel.model().classes()){
                html.li().a().href("op_"+cls.name()+".html").text(cls.name()).end().end();
                generateClassPage(cls);
            }
            html.end()
        .end();
    fw.close();
    this.addToZIPFile(f);
}
```

Figura 39.- Código del índice de clases

Documentación de Sistema

Información de proyecto

Título de Proyecto: osCommerceDemoCastellano

Fecha/hora: 2015/12/21 17:27:26

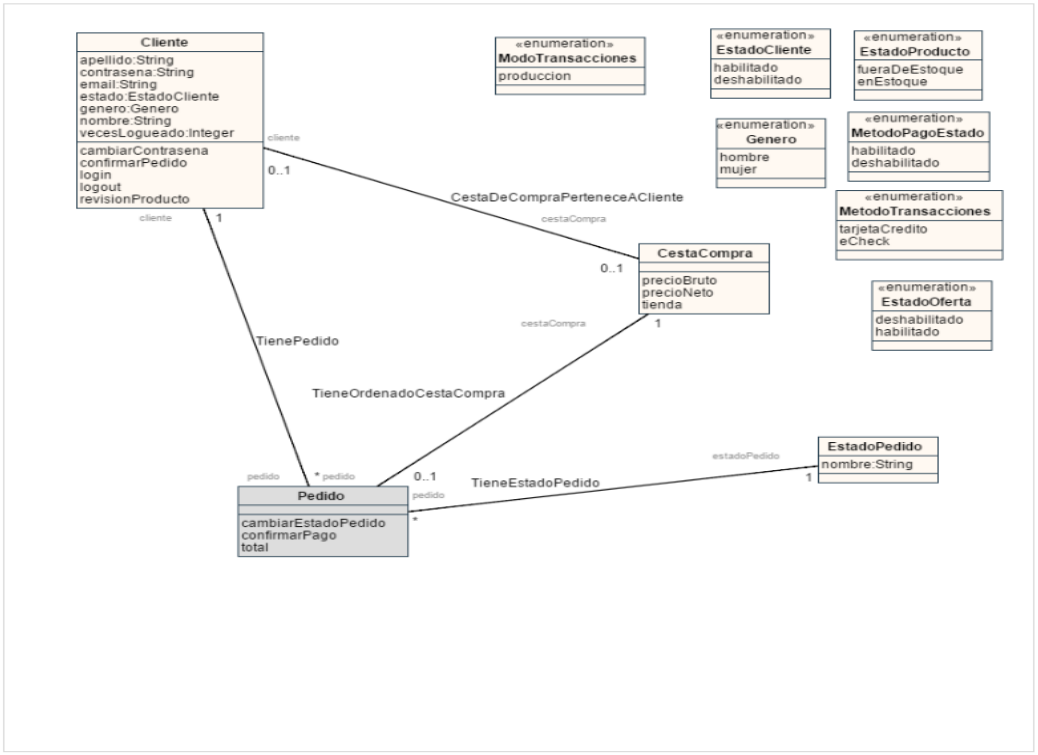
< Volver atrás > Clases > Enumeraciones > Operaciones > Casos de Uso

Especificación de Clase

Nombre de la clase:

Pedido

Start autolayout Stop autolayout



Asociaciones

TieneOrdenadoCestaCompra

pedido[0..1]	Pedido
cestaCompra[1]	CestaCompra

TieneEstadoPedido

pedido[*]	Conjunto(Pedido)
estadoPedido[1]	EstadoPedido

TienePedido

cliente[1]	Cliente
pedido[*]	Conjunto(Pedido)

Operaciones

cambiarEstadoPedido

Parámetros

estado	EstadoPedido

Método de evento

```
delete (self,self.estadoPedido) from TieneEstadoPedido;
   insert (self,estado) into TieneEstadoPedido;
```

confirmarPago

Método de evento

```
delete (self,self.estadoPedido) from TieneEstadoPedido;
   insert (self,self.cliente.tienda.estadoPagado) into TieneEstadoPedido;
```

Figura 40.- Página concreta de una clase

6.2 TRADUCCIÓN DE *UML* Y *OCL* A LENGUAJE NATURAL

La traducción a lenguaje natural es una de las partes más difíciles del proyecto, dado que puede haber numerosas expresiones y de complejidad muy elevada, que pueden resultar en una traducción poco precisa. En esta sección, se explica la solución que combinada con la librería *USE* ha dado resultados positivos para varios proyectos en *Recover*.

La implementación de la traducción se basa en la substitución de partes de cadenas de caracteres genéricas mediante el método *format* de la clase *String* de Java, de esta manera se pueden personalizar las cadenas con diferente contenido, pero siempre utilizando la misma.

Como esta traducción debe ser tanto en inglés como castellano, se ha hecho uso de dos ficheros con extensión '*.properties*', que se utilizan normalmente como almacenamiento clave-valor de las cadenas de caracteres que pueden variar en una aplicación según el idioma.

Utilizando este método de traducción, las cadenas se formatean en código Java de la siguiente forma:

String.format(getString("key"), "param1", "param2", …, "paramN")

De esta manera se substituyen los caracteres *"%s"* de la cadena como valor de la clave por la cantidad de parámetros de la función *format*.

Tomamos como ejemplo básico la primera parte de la Figura 41, que describe las asociaciones y multiplicidades de la clase *Pedido;* esta parte, se genera a partir de la función *generateTextualMultiplicityDescription* de la clase *DocumentationGenerator* y las cadenas de caracteres empleadas se desglosan de la siguiente manera:

- "Un/a **Pedido** debe tiene las siguientes características"

```
std_classGeneralSentence=Un/a <b> %s </b> tiene las siguientes características
std_classGeneralSentence=A/an <b> %s </b> has the following characteristics
```

- "Un/a **Pedido** debe estar asociado (*TieneOrdenadoCestaCompra*) a un/a CestaCompra"

```
std_classAssociationSentence=Un/a <b> %s </b> %s <i>( %s)</i> %s %s
std_classAssociationSentence=A/An <b> %s </b> %s <i>( %s)</i> %s %s
```

  - Dependiendo de la multiplicidad se utilizan las siguientes cadenas:

```
std_multiplicity_may=puede estar asociado
std_multiplicity_must=debe estar asociado
std_multiplicity_MANY =a varios
std_multiplicity_ONE_MANY =como mínimo a un/a
std_multiplicity_ONE=a un/a
std_multiplicity_ZERO_ONE=como máximo a un/a
```

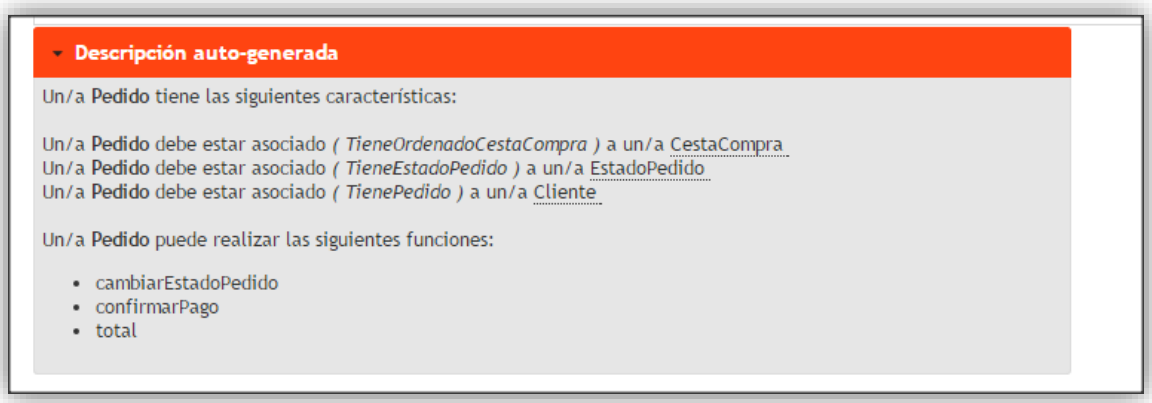


Figura 41.- Resultado de una traducción UML

En cuanto a la traducción de código *OCL* de las operaciones, el método empleado es el mismo que para el *UML*, pero en este caso no se trata de cadenas simples, sino que al haber expresiones dentro de expresiones, la traducción tiene que conservar una lógica, basándose en el árbol sintáctico de la librería *USE* y reordenando cada parte de la expresión si fuese necesario (Figura 42).

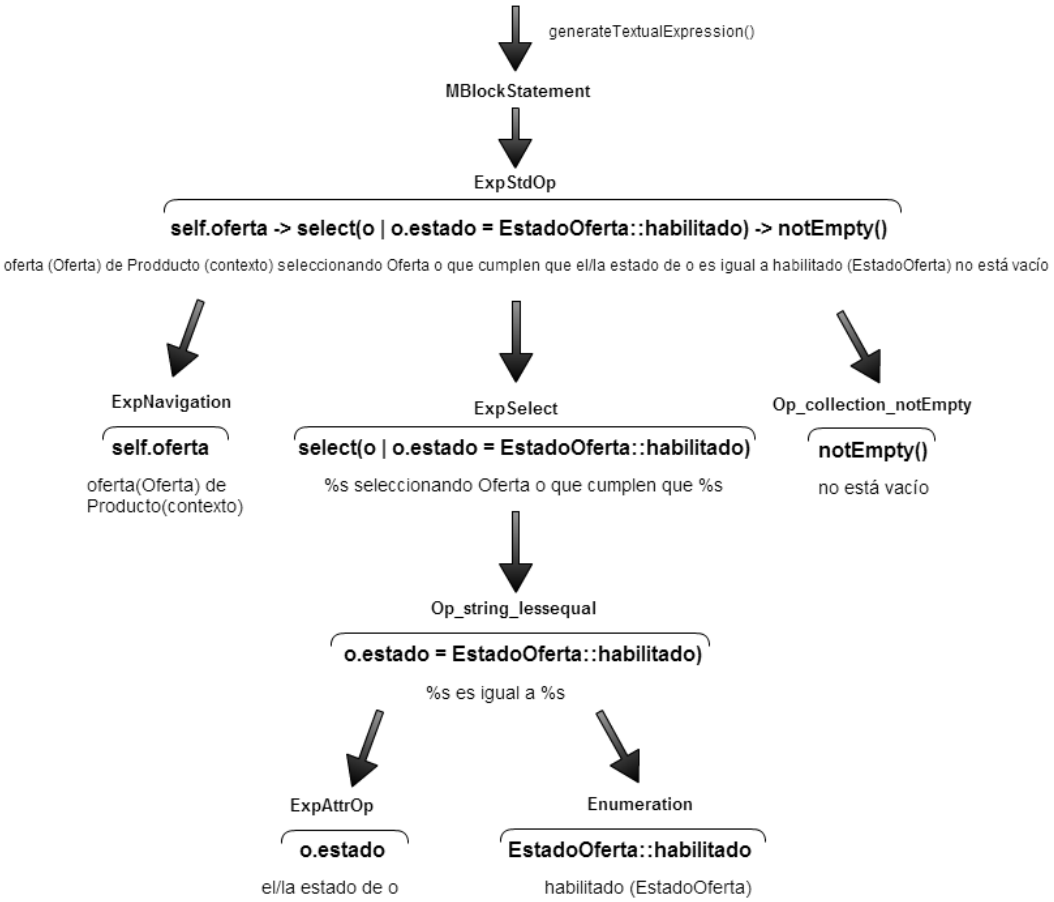


Figura 42.- Procedimiento de traducción de una expresión OCL compleja

Cada una de las expresiones *OCL,* mencionadas en la fase de diseño (ver Tabla 2 y Tabla 3), tiene su propia traducción, deducida a partir de las situaciones en que aparece y sobre qué tipo de objetos se invoca.

A continuación en la Figura 43, se muestran resultados obtenidos de la traducción de una sentencia *SOIL.*

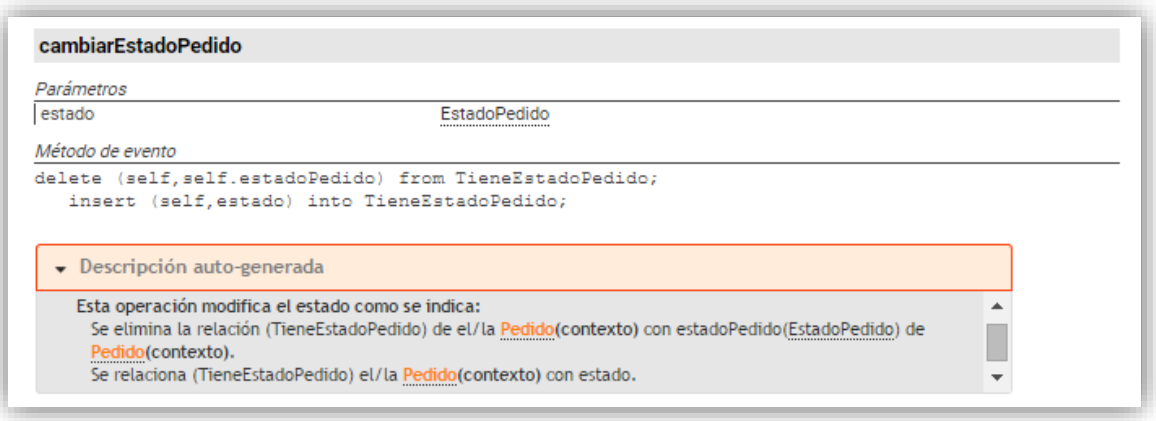


Figura 43.- Resultado de una traducción OCL

6.3 GENERACIÓN DE DIAGRAMAS DE ACTIVIDAD Y DE SECUENCIA

El primer paso para la generación de diagramas, ha sido relacionar cada una de las clases Java del metamodelo, con el correspondiente elemento de la librería *jsUML2* a partir del estudio de su documentación, teniendo así para cada uno el código JavaScript que lo muestra en un diagrama (ver Tabla 6 y Tabla 7).

Clase Java	JavaScript (*jsUML2*)
ActivityDiagram	UMLActivityDiagram({ id:'x', width:*'number'*, height:*'number'* })
InitialState	UMLInitialNode({ x:*'number'*, y:*'number'* })
EndState	UMLActivityFinal({ x:*'number'*, y:*'number'* })
ActivityState	UMLAction({ x:*'number'*, y:*'number'* })
ForkBar/JoinBar	UMLFork_JoinNode({ x:*'number'*, y:*'number'* })
Transition	UMLFlow({ a:*'element'*, b:*'element'* })
Decision	UMLDecision_MergeNode({ x:*'number'*, y:*'number'* })

Tabla 6.- Relación del metamodelo para diagramas de actividad

Clase Java	JavaScript (*jsUML2*)
SequenceDiagram	UMLSequenceDiagram({ id:'x', width:*'number'*, height:*'number'* })
LifeLine	UMLLifeline({ x:*'number'*, y:*'number'* })
CallMessage	UMLCallMessage({ a:*'element'*, b:*'element'*, y:*'number'* })
CreateMessage	UMLCreate({ a:*'element'*, b:*'element'*, y:*'number'* })
ReplyMessage	UMLReplyMessage({ a:*'element'*, b:*'element'*, y:*'number'* })
SendMessage	UMLSendMessage({ a:*'element'*, b:*'element'*, y:*'number'* })

Tabla 7.- Relación del metamodelo para diagramas de secuencia

Una vez estudiados los constructores de *jsUML2* para cada elemento de los diagramas y sus correspondientes parámetros, ya podemos implementar para cada uno su función del metamodelo *toJSUML2.*

