



Máster en Cloud Apps
Desarrollo y despliegue de aplicaciones en la nube

Curso académico 2019/2020
Trabajo de Fin de Máster

HalteroCMS: Haltero Championship Management System

Autores: Jaime Hernández y Natalia Roales
Tutor: Luis Fernández Muñoz



Resumen

El objetivo principal de este proyecto es el desarrollo de una aplicación para la creación y gestión integral de una competición de halterofilia, según la normativa de la Federación Española de Halterofilia (FEH), dirigido mediante el proceso de desarrollo RUP.

A su vez, con el fin de poner en práctica lo aprendido en cada uno de los módulos del curso, utilizaremos las tecnologías que más se adecúen para cumplir con cada una de las disciplinas enmarcadas dentro de los Procesos Unificados.

Demostraremos que es posible combinar un framework de trabajo robusto y con años de historia con las tecnologías más punteras, y que no siempre es necesario emplear las últimas metodologías de moda.

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit
<http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Tabla de contenido

| | |
|---|-----------|
| 1. Introducción..... | 4 |
| 1.1 Objetivos..... | 4 |
| 1.2 ¿Qué es la halterofilia?..... | 4 |
| 2. Proceso unificado de desarrollo (RUP)..... | 4 |
| 3. Modelo de dominio | 6 |
| 3.1. Modelando los estados del dominio..... | 7 |
| 3.1.1. Estados de una competición | 7 |
| 3.1.2. Estados de una ronda..... | 8 |
| 3.1.3. Estados de una iteración..... | 8 |
| 3.1.4. Estados de la fase de clasificación | 8 |
| 4. Disciplina de requisitos..... | 14 |
| 4.1 Actores y casos de uso..... | 15 |
| 4.2 Priorizar casos de uso | 21 |
| 4.3 Especificación de casos de uso | 23 |
| 4.3.1. Especificación de casos de uso del secretario de la competición | 23 |
| 4.3.2. Especificación de casos de uso del secretario de la organización | 29 |
| 4.4 Prototipado interfaz de usuario | 41 |
| 5. Disciplina de análisis | 44 |
| 5.1 Análisis de la arquitectura | 45 |
| 5.2 Análisis de casos de uso..... | 46 |
| 6. Disciplina de diseño | 47 |
| 6.1. Diseñar la arquitectura | 48 |
| 6.2. Diseñar casos de uso..... | 49 |
| 6.2.1. Diseño del caso de uso 'introduceWeighinData' | 51 |
| 6.2.2. Diseño del caso de uso 'showBatchInPlay' | 52 |
| 6.3. Diseñar clases | 52 |
| 7. Disciplina de implementación | 53 |
| 7.1. Casos de uso implementados..... | 54 |
| 7.2. Test-First Development..... | 54 |
| 7.3. Arquitectura por capas vs. arquitectura hexagonal | 55 |
| 8. Disciplina de pruebas | 56 |
| Conclusiones y trabajo futuro..... | 58 |
| Bibliografía | 60 |

1. Introducción

El objetivo principal de este proyecto es el desarrollo de una aplicación para la creación y gestión integral de una competición de halterofilia, según la normativa de la Federación Española de Halterofilia (FEH), mediante el proceso de desarrollo RUP.

En los últimos años el deporte de la halterofilia ha crecido exponencialmente, debido a esto, es necesario que la gestión de competiciones se automatice y se facilite todo lo posible tanto a jueces, espectadores, levantadores, etc. para el seguimiento de la competición.

1.1 Objetivos

El objetivo principal del proyecto es el desarrollo de una aplicación web para la gestión integral de una competición de halterofilia. Para llevar a cabo este proyecto se va a utilizar el proceso unificado de desarrollo, Rational Unified Process (RUP).