```
public String toJSUML2(){
    String jsName = this.getName();
    String s = "var " + jsName + " = new UMLAction({ x:" +  this.posX + ", y:" + this.posY + "});\n" +
               + jsName + ".setName(\"" + this.content + "\");\n" +
               + jsName + ".notifyChange();\n";
    return s;
}
```

Figura 44.- Código jsUML2 para un ActivityState

Esta función, devuelve una cadena de caracteres de la representación en JavaScript de un elemento del metamodelo con su posición correspondiente y otra información asociada (en el caso de la Figura 44, se muestra la función para una *UMLAction*); sin embargo, para obtener estos datos, debemos tener en primer lugar una fuente de datos para alimentar el diagrama y en segundo lugar, un algoritmo que asigne una posición a los elementos del diagrama, para mostrarlo correctamente al usuario en la documentación.

La fuente de datos necesaria para generar un diagrama de actividad, consiste en el conjunto de casos de prueba de un caso de uso; estos, existen en *Recover* porque han sido introducidos por los usuarios de la herramienta y se transforman en elementos de del metamodelo mediante la función *generateActivityDiagram* de la clase ActivityDiagram del metamodelo, la cual recibe como parámetro el conjunto de casos de prueba.

Una vez se tiene el diagrama representado internamente en el metamodelo Java, debemos ejecutar el algoritmo de posicionamiento (ver Figura 46) y pasarlo al formato válido de *jsUML2* para pintarlo en el *canvas* de la documentación web*;* de esto, se encarga la función *toJSUML2* de la clase *ActivityDiagram* (ver Figura 45).

```
public String toJSUML2(){

        // Calculate position of diagram elements and width/height of activity diagram
        calculateElementsPosition(initial, Math.abs(maxWidth-minWidth), Math.abs(maxHeight-minHeight));

        // Setting final width/height of the diagram
        this.WIDTH  = Math.abs(maxWidth)  + Math.abs(minWidth);
        this.HEIGHT = Math.abs(maxHeight) + Math.abs(minHeight);

        // Drawing diagram
        StringBuilder diagram = new StringBuilder();
        diagram.append("var activityDiagram = new UMLActivityDiagram({id: '" +
                    + name + "', width:" + WIDTH + ", height:" + HEIGHT + "});\n"
        + "var maxWidth="+ WIDTH +";\n"
        + "var maxHeight="+ HEIGHT +";\n"
        + "activityDiagram._width=maxWidth;\n"
        + "activityDiagram._mainContext.canvas.width = maxWidth;\n"
        + "activityDiagram._motionContext.canvas.width = maxWidth;\n"
        + "activityDiagram._div.style.width=maxWidth+\"px\";\n"
        + "activityDiagram._height=maxHeight;\n"
        + "activityDiagram._mainContext.canvas.height = maxHeight;\n"
        + "activityDiagram._motionContext.canvas.height = maxHeight;\n"
        + "activityDiagram._div.style.height=maxHeight+\"px\";\n"
        + "activityDiagram.notifyChange();\n"
        + "activityDiagram.draw();\n"
        + "activityDiagram.interaction( true );\n"
        + "activityDiagram.setUpdateHeightCanvas( true );\n");

        StringBuilder diagramElements = new StringBuilder();

        // Adding elements to the diagram
        for(ActivityDiagramElement ade: elements){
            diagramElements.append(ade.toJSUML2());
            diagramElements.append("activityDiagram" + ".addElement("+ ade.getName() +");\n");
        }

        // Creating and adding transitions between diagram elements
        for(Transition t: transitions){
            diagramElements.append(t.toJSUML2());
            diagramElements.append("activityDiagram" + ".addElement("+ t.getName() +");\n");
        }

        return diagram.toString() + diagramElements.toString();
}
```

Figura 45.- Función de transformación a jsUML2 para diagramas de actividad

```
private void calculateElementsPosition(ActivityDiagramElement ade, int posX, int posY){
    if(posX >= maxWidth)       maxWidth  = posX;
    else if(posX < minWidth)   minWidth  = posX;
    if(posY >= maxHeight)      maxHeight = posY;
    else if(posX < minHeight) minHeight = posX;

    final int incX = 300; // X axis increment between elements
    final int incY = 150; // Y axis increment between elements

    List<Transition> tr = ade.getOutgoingTransitions();

    if(!ade.isPositioned()){
        setElementPosition(ade, posX, posY);
        ade.setPositioned(true);
    }

    int newPosX;

    if(tr.size() <= 1)
        newPosX = ade.getPosX();
    else
        newPosX = (tr.size() % 2 == 0) ? posX - ((tr.size()/2) * incX) + (incX/2)
                                       : posX - ((tr.size()/2) * incX);