Los sub-objetivos del proyecto son:

- Profundizar en el proceso de desarrollo RUP aplicado a un proyecto real.
- Realizar un proyecto focalizado en:
 - Afianzar el correcto requisitado, análisis y diseño.
 - Aplicación correcta de diversos patrones.
- Aplicación de un modelo de desarrollo basado en *Gitflow*.
- Implementar un entorno de integración y despliegue continuos mediante el uso de:
 - Git como gestor de versiones.
 - GitHub como repositorio (tanto de código como de artefactos).

1.2 ¿Qué es la halterofilia?

La halterofilia es un deporte que consiste en el levantamiento del máximo peso posible de una barra a cuyos extremos se fijan varios discos de distinto peso. Existen dos modalidades de competición arrancada (*snatch*) y dos tiempos (*clean&jerk*).

Se divide en categoría masculina y femenina. A su vez, ambas se subdividen en distintas categorías de acuerdo a la masa corporal y edad del atleta.

2. Proceso unificado de desarrollo (RUP)

El proceso unificado de desarrollo o RUP (Rational Unified Process) es un proceso iterativo de desarrollo de software basado en componentes interconectados y bien definidos vía sus interfaces. RUP es un marco de trabajo que puede utilizarse en todo tipo de desarrollos.

Las tres principales características de RUP son las siguientes son (1) dirigido por casos de uso, (2) centrado en la arquitectura y (3) proceso iterativo incremental.

(1) Los casos de uso son la piedra angular para establecer el comportamiento deseado del sistema y cómo comunicar este comportamiento entre los diferentes implicados en el sistema. Nos sirven para capturar los requisitos funcionales correctamente representados, etc.

(2) La arquitectura es usada como artefacto primordial para la conceptualización, construcción, gestión y evolución del sistema en desarrollo

(3) Iterativo implica que el proceso involucra un flujo de entregas ejecutables e incremental ya que con cada nueva entrega se proporciona un incremento respecto a la anterior

Las 5 disciplinas o flujos de trabajo de RUP son Requisitos, Análisis, Diseño, Implementación y Pruebas, y cada una se divide en 4 fases: Inicio, Elaboración, Construcción y Transición. El esfuerzo dedicado a la realización de las actividades de las diferentes disciplinas depende del momento del proyecto en el que estemos.

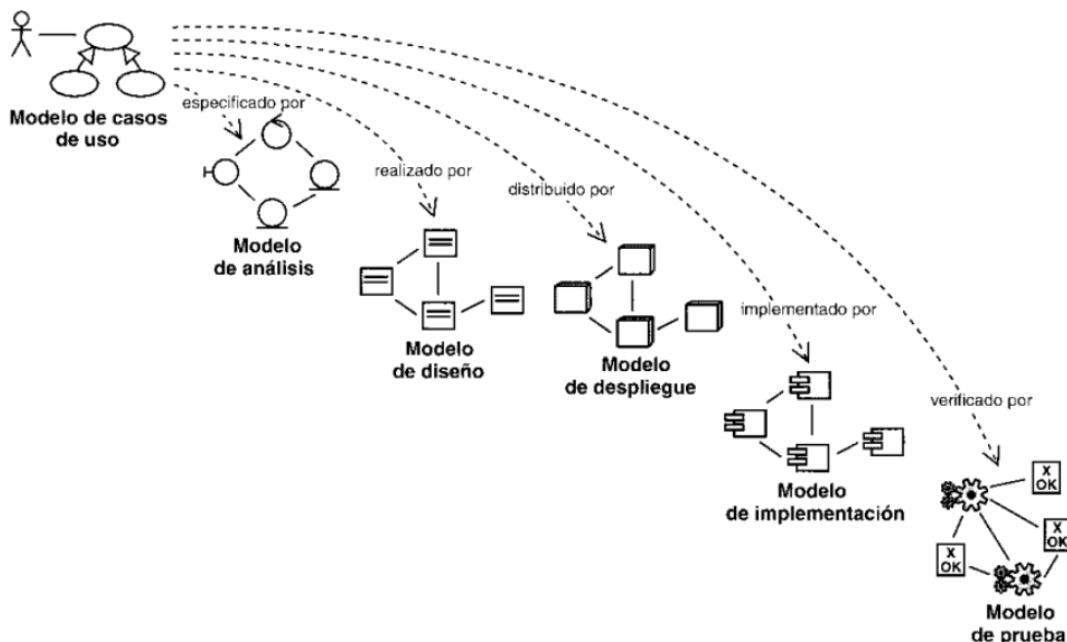


Fig. 1 – Disciplinas de RUP.

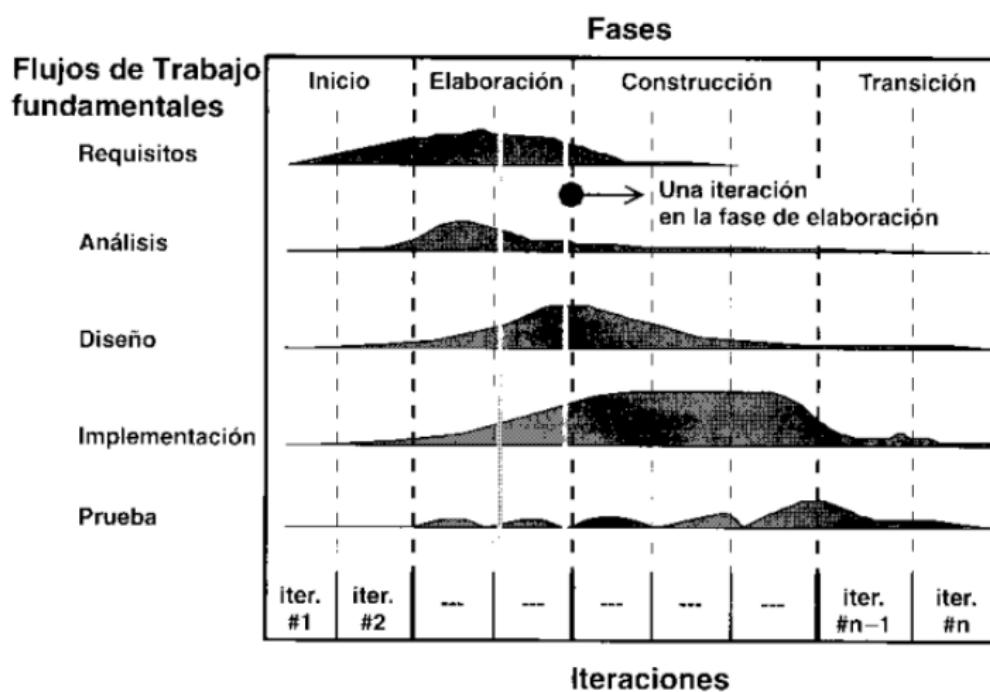


Fig. 2 – Fases de las disciplinas de RUP.

3. Modelo de dominio

El modelo del dominio describe los conceptos más importantes del contexto del sistema como son:

- Objetos de negocio.
- Objetos del mundo real y conceptos que un sistema necesita hacer un seguimiento.
- Eventos que suceden o que sucederán en el sistema.

El modelo del dominio también es importante, ya que nos ayuda a tener y consensuar un vocabulario común.

Para obtener el modelo del dominio se ha tenido en cuenta las necesidades actuales de la Federación Madrileña de Halterofilia a la hora de realizar la gestión integral de una competición de halterofilia. Para ello se ha hablado con distintas personas de la federación para conocer los requisitos necesarios para cada uno de ellos.

En la Fig. 3 se puede observar los objetos del dominio, sus relaciones y el vocabulario consensuado por el cliente.

El modelo de dominio nos puede ayudar también a tener una mejor comprensión de la organización destino. Un ejemplo de esto podría ser una muestra real de como se distribuyen las categorías oficiales en la halterofilia, tal y como se puede ver en la Fig. 4.