    for(Transition t : tr){
        if(!t.getEndPoint().isPositioned()){
            if(t.getInitialPoint().getKind().equals(InitialState.NAME) ||
               t.getEndPoint().getKind().equals(EndState.NAME))
                    calculateElementsPosition(t.getEndPoint(),0, ade.getPosY()+incY);
            else{
                if(t.getEndPoint().getKind().equals(Decision.NAME)) newPosX+=incX;
                if(t.getInitialPoint().getKind().equals(Decision.NAME)) newPosX-=incX;
                calculateElementsPosition(t.getEndPoint(), newPosX, ade.getPosY()+incY);
                if(t.getEndPoint().getKind().equals(Decision.NAME)) newPosX-=incX;
                if(t.getInitialPoint().getOutgoingTransitions().size() > 1) newPosX+=incX;
            }
        }
    }
}
```

Figura 46.- Algoritmo de posicionamiento de diagramas de actividad

Una vez ejecutadas estas funciones se obtiene el diagrama final. En la Figura 47, se muestra el resultado de esta implementación.

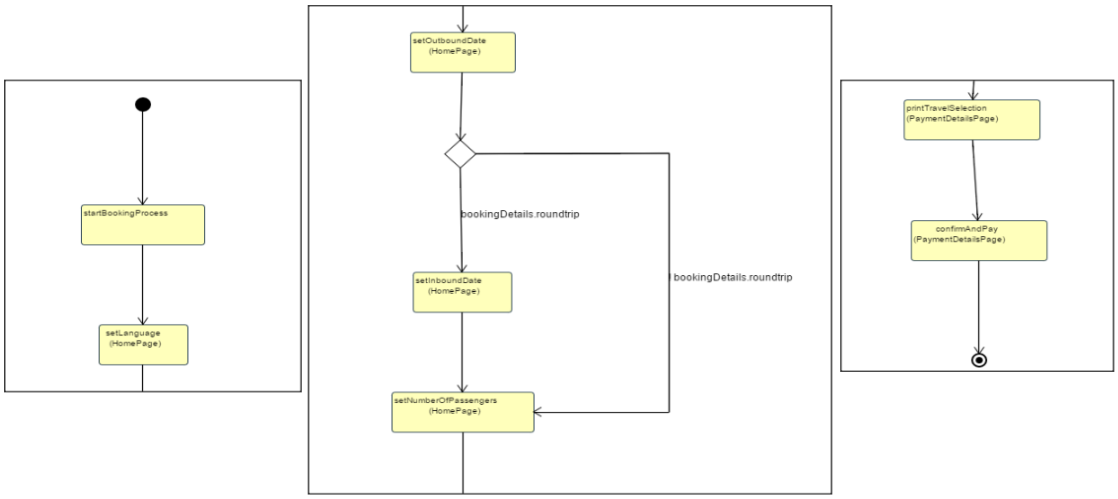


Figura 47.- Resultado final de la generación de diagramas de actividad (fragmentos)

En el caso de los diagramas de secuencia, el procedimiento seguido es el mismo que para los diagramas de actividad, con muy poca variación. Las principales diferencias son: que el algoritmo, utiliza de manera distinta el posicionamiento de los elementos y que la fuente de datos para la generación proviene de una clase llamada *TestCaseExecution*, que contiene una lista de eventos generados a partir de la ejecución de una validación sobre el sistema; aprovechar estos eventos, nos sirve para generar los elementos del metamodelo, ya que indican las acciones que se hacen sobre los diferentes objetos de un caso de prueba, el nombre y la clase de estos; en definitiva, tenemos todo lo que necesitamos para transformarlo a un diagrama de secuencia válido con resultados como el de la Figura 48.

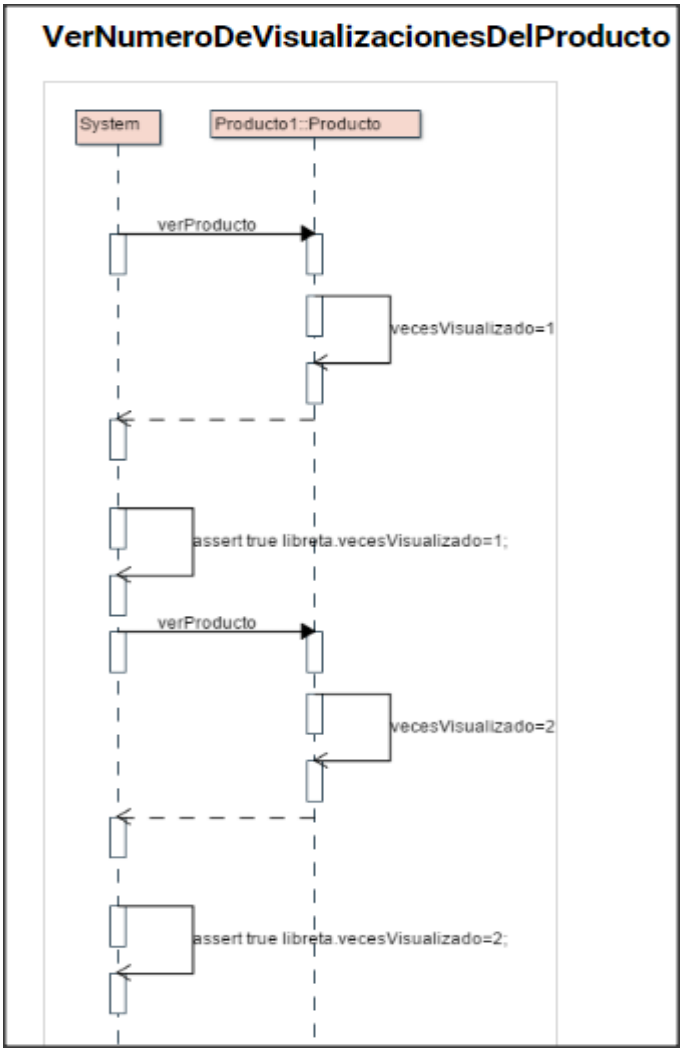


Figura 48.- Resultado final de la generación de diagramas de secuencia

Al acabar de generarse los diagramas en una generación de documentación, se guardan en *XML* a partir de la función *getXMLString* perteneciente a librería, para su posterior uso en una visualización *online* de la documentación, evitando así regenerarlo si no ha habido cambios.

7 PLAN DE PRUEBAS

El *testing* de este proyecto, se ha hecho a dos niveles: pruebas funcionales y pruebas de rendimiento. Las pruebas funcionales, se realizan con la intención de validar que el sistema, cumple con los requisitos funcionales descritos en la fase de especificación (ver capítulo 3.1), mientras que las pruebas de rendimiento, se centran en comprobar cómo responde el sistema ante la ejecución de determinadas funciones.

7.1 PRUEBAS FUNCIONALES

A continuación, se define el plan de pruebas que establece, si los requisitos funcionales del proyecto se han alcanzado; aunque las pruebas, muestran escenarios positivos, también se han validado realizando las mismas acciones para los casos negativos, en los que los criterios de aceptación no se cumplen, asegurando así, que los errores no esperados están controlados por igual.

Este plan de pruebas, se ha ejecutado después de cada cambio nuevo introducido una vez acabada la primera iteración.

Caso de prueba #1	**Generar documentación**
Acciones	1. El usuario, entra en la página de *tests* y realiza una validación satisfactoria. 2. El usuario, entra en la página de documentación. 3. El usuario, presiona el botón de generar documentación *HTML.*
Criterios de aceptación	• El sistema, muestra un mensaje al usuario al acabar el proceso de generación, indicando que la ejecución se ha realizado correctamente.

Caso de prueba #2	**Visualizar documentación**
Acciones	1. El usuario, entra en la página de *tests* y realiza una validación satisfactoria. 2. El usuario, entra en la página de documentación. 3. El usuario, presiona el botón de generar documentación *HTML.* 4. El usuario, cierra el mensaje emergente de generación finalizada correctamente. 5. El usuario, presiona sobre el enlace a la documentación online.
Criterios de aceptación	• El sistema, muestra un mensaje al usuario al acabar el proceso de generación, indicando que la ejecución se ha realizado correctamente.

	• El sistema, muestra al usuario en el navegador, la última documentación generada de un proyecto.

Caso de prueba #3	**Descargar documentación**
Acciones	1. El usuario, entra en la página de *tests* y realiza una validación satisfactoria. 2. El usuario, entra en la página de documentación. 3. El usuario, presiona el botón de generar documentación *HTML.* 4. El usuario, presiona sobre el enlace de descarga en la ventana emergente de generación finalizada correctamente.
Criterios de aceptación	• El sistema, muestra un mensaje al usuario al acabar el proceso de generación, indicando que la ejecución se ha realizado correctamente. • El sistema, permite la descarga de la documentación en cualquier navegador que utilice el usuario.

Caso de prueba #4	**Cambiar idioma**
Acciones	1. El usuario, entra en la página de *tests* y realiza una validación satisfactoria. 2. El usuario, entra en la página de documentación. 3. El usuario, selecciona el idioma en que desea la documentación. 4. El usuario, presiona el botón de generar documentación *HTML.*
Criterios de aceptación	• El sistema, muestra un mensaje al usuario al acabar el proceso de generación, indicando que la ejecución se ha realizado correctamente. • Tanto la documentación descargable, como la *online,* están en el idioma seleccionado previamente a la generación.

Adicionalmente a estas pruebas, una vez funcionan de manera esperada, según los criterios de aceptación definidos y que la documentación es funcional, se ha comprobado para cada proyecto en Recover, que la traducción en la documentación de los diagramas de clases *UML* y operaciones *OCL* se ejecuta correctamente; como en estos proyectos, las expresiones de las operaciones *OCL* se encuentran combinadas, se ha creado en paralelo un proyecto nuevo, donde probar que cada una funciona por separado; y por otra parte, se ha comprobado también que los diagramas de actividad y de secuencia, se generaban a partir de los casos de uso utilizando el algoritmo correctamente,

validándolos visualmente y examinando el código *JavaScript* generado por el metamodelo.

7.2 PRUEBAS DE RENDIMIENTO

Este tipo de pruebas, tienen un papel muy importante en el proyecto, debido a que es necesario que la autogeneración de documentación sea rápida y eficiente, porque es una funcionalidad muy utilizada por los usuarios de *Recover,* para entregar documentación al cliente.

Para validar que el rendimiento es el deseado, se han hecho las pruebas, utilizado los proyectos existentes en la base de datos de la herramienta, que contienen más cantidad de información a procesar para la generación.

Los resultados obtenidos han sido excelentes, donde para el proyecto de más extensión (aproximadamente 30 clases y 200 casos de prueba), el tiempo de ejecución completo no supera los 15 segundos. Otros resultados probados sobre proyectos más reducidos que se utilizan actualmente, ejecutan el proceso en menos de 5 segundos.

7.3 GESTIÓN DE DEFECTOS E INFORMES

Trabajando en un entorno colaborativo de desarrollo de *software,* es muy importante mantener un control exhaustivo y preciso, de los fallos que puedan aparecer al añadir funcionalidades nuevas a un producto, o arreglar existentes.

La herramienta utilizada para este tipo de gestión es *Jira*, donde cada uno de los defectos encontrados en la ejecución manual del plan de pruebas, es enlazado a su prueba en la plataforma *TestLink*[23] y almacenado como *bug* a resolver con la siguiente información*:*

- **Precondiciones:** Estado previo a la realización del test o configuración previa.
- **Comportamiento esperado:** Qué resultados se esperan del test.
- **Comportamiento actual:** Qué resultados se están obteniendo.
- **Pasos para reproducirlo:** Conjunto de pasos a seguir para obtener el defecto**.**
- **Información extra:** Datos de interés complementarios (imágenes, videos, etc…).

A partir de estos datos, se saben todos los detalles del defecto y permite abordar el problema sabiendo detalles de lo sucedido.

Figura 49.- Ciclo de vida de un defecto

En este proyecto, cada defecto abierto tiene su ciclo de vida, que va cambiando según va avanzando el trabajo sobre ellos (ver Figura 49); Los estados posibles que puede adoptar en el tiempo son los siguientes:

- **To do:** El defecto está pendiente de solucionar.
- **In progress:** Se está trabajando para solucionar el defecto.
- **Review:** Revisión de que la solución al defecto funciona correctamente.
- **Done:** Defecto solucionado con éxito.

Cabe destacar en cuanto al ciclo de vida, que un defecto en los estados *'In progress'* o *'Done'* pueden volver a aparecer en cualquier ocasión, en cuyo caso volverá a pasarse al estado *'To do'*.

A modo de resumen del flujo de pruebas, se incluye la Figura 50, donde el *tester* hace las pruebas manuales que están registradas en *TestLink*, genera un informe del resultado de la prueba y dependiendo de este, crea el defecto en *Jira* con estado *'To do'* o lo registra como prueba pasada.

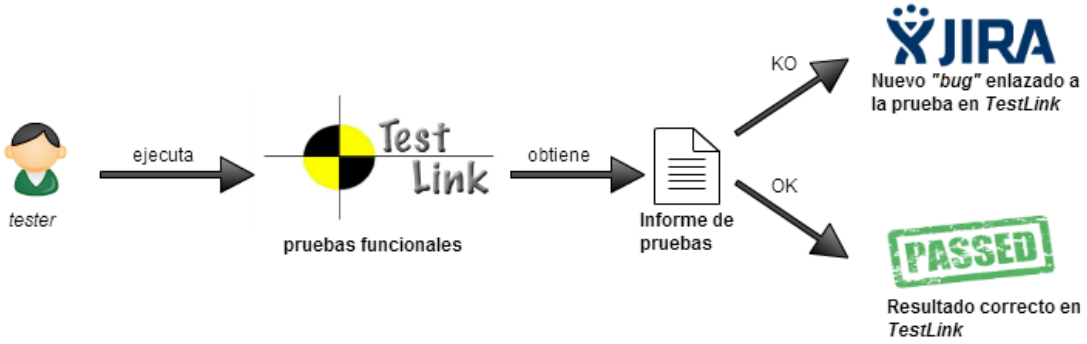


Figura 50.- Flujo de pruebas completo

8 GESTIÓN DEL PROYECTO

8.1 PLANIFICACIÓN TEMPORAL

8.1.1 Duración estimada del proyecto

El proyecto, tiene una duración estimada de cuatro meses y medio, con fecha inicial el 14 de setiembre del 2015, coincidiendo con el inicio de GEP (Gestión de Proyectos) y definiendo la semana del 25 de enero del 2016 como fecha límite, dónde se realizará la presentación del proyecto.

8.1.2 Descripción de tareas

En este apartado, se describe en detalle cada etapa dentro del plan de proyecto, que consta de cinco fases bien diferenciadas.

Gestión de proyectos (GEP)

En esta fase, se elaboran todos los documentos de gestión del proyecto e incluye todos los entregables de la asignatura GEP. Esta es una de las etapas más importantes, ya que se especifica qué se va a realizar en el proyecto y la manera en que se llevará cabo.

La duración es aproximadamente de un mes, una vez finalizada, el proyecto quedará definido en términos de alcance, planificación, presupuesto y sostenibilidad; esta fase, no tiene ninguna dependencia de precedencia, pero cuenta con un calendario de entregas a seguir.

Primera Iteración: Generación de documentación básica

Esta primera iteración, se aborda el desarrollo del mecanismo de generación de documentación web. Se espera obtener un primer producto entregable integrado sobre *Recover*, donde se autogenere una documentación navegable y descargable en formato '*zip*'.

Para ello, se realizarán todas las fases del ciclo de vida del desarrollo *software* (análisis de requisitos, especificación, implementación y *testing*), cada una dependiente de la anterior y en orden secuencial. La dependencia de precedencia de esta etapa proviene del módulo GEP.

Segunda Iteración: Traducción de *UML* y *OCL* a lenguaje natural

En esta iteración se añade un componente, que extiende las capacidades funcionales del motor de generación desarrollado en la primera iteración, mediante la implementación de la traducción *UML* y *OCL* a lenguaje natural. El objetivo de esta traducción, es conseguir una nueva versión entregable del producto que no requiera conocer el lenguaje *UML/OCL*; concretamente, se traduce el esquema conceptual y sus operaciones *OCL*.

Se continuará con la misma metodología de desarrollo que en la primera iteración, basándose en el ciclo de vida de desarrollo de *software* en orden secuencial. La dependencia de precedencia de esta etapa, proviene de la primera iteración, ya que es necesario tener la documentación generada para realizar su traducción.

Tercera iteración: Diagramas de actividad y de secuencia

En esta fase, se añade la última ampliación al motor de generación y se obtiene el producto final del proyecto. El nuevo módulo a implementar, permitirá tener una representación mediante diagramas de actividad y de secuencia de los casos de uso existentes en la documentación.

Al igual que en las demás iteraciones, se seguirá el ciclo de vida de desarrollo de software en orden secuencial. La dependencia de precedencia de esta etapa, proviene de la primera iteración, ya que se necesita la documentación generada y los casos de uso añadidos en la primera iteración para añadir la ampliación.

Etapa final

Durante esta última tarea, se acabará de preparar la memoria del proyecto, así como su presentación una vez cerrada la memoria; estos dos artefactos contarán con la información generada de todas las fases anteriores.

La memoria del proyecto se habrá ido elaborando a medida que avanza el propio proyecto, por este motivo contará con toda la documentación de la fase GEP, el análisis de requisitos y diseño de cada una de las iteraciones; por tanto, en esta etapa el trabajo sobre la memoria se reduce a ampliación y correcciones.

Las dependencias de precedencia de esta tarea; provienen de todas las tareas anteriores.

8.1.3 Recursos

Los recursos previstos que se utilizarán para este proyecto son:

Recursos humanos: Una única persona con una dedicación en el proyecto de 25 horas semanales, asumiendo todos los roles (analista, programador, *tester*…).

Recursos materiales: Un ordenador de sobremesa y un servidor donde alojar la aplicación *Recover.*

Recursos *software*:

- Entorno de desarrollo para tecnología JBoss: Eclipse IDE.
- Sistema de control de versiones: GitHub y Git para su gestión.
- Sistema de control de tareas: Jira.
- Herramientas de documentación: Microsoft Word 2013, Project 2013, Excel 2013, PowerPoint 2013.
- Herramienta de diseño y especificación de *software*: Enterprise Architect
- Gestor de documentos *on-line*: Google Drive

8.1.4 Estimación del tiempo

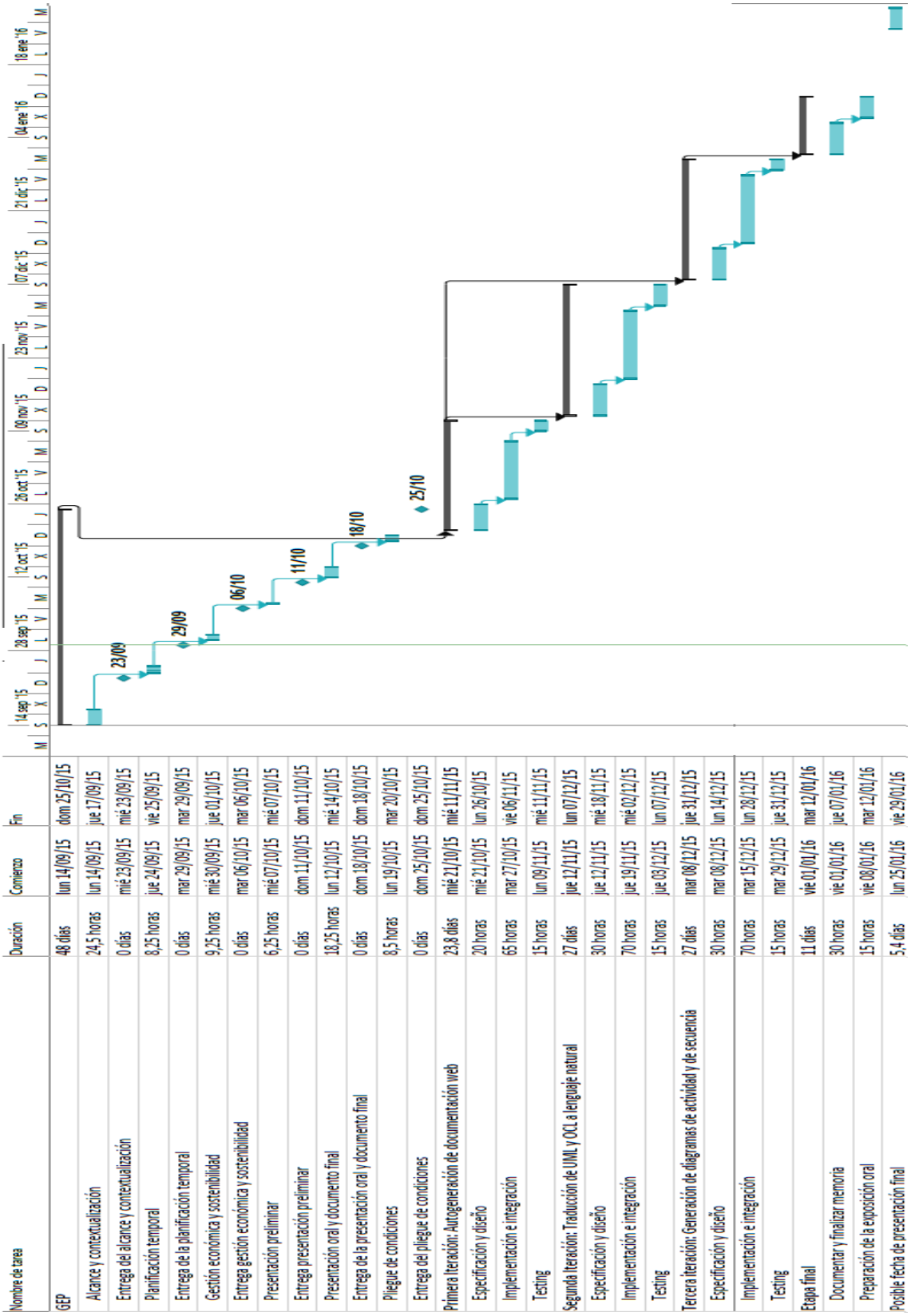
A continuación, se muestra detalladamente la tabla del número de horas de dedicación a cada tarea:

Tarea	Dedicación (horas)
Gestión de Proyectos (GEP)	**75**
Alcance y contextualización	24,50
Planificación temporal	8,25
Gestión económica y sostenibilidad	9,25
Presentación preliminar	6,25
Presentación oral y documento final	18,25
Pliegue de condiciones	8,5
Primera Iteración: Generación de documentación básica	**100**
Especificación y diseño	20
Implementación e integración	65
Testing	15
Segunda Iteración: Traducción de *UML* y *OCL* a lenguaje natural	**115**
Especificación y diseño	30
Implementación e integración	70
Testing	15
Tercera Iteración: Generación de diagramas de actividad y secuencia	**115**
Especificación y diseño	30
Implementación e integración	70
Testing	15
Etapa final	**45**
Documentar y finalizar memoria	30
Preparación de la exposición oral	15
Total	**450**

Tabla 8.- Dedicación al proyecto (en horas)

Diagrama de *Gantt*

El diagrama de *Gantt* representa la planificación temporal del proyecto, incluyendo todas las tareas mencionadas con su fecha de inicio, fecha final y duración en horas.



Nombre de tarea	Duración	Comienzo	Fin
GEP	48 días	lun 14/09/15	dom 25/10/15
Alcance y contextualización	24,5 horas	lun 14/09/15	jue 17/09/15
Entrega del alcance y contextualización	0 días	mié 23/09/15	mié 23/09/15
Planificación temporal	8,25 horas	jue 24/09/15	vie 25/09/15
Entrega de la planificación temporal	0 días	mar 29/09/15	mar 29/09/15
Gestión económica y sostenibilidad	9,25 horas	mié 30/09/15	jue 01/10/15
Entrega gestión económica y sostenibilidad	0 días	mar 06/10/15	mar 06/10/15
Presentación preliminar	6,25 horas	mié 07/10/15	mié 07/10/15
Entrega presentación preliminar	0 días	dom 11/10/15	dom 11/10/15
Presentación oral y documento final	18,25 horas	lun 12/10/15	mié 14/10/15
Entrega de la presentación oral y documento final	0 días	dom 18/10/15	dom 18/10/15
Pliegue de condiciones	8,5 horas	lun 19/10/15	mar 20/10/15
Entrega del pliegue de condiciones	0 días	dom 25/10/15	dom 25/10/15
Primera Iteración: Autogeneración de documentación web	23,8 días	mié 21/10/15	mié 11/11/15
Especificación y diseño	20 horas	mié 21/10/15	lun 26/10/15
Implementación e integración	65 horas	mar 27/10/15	vie 06/11/15
Testing	15 horas	lun 09/11/15	mié 11/11/15
Segunda Iteración: Traducción de UML y OCL a lenguaje natural	27 días	jue 12/11/15	lun 07/12/15
Especificación y diseño	30 horas	jue 12/11/15	mié 18/11/15
Implementación e integración	70 horas	jue 19/11/15	mié 02/12/15
Testing	15 horas	jue 03/12/15	lun 07/12/15
Tercera iteración: Generación de diagramas de actividad y de secuencia	27 días	mar 08/12/15	jue 31/12/15
Especificación y diseño	30 horas	mar 08/12/15	lun 14/12/15
Implementación e integración	70 horas	mar 15/12/15	lun 28/12/15
Testing	15 horas	mar 29/12/15	jue 31/12/15
Etapa final	11 días	vie 01/01/16	mar 12/01/16
Documentar y finalizar memoria	30 horas	vie 01/01/16	jue 07/01/16
Preparación de la exposición oral	15 horas	vie 08/01/16	mar 12/01/16
Posible fecha de presentación final	5,4 días	lun 25/01/16	vie 29/01/16

Figura 51.- Diagrama de Gantt del proyecto

Consideraciones adicionales sobre diagramas

No se ha elaborado un diagrama *PERT*, porque las tareas serán realizadas en orden secuencial, debido a la metodología y a que tan sólo hay un responsable de su realización. Esta situación supone una **dependencia de precedencia** de una tarea con su anterior, lo que implica que la duración total del proyecto, es la suma de horas de las tareas y tan solo hay un camino crítico.

8.1.5 Valoración de alternativas y plan de acción

Se tiene que considerar, que es un proyecto llevado a cabo por una sola persona, lo que puede suponer ciertas desviaciones en la planificación; por ello, se ha tenido en cuenta un **margen de tiempo** desde la etapa final hasta el turno de lectura del proyecto (del 12 de enero a la semana del 25 de enero). De esta forma se garantiza un margen de tiempo en caso de existir alguna desviación, y este margen puede ser eliminado en caso de acabar las tareas en el tiempo establecido o utilizado para realizar mejoras opcionales.