3.1. Modelando los estados del dominio

Otra forma muy interesante de modelar la realidad es mediante los estados que conforman los procesos involucrados en el dominio. Para ello se han realizado una serie de diagramas de estados, los cuales modelan la realidad correspondiente a los siguientes procesos:

- La competición de principio a fin.
- Cada ronda.
- Cada iteración dentro de una ronda.
- La fase de clasificación.

3.1.1. Estados de una competición

En la Fig. 5 se muestra el diagrama de estados para una competición.

3.1.2. Estados de una ronda

En la Fig. 6 se muestra el diagrama de estados para una ronda.

3.1.3. Estados de una iteración

En la Fig. 7 se muestra el diagrama de estados para una iteración dentro de una ronda.

3.1.4. Estados de la fase de clasificación

En la Fig. 8 se muestra el diagrama de estados para la fase de clasificación.

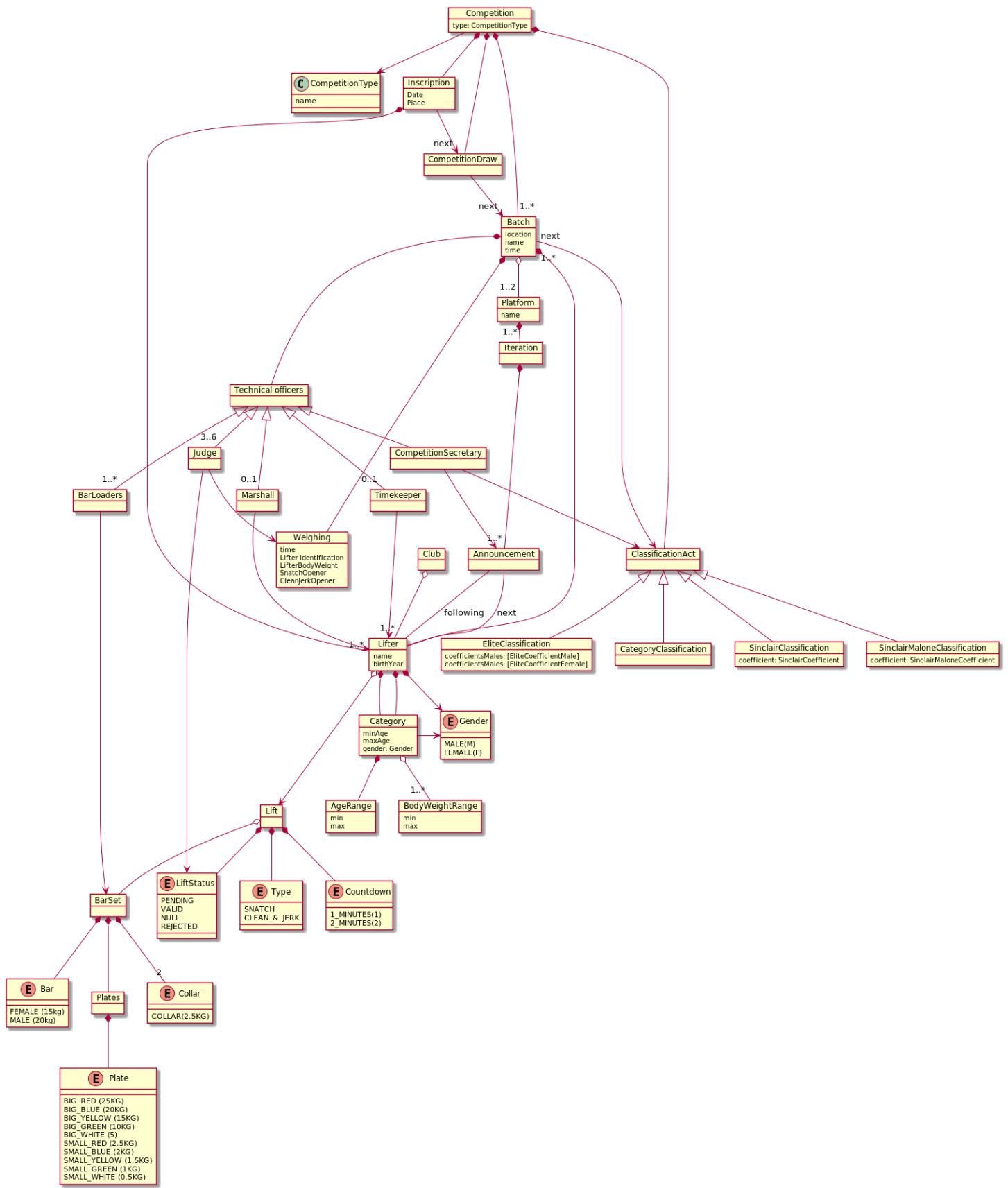


Figura 1. Modelo del dominio

Fig. 3 – Modelo del dominio de la competición.

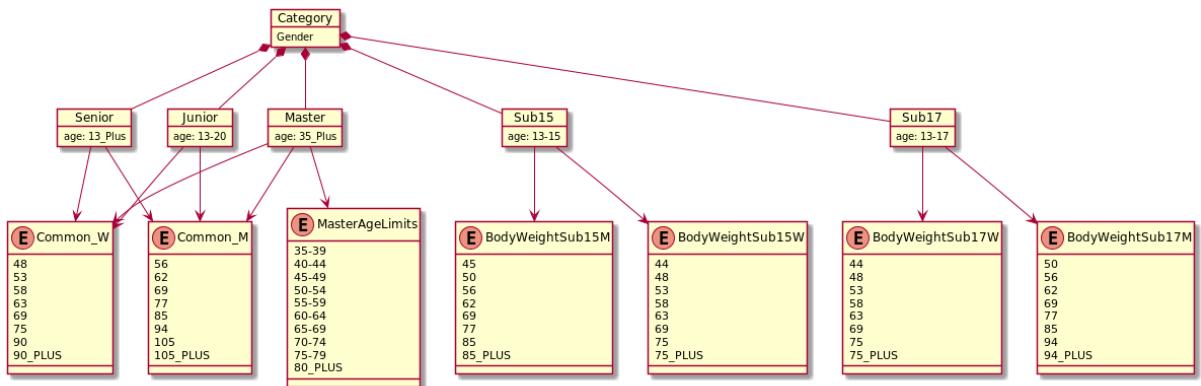


Figura 2. Caso concreto de categorías de una competición

Fig. 4 – Modelado de las categorías reales en halterofilia.