Para conseguir llevar a cabo este sistema y aprovechar al máximo su flexibilidad, será necesario realizar un **seguimiento en cada iteración** de la planificación, permitiendo detectar posibles desviaciones de iteraciones futuras.

En cualquier caso, a partir de lo mencionado, la aparición de algún tipo de desviación estaría controlada y no afectaría ni al consumo de recursos ni a la duración total del proyecto. En consecuencia, se garantiza que la finalización del proyecto se realizará en el tiempo establecido.

8.1.6 Desviaciones de planificación

La planificación inicial del proyecto se ha visto modificada como consecuencia de los imprevistos que han ido surgiendo a lo largo de las iteraciones. A continuación, se muestra la planificación inicial y la actual de las iteraciones con sus diferencias justificadas.

	Planificación inicial	Planificación final
Primera iteración	**(21/10/15 – 11/11/15)**	**(21/10/15 – 07/11/15)**
Especificación y diseño	21/10/15 – 26/10/15	**21/10/15 – 26/10/15**
Implementación e integración	27/10/15 – 06/11/15	**27/10/15 – 03/11/15**
Testing	09/11/15 – 11/11/15	**05/10/15 –07/11/15**
Segunda iteración	**(12/11/15 – 07/12/15)**	**(09/11/15 – 15/12/15)**
Especificación y diseño	12/11/15 – 18/11/15	**09/11/15 – 17/11/15**
Implementación e integración	19/11/15 – 02/12/15	**18/11/15 – 07/12/15**
Testing	03/12/15 – 07/12/15	**08/12/15 – 15/12/15**

Tercera iteración	(08/12/15 – 31/12/15)	(16/12/15 – 08/01/16)
Especificación y diseño	08/12/15 – 14/12/15	**16/12/15 – 23/12/15**
Implementación e integración	15/12/15 – 28/12/15	**24/12/15 – 05/01/16**
Testing	29/12/15 – 31/12/15	**06/12/16 – 08/01/16**
Etapa final	**(01/01/16 – 12/01/16)**	**(11/01/16 – 21/01/16)**
Documentar y finalizar memoria	01/01/16 – 07/01/16	**11/01/16 – 17/01/16**
Preparación exposición oral	08/01/15 – 12/01/16	**18/01/16 – 21/01/16**

Los cambios observables en la primera iteración, se deben a que en la fase de implementación e integración, se ha creado la documentación básica autogenerada más rápido de lo esperado; además, la integración ha sido sencilla gracias a la buena organización de la herramienta *Recover*, por ello se ha visto reducido el tiempo.

En la segunda iteración la dificultad ha aumentado considerablemente. Al tratarse de una librería tan amplia, se ha tenido que realizar un estudio más exhaustivo del esperado inicialmente, tanto en especificación y diseño como en implementación e integración. Concretamente, en diseño se ha tenido que decidir la mejor manera de abordar la traducción a lenguaje natural sobre la arquitectura de la librería una vez estudiada. Por lo que respecta a la implementación, se ha debido estudiar la lógica de la traducción, lo cual ha supuesto un mayor estudio.

A partir de la tercera iteración, se tuvieron en cuenta las desviaciones surgidas en la segunda iteración, modificando la planificación a una más precisa y realista; este cambio ha resultado ser acertado, ajustándose a una situación real y permitiendo acabar el proyecto en los plazos establecidos.

8.2 GESTIÓN ECONÓMICA

Una vez planteados los problemas que pueden surgir y sus posibles soluciones, es necesario elaborar un estudio económico, con el fin de estimar la viabilidad del proyecto; para hacer este estudio, es necesario identificar y estimar los diversos costes de la realización del proyecto.

8.2.1 Identificación y estimación de costes

Costes directos por actividad

Recursos humanos

En este apartado, analizamos el coste de los recursos humanos. Para la realización de este proyecto se requiere de personal especializado en cada una de las fases.

A continuación, se muestra la relación del coste por hora, en relación al rol desempeñado en la empresa.

Rol	Actividad	Coste por hora
Jefe de proyecto	Participa en la gestión del proyecto y su cierre en la etapa final	35 €
Analista	Establece los requisitos y objetivos del proyecto	30 €
Arquitecto	Realiza el diseño del proyecto	30 €
Programador	Participa en el desarrollo e integración en *Recover*	20 €
Tester	Realiza pruebas sobre el desarrollo del programador	20 €

Tabla 9.- Costes por rol

A continuación, se detalla el cálculo del coste en recursos humanos para la actividad planificada en el diagrama de Gantt, mostrando para cada una de las tareas, el rol que la realiza y el número de horas que dedica. Cada uno de estos cálculos se ha realizado basándose en los costes de la Tabla 9.

Fase	Nombre de la tarea	Horas	Recurso	Coste
Gestión del proyecto	Alcance y contextualización	75	Jefe de proyecto	2.625 €
	Planificación temporal			
	Gestión económica y sostenibilidad			
	Presentación preliminar			

	Presentación y documento final			
	Pliego de condiciones			
Desarrollo del proyecto	Especificación y diseño	80	Analista [50%] Arquitecto [50%]	2.400 €
	Implementación e integración	205	Programador	4.100 €
	Testing	45	*Tester*	900 €
Etapa final	Documentar y finalizar memoria	45	Jefe de proyecto	1.610 €
	Preparación de la exposición oral			
Total				**11.635 €**

Tabla 10.- Costes directos por actividad Gantt

Recursos *software* y *hardware*

En este apartado, se detallan los costes de los recursos *software* y *hardware* que se utilizarán en el proyecto, junto con su correspondiente coste. Para este cálculo, se presupone que el material será utilizado durante el transcurso del proyecto.

Recurso	Unidad	Coste total	Tiempo de vida / duración	Coste
Lenovo ThinkStation P300	1	650 €	4 años	650 €
Dell PowerEdge R320 Server	1	760 €	4 años	760 €
Microsoft Office 2013	1	150 €	1 año	150 €
Enterprise Architect	1	130 €	1 año	130 €
Eclipse JBoss	1	-	-	0 €
Jira	1	35 €	1 año	35 €
GitHub	1	25 €	1 mes	113 €
Git	1	-	-	0 €
Google Drive	1	-	-	0 €
Total				**1.838 €**

Tabla 11.- Costes por recursos software y hardware

Costes indirectos

Internet: se utiliza una conexión a internet con 40 €/mes de cuota y suponemos que un 35% del ancho de banda, se destina al proyecto durante los 4 meses y medio de su duración.

Impresiones: la memoria final del proyecto, debe ser impresa para 3 miembros del tribunal; esta memoria tendrá una extensión aproximada de 150 páginas y la impresión de cada una tiene un coste de 0,05€.

Consumo eléctrico: el servidor tiene un consumo de 350W y el ordenador de 280W. Suponemos un precio de 0,1400 €/kWh, un uso diario de 24 horas por parte del servidor y 5 horas el ordenador; considerando la información anterior, el consumo del servidor y el ordenador es de 300kW y 43kW al mes aproximadamente.

Recurso	Unidad	Precio	Coste
Internet	4 meses (35%)	40 € / mes	56 €
Impresiones	450 páginas	0,05 € / página	23 €
Consumo servidor	300 kW/mes (24 h)	0,1400€ / kWh	168 €
Consumo ordenador	42 kW/mes (5 h)	0,1400€ / kWh	24 €
Total			**271 €**

Tabla 12.- Costes indirectos

8.2.2 Contingencias e imprevistos

Para poder hacer frente a posibles imprevistos que surjan de la realización del proyecto, se destina un margen de la suma del 15% de costes directos e indirectos.

Concepto	Coste	Porcentaje de contingencia	Coste de contingencia
Costes directos	13.473 €	15%	2.021 €
Costes indirectos	271 €	15%	41 €
Total			**2.063 €**

Tabla 13.- Partida de contingencia e imprevistos

Los imprevistos que pueden surgir durante el proyecto, son los siguientes:

- **Avería del ordenador o servidor:** Aunque la probabilidad de avería es muy baja, hay que tener en cuenta los posibles fallos de *hardware* y afrontar su reparación o substitución.
- **Pérdida de datos:** Hay que tener en cuenta que existe la posibilidad de que surjan fallos en *Google Drive* o *Git,* por eso se realizarán copias de seguridad diarias. No obstante, aun realizando esto se podría perder un máximo de un día de trabajo, lo que supondría un coste adicional en recuperación.
- **Retraso de dos semanas:** En el plan de acción de la planificación, se ha mencionado el margen de dos semanas para posibles desviaciones del proyecto; en caso de necesitar este tiempo, tenemos que contar con un programador durante las dos semanas y un analista a media jornada para supervisar los requisitos, haciendo un total de 35€/h para semanas de 25h de trabajo.

El hecho de haber planificado un margen de tiempo extra para posibles desviaciones, haría que estos imprevistos no afectaran de manera crítica al proyecto, no obstante, si hubiese algún otro imprevisto se podría gestionar gracias al margen del plan de contingencia.

8.2.3 Coste total

A continuación, se muestra un resumen de todos los costes mencionados anteriormente y el coste total de realizar el proyecto.

Concepto	Coste
Recursos humanos	11.635 €
Recursos *software* y *hardware*	1.838 €
Costes indirectos	271 €
Contingencias e imprevistos	2.063 €
Total	**15.807 €**

Tabla 14.- Coste total del proyecto

8.2.4 Control de gestión

Es necesario llevar un control de los recursos, para ver si procede el uso del plan de contingencia. Para la gestión de recursos humanos, se llevará un registro de las horas que suponen las tareas, a medida que se van realizando a lo largo de la iteración; con este método, se pretende comprobar que se está cumpliendo con la planificación y corregir posibles desviaciones en iteraciones futuras.

Al finalizar del proyecto, gracias a la planificación real de este, se podrá calcular el coste real y la diferencia a la estimación realizada inicialmente.

8.2.5 Viabilidad

Este proyecto, se desarrolla con el fin de obtener una ampliación importante sobre *Recover* que ha sido demandada por clientes; la herramienta en su estado actual, ya aporta beneficios, al estar contratada por varios clientes, por tanto, el añadir esta nueva funcionalidad, permitirá conservar a los clientes demandantes y llamar la atención de nuevos.

A partir de lo mencionado y después de analizar económicamente el coste de su realización, se llega a la conclusión de que la adición de esta nueva característica es económica, supone un valor añadido a la herramienta y aportará beneficios con seguridad; en definitiva, el proyecto es viable y competitivo económicamente.

8.2.7 Desviaciones de presupuesto

Seguidamente, se muestra el balance de coste por iteración que suponen los cambios de planificación:

Iteración	Concepto	Implicados	Balance (aprox)
Primera iteración	Especificación y diseño	Analista[50%,30€/h] Arquitecto [50%,30€/h]	**0**
	Implementación e integración	Programador (20€/h)	-15h (**+300€**)
	Testing	Tester (20€/h)	**0**
Segunda iteración	Especificación y diseño	Analista[50%,30€/h] Arquitecto [50%,30€/h]	+10h(**-300€**)
	Implementación e integración	Programador (20€/h)	+20h(**-400€**)
	Testing	Tester (20€/h)	**0**
Tercera iteración	Especificación y diseño	Analista[50%,30€/h] Arquitecto [50%,30€/h]	+5h (**-150€**)
	Implementación e integración	Programador (20€/h)	-5h (**+100€**)
	Testing	Tester (20€/h)	**0**
Balance total			**-450€**

Tabla 15.- Desviaciónes en el presupuesto

Como efecto del cambio a la planificación, podemos observar de la Tabla 15 que esto implica una pérdida de 450€. No obstante, el hecho de haber tenido en cuenta un margen de tiempo y un presupuesto para contingencias e imprevistos de 2063€, implica que esta cantidad podrá ser asumida y no afectará negativamente al proyecto ni a los objetivos de este.

También ha habido cambios referentes a los recursos *software*, ahorrando 130€, al no utilizar la herramienta de modelado Enterprise Architect y substituirla por Cacoo, una solución gratis y *cloud.*

Haciendo un recuento del presupuesto con todos los nuevos costes con las desviaciones, obtenemos el precio final (Tabla 16); este no se ve afectado, debido al uso de la cantidad reservada para contingencias e imprevistos.

Concepto	Coste
Recursos humanos	12.085 €
Recursos *software* y *hardware*	1.708 €
Costes indirectos	271 €
Contingencias e imprevistos	1.743 €
Total	**15.807 €**

Tabla 16.- Precio final del proyecto

8.3 SOSTENIBILIDAD Y COMPROMISO SOCIAL

A continuación, se presenta el estudio de sostenibilidad referente al proyecto; en este se muestra el impacto que provoca la realización del proyecto para las dimensiones económica, social y ambiental.

8.3.1 Dimensión económica

Se ha realizado una evaluación de costes tanto en recursos materiales como humanos, teniendo en cuenta el coste de posibles desajustes durante el desarrollo del proyecto.

Los recursos utilizados para la realización de este proyecto, se han intentado ajustar al máximo, lo que implica que probablemente sea imposible realizar un proyecto similar por un coste menor o con una diferencia significativa; además, se han utilizado librerías de terceros, ahorrando tiempo a la parte de desarrollo, reutilizado tecnologías existentes y dedicando así a cada tarea su tiempo proporcional a su importancia.

La herramienta de innovación *Recover,* se utiliza actualmente en varios proyectos de la empresa, lo que implica que la aportación de este proyecto será muy beneficiosa; en definitiva, el coste del proyecto teniendo en cuenta todas las características anteriores resulta viable y competitivo.

8.3.2 Dimensión social

La situación económica de la sociedad actual no es buena, debido a la crisis general que impera, así el sector de la informática y sus empresas punteras también están sufriendo las consecuencias , por ello, cada vez es más importante el ahorro y una buena gestión de los recursos para obtener beneficios.

Debido al crecimiento y la complejidad del *software*, su probabilidad de error aumenta considerablemente, lo que provoca que se tenga que realizar una gran inversión en solucionar fallos; habitualmente, el coste de los errores y mantenimiento suele exceder el de la creación del propio *software,* lo que implica una pérdida de recursos económicos.

Esta pérdida de recursos, podría ser reducida mediante la utilización de la herramienta *Recover* y su nueva incorporación de la documentación navegable, reduciendo

los costes de mantenimiento, cuyos ahorros se podrían destinar a otras tareas que resulten más productivas o a nuevos recursos.

8.3.3 Dimensión ambiental

La naturaleza del proyecto, obliga a la utilización de recursos electrónicos, por tanto es inevitable evitar el impacto ambiental que supondrá en un futuro el *hardware* al final de su tiempo útil de vida; esto se procura contrarrestar mediante la reducción de impresiones en documentación de proyectos software, ya que esta pasa a ser puramente electrónica.

El uso de recursos *hardware* como el ordenador y el servidor de la aplicación implican un consumo y una emisión CO2 de 175 kg por año; para este proyecto habrá una emisión de CO2 aproximada de 66kg por parte del servidor, que estará encendido las 24 horas; el ordenador tendrá un consumo de CO2 aproximado de 13 kg, contabilizando un tiempo de actividad de 5 horas a la semana durante la duración del proyecto.

En resumen, la emisión de CO2 por la realización de este producto, asciende a un total de 0,079 toneladas.

8.3.4 Puntuación final

En la Tabla 17, se muestran los resultados del estudio de sostenibilidad, mediante la matriz propuesta en la guía de la asignatura, cuyas valoraciones están basadas en los comentarios sobre cada una de las dimensiones.

A partir de este resultado, podemos concluir que el proyecto es sostenible.

Sostenibilidad	Económica	Social	Ambiental	Total
Planificación	Viabilidad económica	Mejora en la calidad de vida	Análisis de recursos	
Valoración	8	8	7	**23**
Resultados	Coste final vs previsión	Impacto en el entorno social	Consumo de recursos	
Valoración	9	10	9	**28**
Riesgos	Adaptación a cambios de escenario	Daños sociales	Daños ambientales	
Valoración	-5	0	0	**-5**
Valoración total				**46**

Tabla 17.- Matriz de sostenibilidad

9 CONCLUSIONES

9.1 CONSECUCIÓN DE LOS OBJETIVOS

Una vez finalizado el proyecto, podemos concluir que se han alcanzado con éxito todos los objetivos prestablecidos; las funcionalidades obtenidas responden de la manera esperada, se ha conseguido un mecanismo automático que permite entregar a los clientes que solicitan los servicios de *Recover,* una documentación actualizada sobre un sistema *software*. No obstante, alcanzar los objetivos no ha sido sencillo, debido a los imprevistos que han surgido a lo largo del proyecto por el estudio y diseño necesario de las librerías.

Cabe destacar, que aunque la planificación temporal ha sufrido desviaciones notables, ninguna de ellas ha sido un impedimento para la finalización del proyecto en la fecha establecida; el haber tenido el margen de desviación temporal y económica, ha servido para solventar estos posibles problemas.

Las soluciones propuestas a las funcionalidades, aunque seguramente sean mejorables, han sido todas estudiadas al detalle y utilizando buenas prácticas aprendidas a lo largo de la carrera en la medida de lo posible; además, el proyecto cumple con todos los requisitos establecidos en la fase de especificación. Podemos concluir que el proyecto se ha llevado a cabo satisfactoriamente.

9.2 TRABAJO FUTURO

Aunque se ha cumplido con las expectativas del proyecto, las funcionalidades implementadas admiten modificaciones e ampliaciones de manera sencilla que pueden enriquecer *Recover* como producto.

A continuación se destacan algunas posibles mejoras, que se deberían añadir como trabajo futuro.

Ampliaciones

- **Traducción *OCL:*** Todavía quedan bastantes expresiones posibles en el lenguaje *OCL* que no están traducidas, sin embargo, sí que están incluidas en la versión de la librería *USE* utilizada y se podrían traducir por si se utilizan en un futuro.

- **Documentación web:** La documentación web, está completa con los apartados esperados en diseño para este trabajo, pero admite ampliaciones de manera sencilla. Como la herramienta *Recover,* se está ampliando continuamente, se podrán ir añadiendo nuevos apartados a la documentación.

- **Diagramas de actividad:** Aunque los diagramas de actividad contienen los elementos necesarios, para expresar el flujo de los casos de uso, aún se puede ampliar más añadiendo elementos que no se han implementado debido al alcance

del proyecto y porque no eran estrictamente necesarios para *Recover*; sin embargo, en un futuro pueden ser útiles si la herramienta lo necesitara, como es el caso de las '*Swimlanes'.*

Mejoras

- **Traducción UML y OCL:** Una mejora interesante en la traducción implementada de este trabajo, sería añadir un servicio que permitiese a la documentación consultar en la ejecución, el género de las palabras a incluir, pudiendo así ser más precisos en la traducción.

- **Mejorar diagramas de actividad:** La generación de diagramas de actividad funciona correctamente, pero en algunos casos las flechas se cruzan por limitación de la librería jsUML2. Se podría implementar un mecanismo, que tuviera en cuenta los solapamientos entre objetos dentro de la librería.

9.3 VALORACIÓN PERSONAL

Haber realizado este proyecto, ha sido una gran experiencia para mejorar, tanto en la ingeniería del software como en el de la programación. El proyecto *Recover,* es una herramienta innovadora, que me ha permitido poner en práctica todos los aspectos aprendidos a lo largo de la carrera y ver nuevas tecnologías de *testing.*

También destacar que gracias a la modalidad empresa, he podido trabajar y aprender de profesionales del sector y cómo funciona el desarrollo de *software* en el ámbito empresarial.

Aunque se han alcanzado los objetivos del proyecto, no ha sido fácil, dado que los retos planteados por este trabajo, han sido de una dificultad bastante elevada y con un tiempo limitado que cumplir; no obstante, esto me ha ayudado a trabajar de una manera más eficiente para lograr los objetivos.

En definitiva, estoy satisfecho con el trabajo realizado, por lo que he aprendido y de haber contribuido a un proyecto tan importante, como es *Recover* para la empresa Sogeti.

9.4 JUSTIFICACIÓN DE LAS COMPETENCIAS TÉCNICAS

Listado de las competencias técnicas asociadas al proyecto y su correspondiente justificación del nivel de logro en el mismo.

CES1.1: Desarrollar, mantener y evaluar sistemas y servicios software complejos y/o críticos [en profundidad]

Se ha elegido esta competencia, porque está directamente relacionada con el desarrollo de cualquier software; para hacer efectivo el objetivo de este proyecto, primero deberá evaluarse el sistema y los requisitos de las funcionalidades a añadir, para previamente desarrollarlas; además, *Recover* es una herramienta de carácter investigador, con una complejidad elevada y en uso comercial, lo que implica un entorno de desarrollo crítico.

CES1.2: Dar solución a problemas de integración en función de las estrategias, estándares y de las tecnologías disponibles [bastante]

Para implementar las funcionalidades necesarias y alcanzar el objetivo del proyecto, se requiere de librerías externas, las cuales tendrán que interactuar con la tecnología disponible en el marco de *Recover*. Por tanto, esto supone la creación de una solución, que nos permita abordar cualquier problema de compatibilidad.

CES1.3: Identificar, evaluar y gestionar los riesgos potenciales asociados a la construcción de software que se pudiesen presentar [bastante]

Esta competencia, se ve reflejada en el hecho de integrar sobre *Recover* las nuevas funcionalidades, las cuales supondrán riesgos, como por ejemplo, problemas de compatibilidad. Estos riesgos, se han identificado en el alcance del proyecto y es decisiva su gestión en caso de surgir, ya que el objetivo final del proyecto, puede verse afectado y con ello el plan de *Recover* como herramienta comercial.

CES1.7: Controlar la calidad y diseñar pruebas en la producción de software [bastante]

El software a implementar, será realizado desde cero y se tendrá que utilizar en un entorno de producción, por tanto, es necesario controlar su calidad. Mediante el diseño de un plan de pruebas exhaustivo, se garantizará que las nuevas funcionalidades serán fiables y robustas para utilizarse con seguridad.

CES2.1: Definir y gestionar los requisitos de un sistema software [bastante]

Es necesario definir y gestionar los requisitos de las nuevas funcionalidades a añadir, con el fin de que cumplan las expectativas que se buscan en el objetivo final y las de las partes interesadas del proyecto.

CES2.2: Diseñar soluciones apropiadas en uno o más dominios de aplicación utilizando métodos de ingeniería del software que integren aspectos éticos, sociales, legales y económicos [en profundidad]

Recover es una herramienta, encaminada a disminuir el coste económico, que suponen los errores en el desarrollo de software y la pérdida de conocimiento funcional de un sistema de información; de este modo, contribuir sobre ella con las nuevas funcionalidades, implica mejorar en el aspecto mencionad y además para el desarrollo, dado que se utilizarán librerías externas, deberán comprobarse aspectos legales, como las licencias y condiciones de uso.

10 GLOSARIO

Definiciones de los conceptos específicos, del ámbito del proyecto y que se requieren para la comprensión de este documento:

Esquema conceptual

En un sistema software, un **esquema conceptual** especifica el conocimiento funcional de sistema de información, incluyendo los tipos de que datos maneja, los posibles estados y las operaciones que permiten pasar de un estado a otro en el sistema.

Gestión del conocimiento

Todo sistema de información, gestiona conocimiento sobre su organización. La **gestión del conocimiento** es el conjunto de habilidades, procesos y metodologías, dirigidos a manejar el conocimiento de un sistema de información; si este conocimiento, está especificado de forma explícita (esquema conceptual), su gestión es más efectiva y objetiva.

Java

Java[24]**,** es un lenguaje de programación imperativo, orientado a objetos y multiplataforma de la empresa *Oracle*[25]. Apareció el año 1995 y es uno de los más utilizados hoy en día.

JavaScript

JavaScript[26][27]**,** es un lenguaje de programación interpretado, orientado a objetos y sin tipos de datos. Aparecido en 1995 y diseñado por *Netscape*[28] y *Mozilla Foundation*[29], se utiliza en el lado del cliente, para la implementación de mejoras dinámicas sobre las páginas web.

Unified Modeling Language (UML)

UML[30]**,** *es* un lenguaje estándar del *Object Management Group (OMG)* especializado en la modelización de sistemas software, a través de diversos diagramas. En este proyecto, se utilizan distintos diagramas (de clases, de secuencia, de actividad…) de *UML* para especificar el conocimiento del sistema, desde distintas visiones estructurales y de comportamiento.

JavaServer Pages (JSP)

JSP[31]**,** es una tecnología que permite desarrollar páginas web dinámicas en HTML con código Java incrustado; se utiliza en *Recover,* para el desarrollo de las páginas que se mostrarán al cliente en su navegador.

JBoss

JBoss[32]**,** es un servidor de aplicaciones implementado en Java, que permite desplegar la aplicación *Recover* y ejecutar todas las peticiones de los usuarios.

Object Constraint Language (OCL)

OCL[33]**,** es un lenguaje estándar del *Object Management Group (OMG*) utilizado para la especificación formal de contratos de operaciones y restricciones de integridad de un esquema conceptual *UML*; permite representar el subconjunto de restricciones de integridad, que no pueden ser especificadas gráficamente.

Test-Driven Development (TDD)

Es una metodología de desarrollo, utilizada en ingeniería del software, que consiste en diseñar las pruebas en función de los requisitos; mediante las pruebas, se verifica si son satisfactorias o fallidas, y en función de esto, se implementa un código que permita obtener un resultado positivo de todas las pruebas.

11 REFERENCIAS

[1] "Sogeti Spain." [Online]. Available: http://www.sogeti.es/. [Accessed: 20-Sep-2015].

[2] N. US Department of Commerce, "National Institute of Standards and Technology."

[3] "The economic impacts of inadequate infrastructure for Software Testing." [Online]. Available: http://www.nist.gov/director/planning/upload/report02-3.pdf. [Accessed: 15-Nov-2015].

[4] A. Tort Pugibet, "Testing and test-driven development of conceptual schemas." Universitat Politècnica de Catalunya, 11-Apr-2012.

[5] A.Tort, A. Olivé, and M. R. Sancho, *An Approach to Test-Driven Development of Conceptual Schemas*. 2011.

[6] A. Olivé and A. Tort, "Testing conceptual schema satisfiability," 2010, pp. 277–288.

[7] A. Olivé, *Conceptual schema-centric development: a grand challenge for information systems research*. Springer, 2005.

[8] A. Tort Pugibet, "CSTL: A Conceptual Schema Testing Language." Universitat Politècnica de Catalunya, 19-Jan-2009.

[9] "Doxygen, Generate documentation from source code." [Online]. Available: http://www.stack.nl/~dimitri/doxygen/. [Accessed: 21-Sep-2015].

[10] "Javadoc Tool ." [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html. [Accessed: 21-Sep-2015].

[11] Imran Sarwar Bajwa, "A natural language processing approach to generate SBVR and OCL," University of Birmingham, 2014.

[12] K. J. David A. Burke, "Translating formal software specifications to natural language/a grammar based approach."

[13] "GF - Grammatical Framework QS." [Online]. Available: http://www.grammaticalframework.org/doc/gf-quickstart.html. [Accessed: 21-Sep-2015].

[14] "The UML-based Specification Environment - USE: UML-based Specification Environment." [Online]. Available:

http://useocl.sourceforge.net/w/index.php/Main_Page. [Accessed: 21-Sep-2015].

[15] "USE Quick Tour." [Online]. Available: http://www.db.informatik.uni-bremen.de/projects/USE/qt.html. [Accessed: 21-Sep-2015].

[16] "Microsoft Visio | Software profesional de flujos de trabajo y diagramas." [Online]. Available: https://products.office.com/es-es/visio/flowchart-software. [Accessed: 21-Sep-2015].

[17] "Visual Paradigm | Software Design Tools for Agile Teams, with UML, BPMN and More." [Online]. Available: http://www.visual-paradigm.com/. [Accessed: 21-Sep-2015].

[18] "Enterprise Architect - UML Design Tools and UML CASE tools for software development." [Online]. Available: http://www.sparxsystems.com.au/products/ea/. [Accessed: 21-Sep-2015].

[19] "jsUML2 Editor: The online UML 2 diagramming tool using the jsUML2 javascript/HTML5 library - UCO.es (Spain)." [Online]. Available: http://www.uco.es/~in1rosaj/tool_jsUML2.html. [Accessed: 21-Sep-2015].

[20] "JUMLY, Advanced modern rendering engine for UML diagram with JavaScript/HTML5/CSS." [Online]. Available: http://jumly.tmtk.net/. [Accessed: 21-Sep-2015].

[21] "JIRA Atlassian." [Online]. Available: https://es.atlassian.com/software/jira. [Accessed: 12-Oct-2015].

[22] "jsUML2: The programmer's guide." [Online]. Available: http://www.uco.es/~in1rosaj/tools/jsUML2/doc/TheProgrammersGuide_003.htm. [Accessed: 19-Jan-2016].

[23] "TestLink." [Online]. Available: http://testlink.org/. [Accessed: 15-Jan-2016].

[24] "Conozca más sobre la tecnología Java." [Online]. Available: https://www.java.com/es/about/. [Accessed: 21-Sep-2015].

[25] "Oracle | Integrated Cloud Applications and Platform Services." [Online]. Available: http://www.oracle.com/index.html. [Accessed: 21-Sep-2015].

[26] "JavaScript | MDN." [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Accessed: 21-Sep-2015].

[27] "JavaScript w3." [Online]. Available: http://www.w3schools.com/js/. [Accessed: 21-Sep-2015].

[28] "Netscape ISP." [Online]. Available: http://isp.netscape.com/. [Accessed: 21-Sep-2015].

[29] "The Mozilla Foundation." [Online]. Available: https://www.mozilla.org/en-US/foundation/. [Accessed: 21-Sep-2015].

[30] "Unified Modeling Language (UML)." [Online]. Available: http://www.uml.org/. [Accessed: 20-Sep-2015].

[31] "JavaServer Pages Technology." [Online]. Available: http://www.oracle.com/technetwork/java/javaee/jsp/index.html. [Accessed: 12-Oct-2015].

[32] "JBoss Developer." [Online]. Available: http://www.jboss.org/. [Accessed: 12-Oct-2015].

[33] "OCL." [Online]. Available: http://www.omg.org/spec/OCL/. [Accessed: 20-Sep-2015].

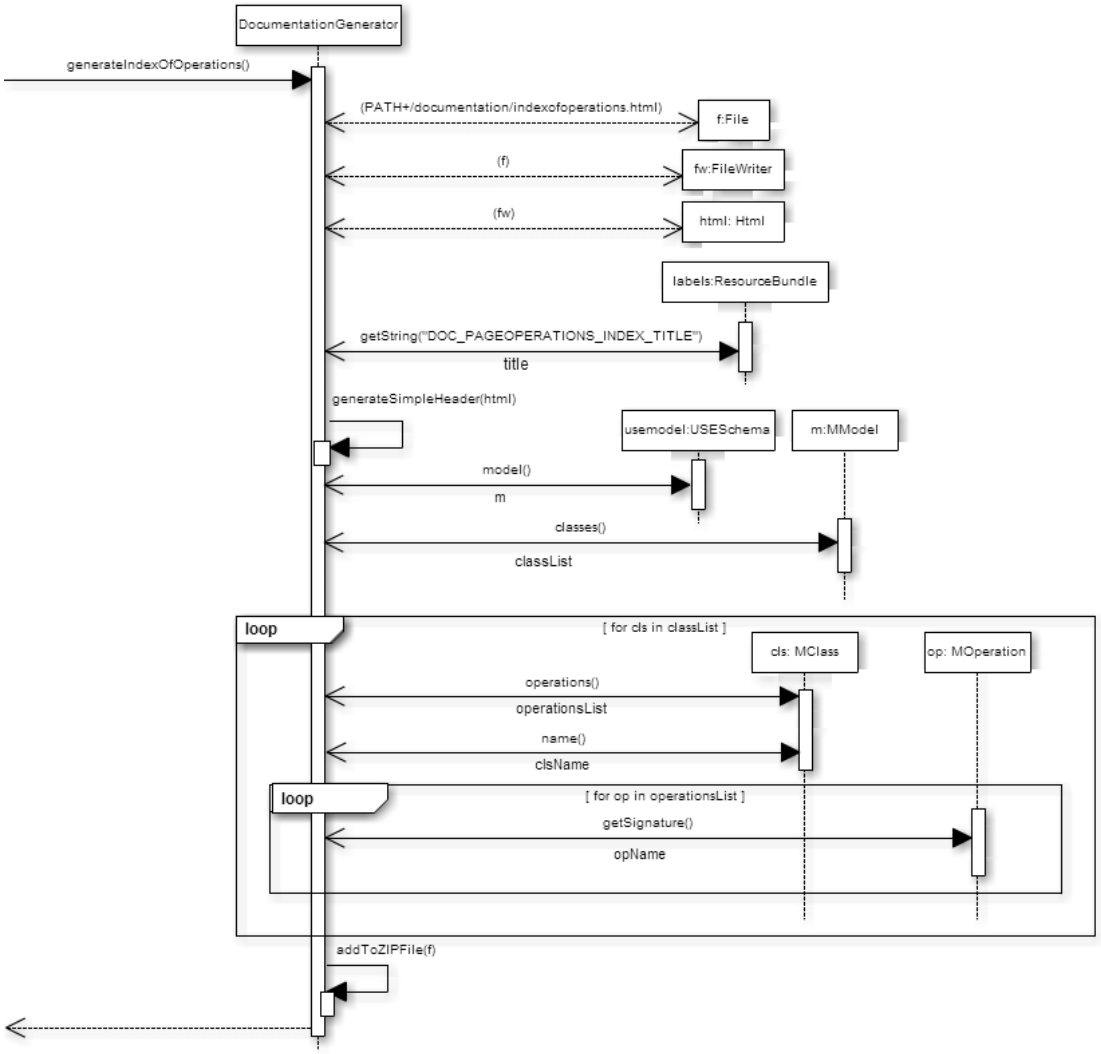
ANEXO A: IMÁGENES COMPLETAS DE LA DOCUMENTACIÓN WEB

Para ver todas las imágenes finales de la documentación, dirigirse al siguiente enlace.

ANEXO B: DIAGRAMAS DE LA DOCUMENTACIÓN AUTO-GENERADA DE *RECOVER*

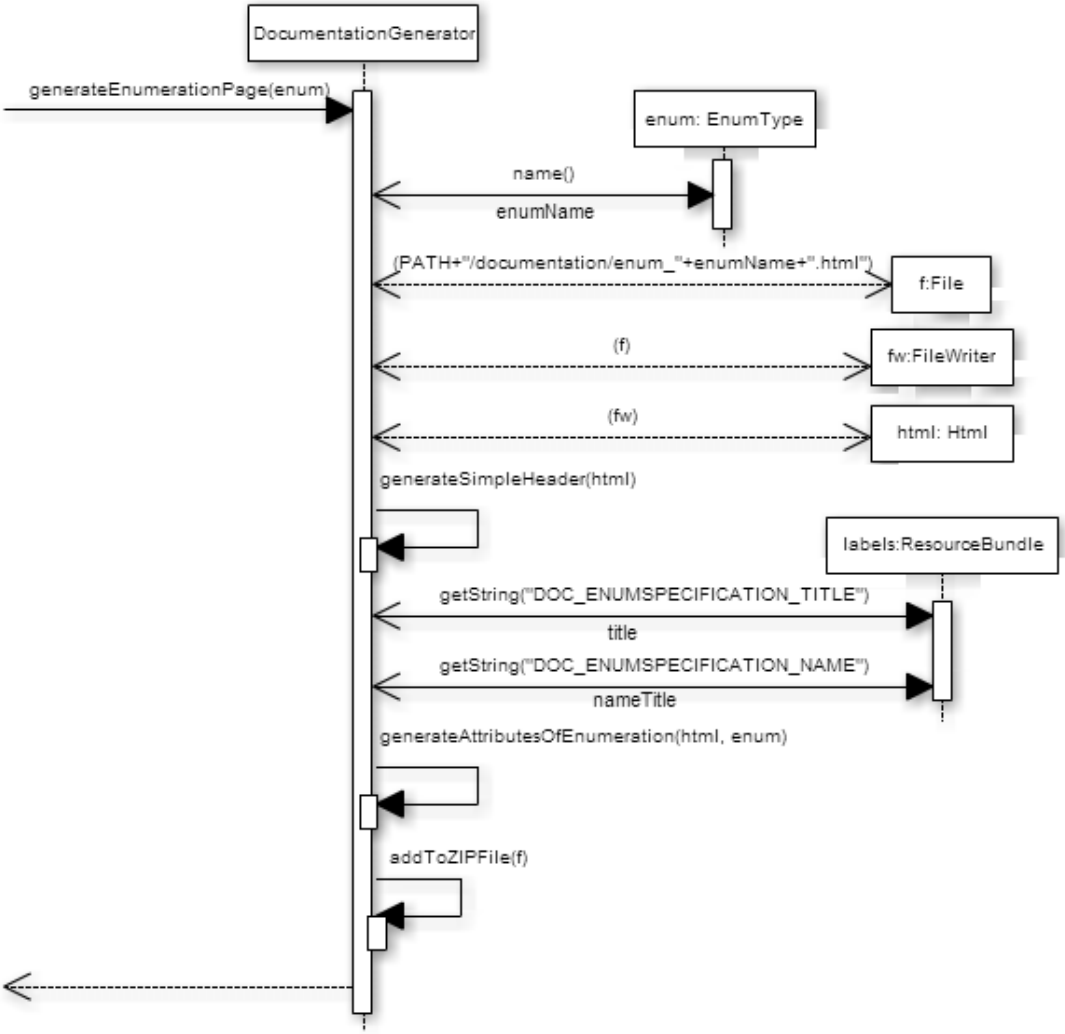
GenerateIndexOfOperations

La siguiente función, es la encargada de generar el índice de las operaciones existentes en el sistema. En primer lugar, se genera el fichero "indexofoperations.html" y se le añade un encabezado simple, que muestra información general sobre la documentación y enlaces a otras secciones de esta: seguidamente, para cada una de las clases del modelo, se recorren sus diferentes operaciones para conseguir formar el índice (cada una de las operaciones de la lista, será un enlace a la clase que la contiene) y una vez acabado todo el proceso, se añade el fichero al *'zip'.*



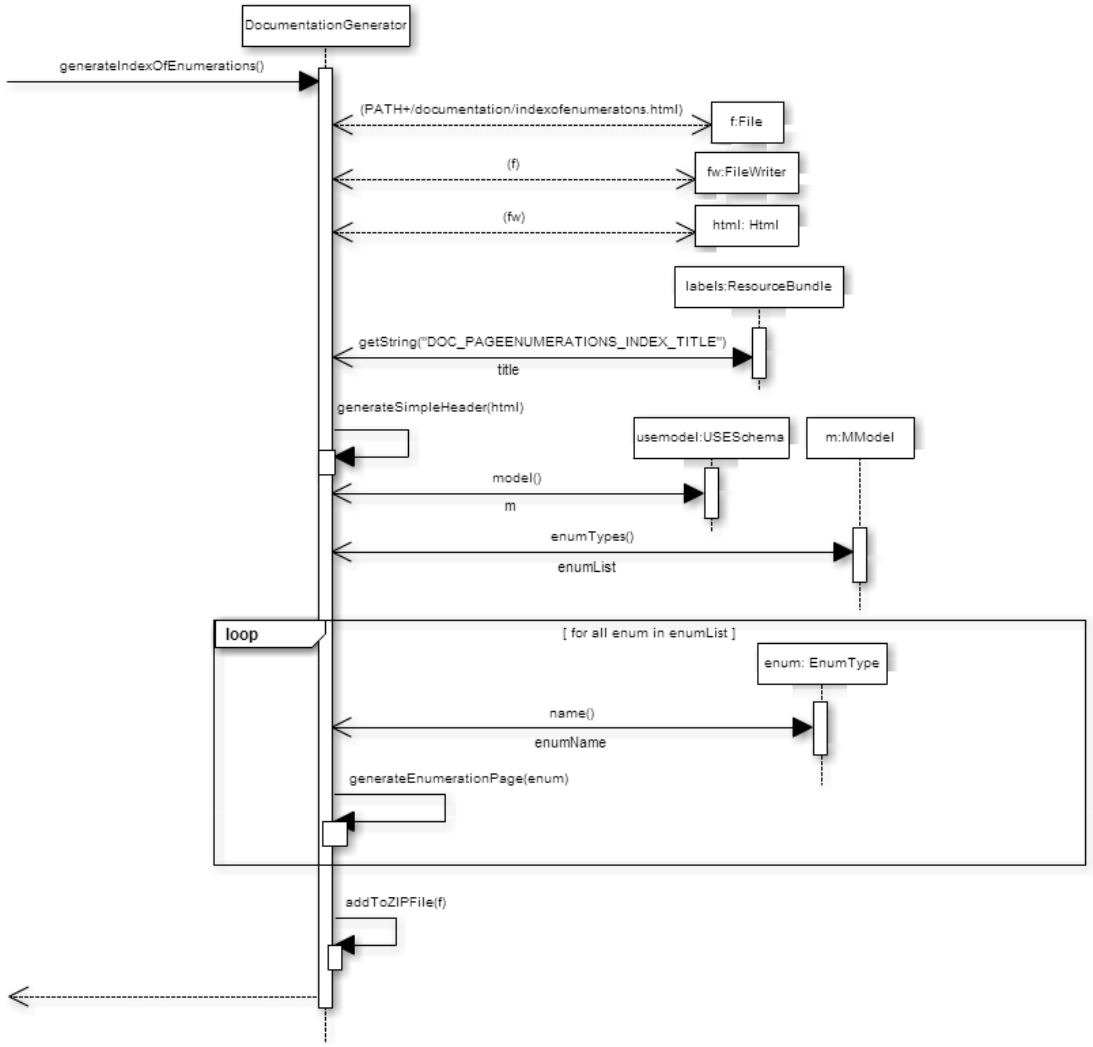
GenerateEnumerationPage

La función *generateEnumerationPage,* es la responsable de generar una página concreta de una enumeración, dónde se describen los detalles de los valores literales a partir de una enumeración *enum* existente en el sistema *Recover*; el proceso empieza mediante la creación de un fichero llamado "enum_enumName.html" (donde *'enumName'* es el nombre de la enumeración específica).



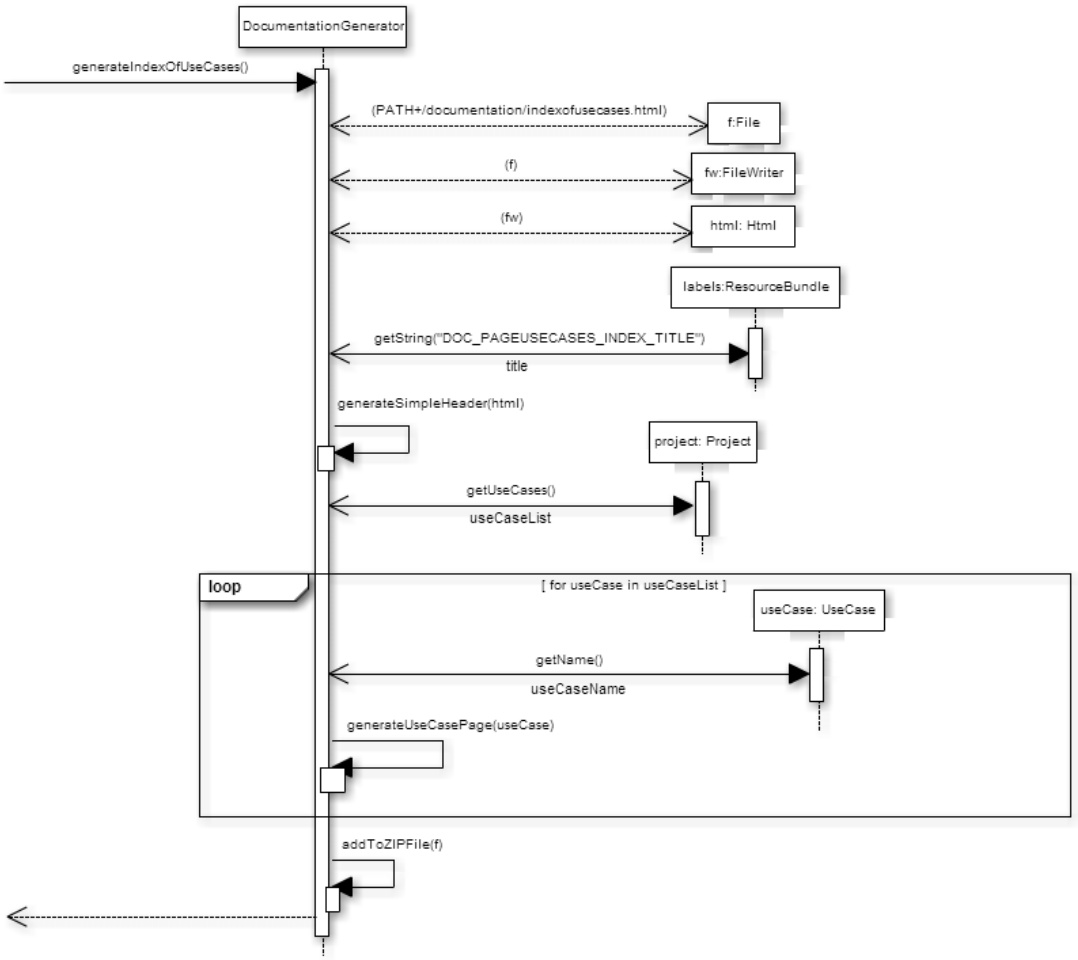
GenerateIndexOfEnumerations

En este diagrama, se muestra el proceso de generación del índice de enumeraciones. Primero se genera el fichero "indexofenumerations.html", que será la página que contendrá el índice de todas las enumeraciones disponibles en el sistema *Recover*. Seguidamente, a partir del modelo se recuperan todas las enumeraciones e iterando sobre ellas, se genera cada una de las páginas específicas para cada enumeración, mediante la función *generateEnumerationPage* y al terminar, se añade la página al fichero *'zip'.*



GenerateIndexOfUseCases

El siguiente diagrama, muestra la operación *generateIndexOfUseCases,* la cual se encarga de generar la página del índice de casos de uso de la documentación sobre el fichero "indexofusecases.html", para ello, se recupera la lista de casos de uso existentes en *Recover* y se recorre cada uno de ellos, generando la página específica de cada caso de uso y al finalizar se añade la página del índice al *'zip'.*



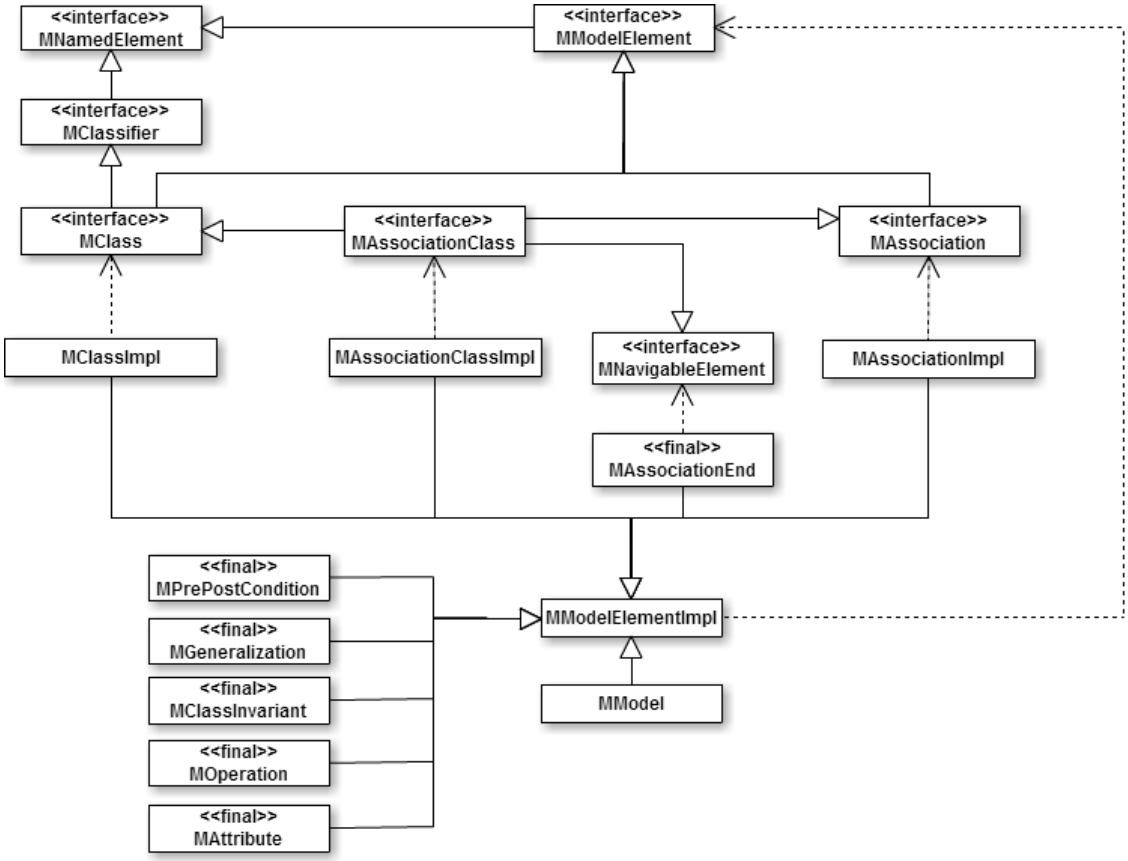
ANEXO C: DIAGRAMAS DE TRADUCCIÓN *UML* Y *OCL* A LENGUAJE NATURAL

Para ver todos los diagramas de la traducción a lenguaje natural, dirigirse al siguiente enlace.

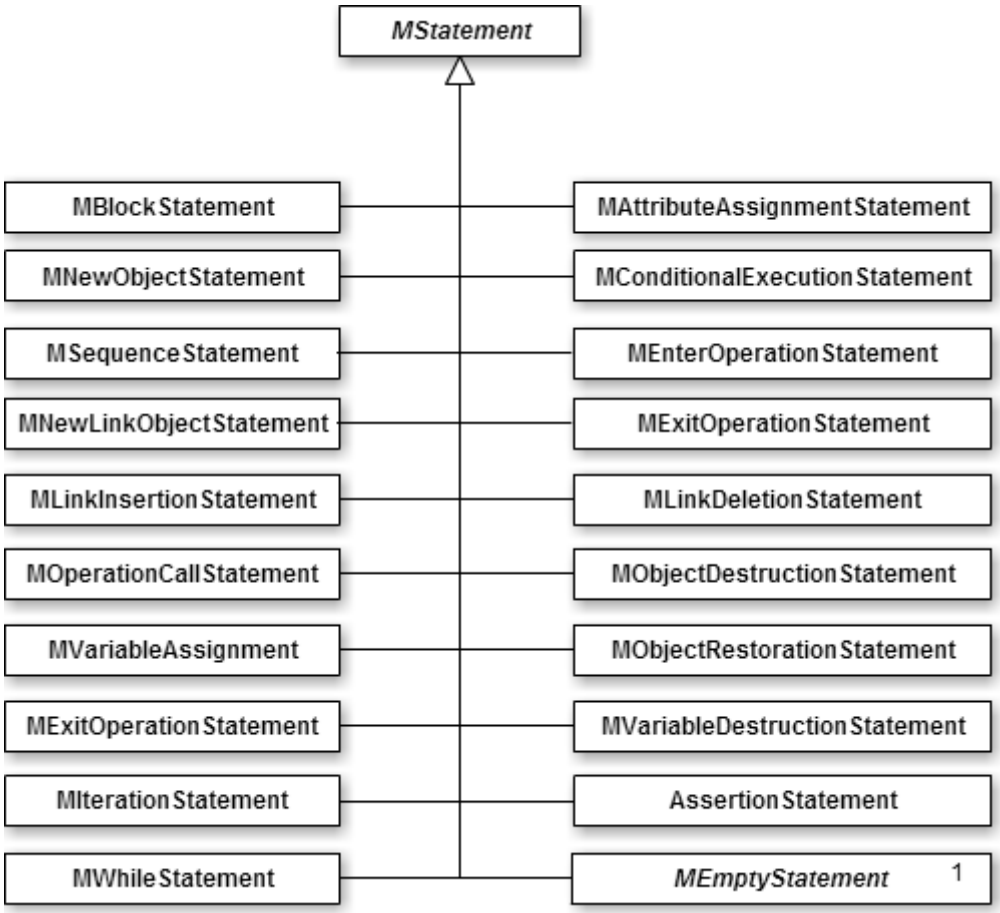
ANEXO D: DIAGRAMAS DE LA LIBRERÍA *USE (UML-BASED SPECIFICATION ENVIRONMENT)*

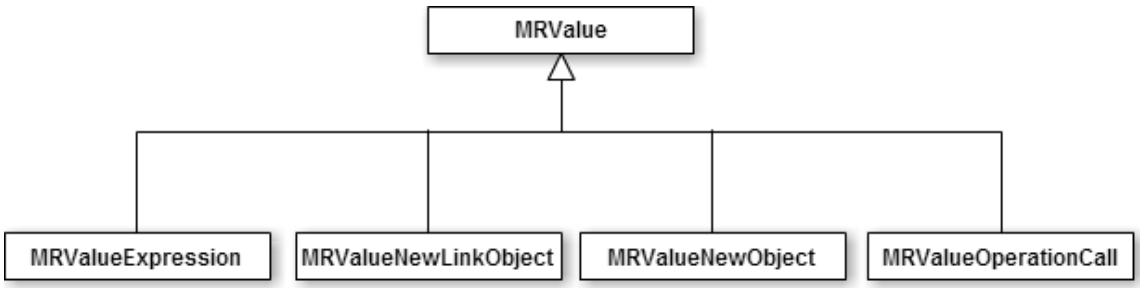
En esta sección tan solo se muestran diagramas generales, para ver todos los diagramas dirigirse al siguiente enlace.

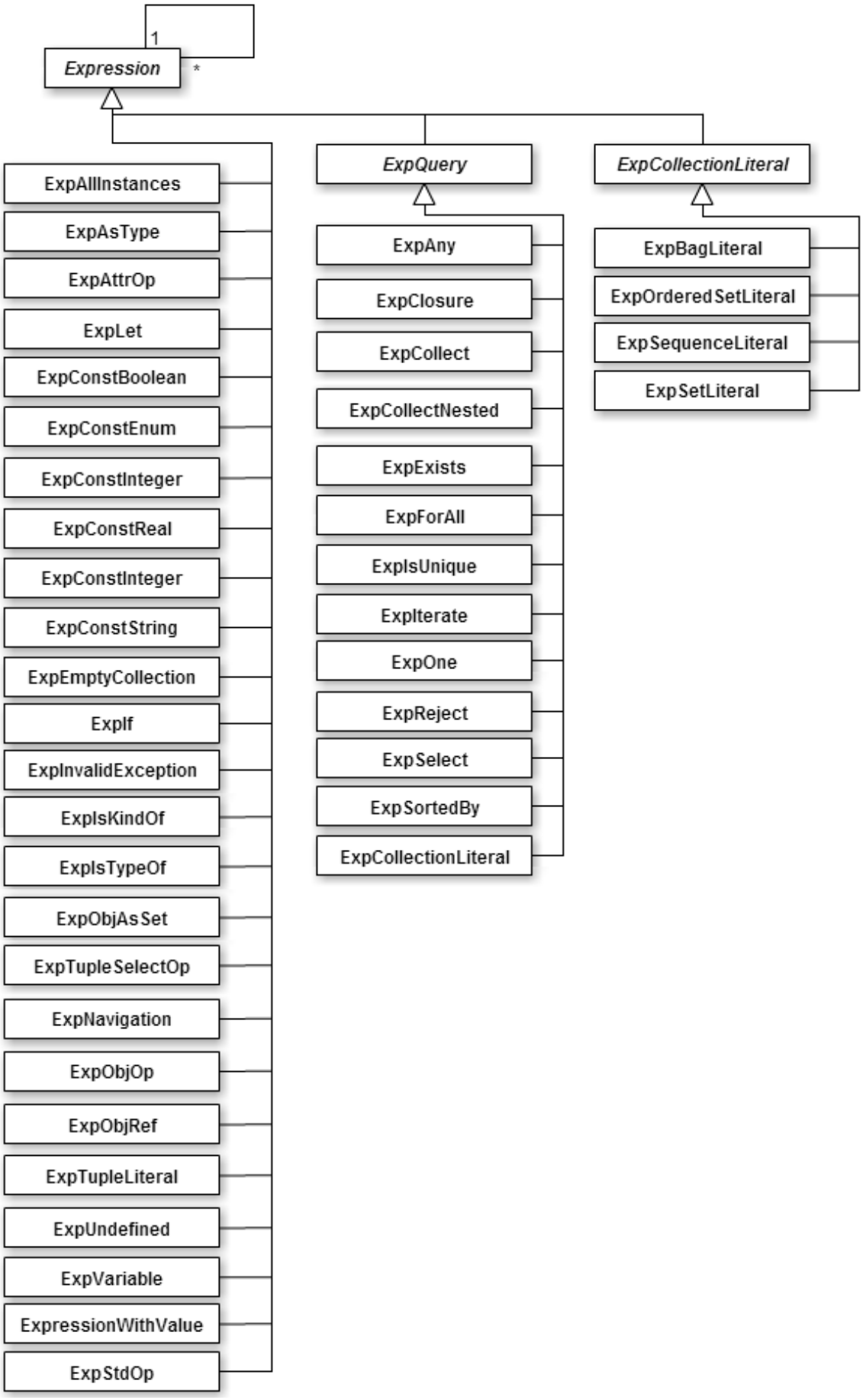
Diagramas del paquete *UML*

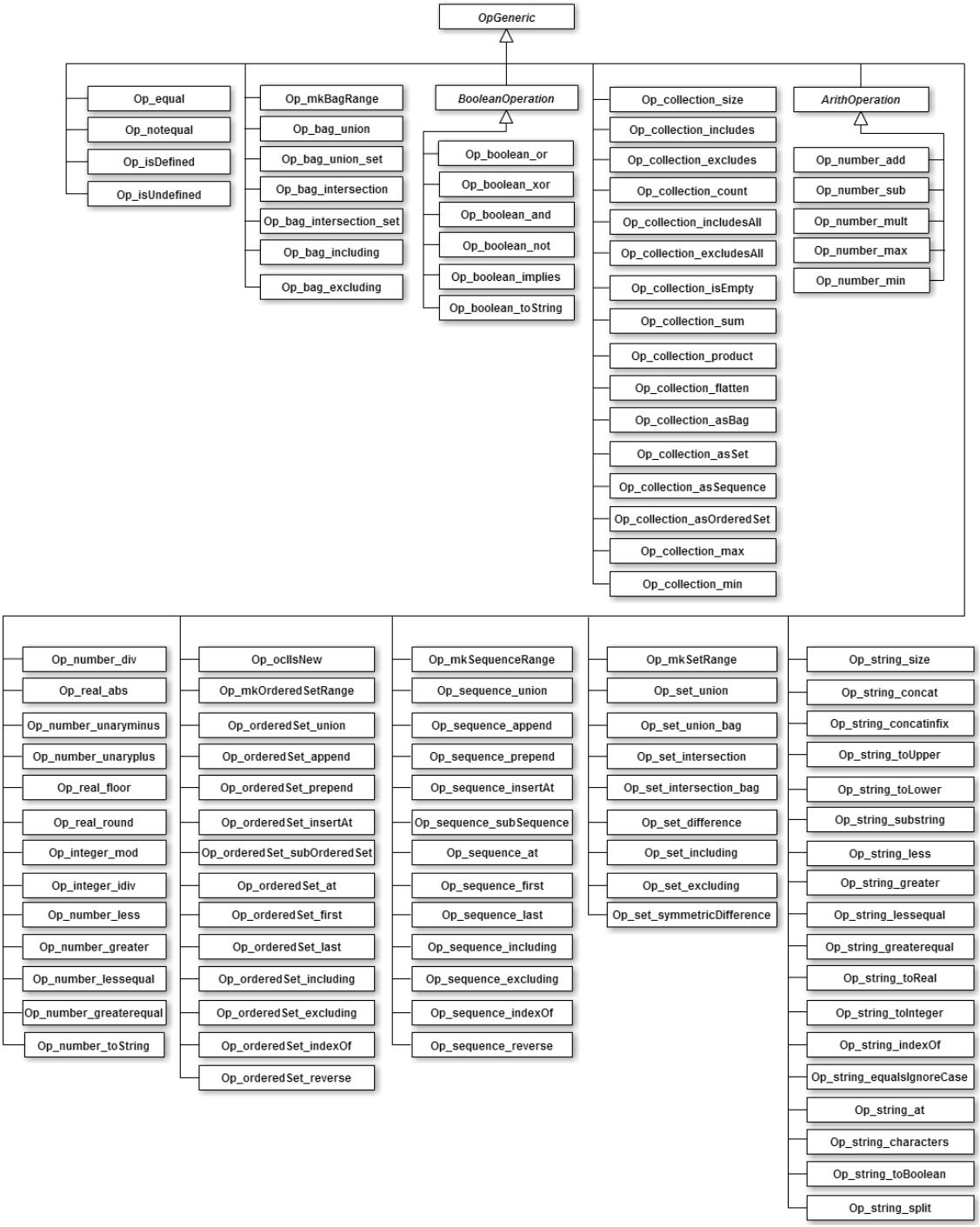


Diagramas del paquete *OCL*






1
Expression
*
ExpAllInstances
ExpAsType
ExpAttrOp
ExpLet
ExpConstBoolean
ExpConstEnum
ExpConstInteger
ExpConstReal
ExpConstInteger
ExpConstString
ExpEmptyCollection
ExpIf
ExpInvalidException
ExpIsKindOf
ExpIsTypeOf
ExpObjAsSet
ExpTupleSelectOp
ExpNavigation
ExpObjOp
ExpObjRef
ExpTupleLiteral
ExpUndefined
ExpVariable
ExpressionWithValue
ExpStdOp
ExpQuery
ExpAny
ExpClosure
ExpCollect
ExpCollectNested
ExpExists
ExpForAll
ExpIsUnique
ExpIterate
ExpOne
ExpReject
ExpSelect
ExpSortedBy
ExpCollectionLiteral
ExpCollectionLiteral
ExpBagLiteral
ExpOrderedSetLiteral
ExpSequenceLiteral
ExpSetLiteral


OpGeneric
Op_equal
Op_notequal
Op_isDefined
Op_isUndefined
Op_mkBagRange
Op_bag_union
Op_bag_union_set
Op_bag_intersection
Op_bag_intersection_set
Op_bag_including
Op_bag_excluding
BooleanOperation
Op_boolean_or
Op_boolean_xor
Op_boolean_and
Op_boolean_not
Op_boolean_implies
Op_boolean_toString
Op_collection_size
Op_collection_includes
Op_collection_excludes
Op_collection_count
Op_collection_includesAll
Op_collection_excludesAll
Op_collection_isEmpty
Op_collection_sum
Op_collection_product
Op_collection_flatten
Op_collection_asBag
Op_collection_asSet
Op_collection_asSequence
Op_collection_asOrderedSet
Op_collection_max
Op_collection_min
ArithOperation
Op_number_add
Op_number_sub
Op_number_mult
Op_number_max
Op_number_min
Op_number_div
Op_real_abs
Op_number_unaryminus
Op_number_unaryplus
Op_real_floor
Op_real_round
Op_integer_mod
Op_integer_idiv
Op_number_less
Op_number_greater
Op_number_lessequal
Op_number_greaterequal
Op_number_toString
Op_oclIsNew
Op_mkOrderedSetRange
Op_orderedSet_union
Op_orderedSet_append
Op_orderedSet_prepend
Op_orderedSet_insertAt
Op_orderedSet_subOrderedSet
Op_orderedSet_at
Op_orderedSet_first
Op_orderedSet_last
Op_orderedSet_including
Op_orderedSet_excluding
Op_orderedSet_indexOf
Op_orderedSet_reverse
Op_mkSequenceRange
Op_sequence_union
Op_sequence_append
Op_sequence_prepend
Op_sequence_insertAt
Op_sequence_subSequence
Op_sequence_at
Op_sequence_first
Op_sequence_last
Op_sequence_including
Op_sequence_excluding
Op_sequence_indexOf
Op_sequence_reverse
Op_mkSetRange
Op_set_union
Op_set_union_bag
Op_set_intersection
Op_set_intersection_bag
Op_set_difference
Op_set_including
Op_set_excluding
Op_set_symmetricDifference
Op_string_size
Op_string_concat
Op_string_concatinfix
Op_string_toUpper
Op_string_toLower
Op_string_substring
Op_string_less
Op_string_greater
Op_string_lessequal
Op_string_greaterequal
Op_string_toReal
Op_string_toInteger
Op_string_indexOf
Op_string_equalsIgnoreCase
Op_string_at
Op_string_characters
Op_string_toBoolean
Op_string_split