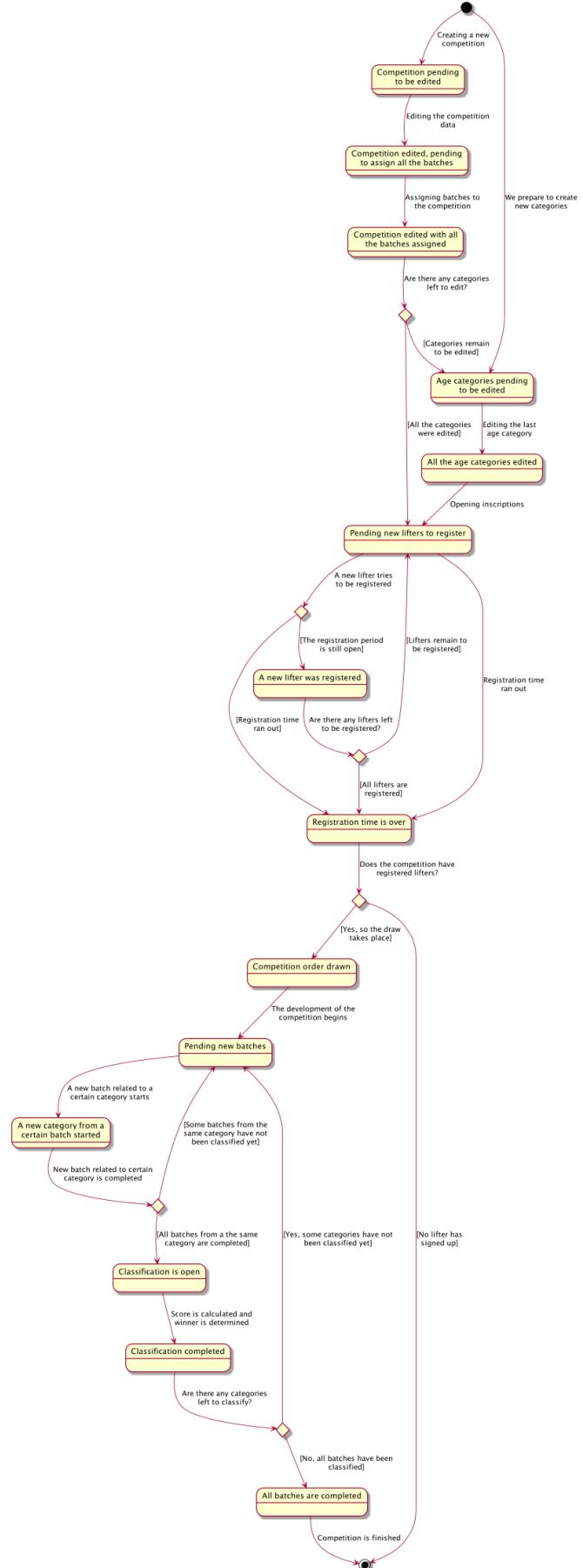


Fig. 5 – Diagrama de estados de la competición.

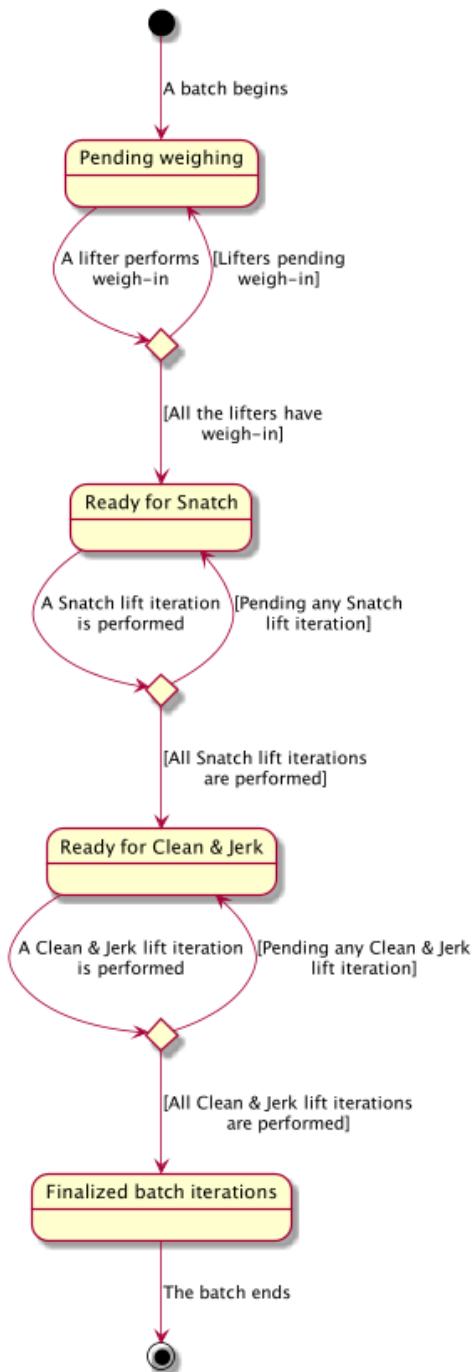


Fig. 6 – Diagrama de estados para una ronda.

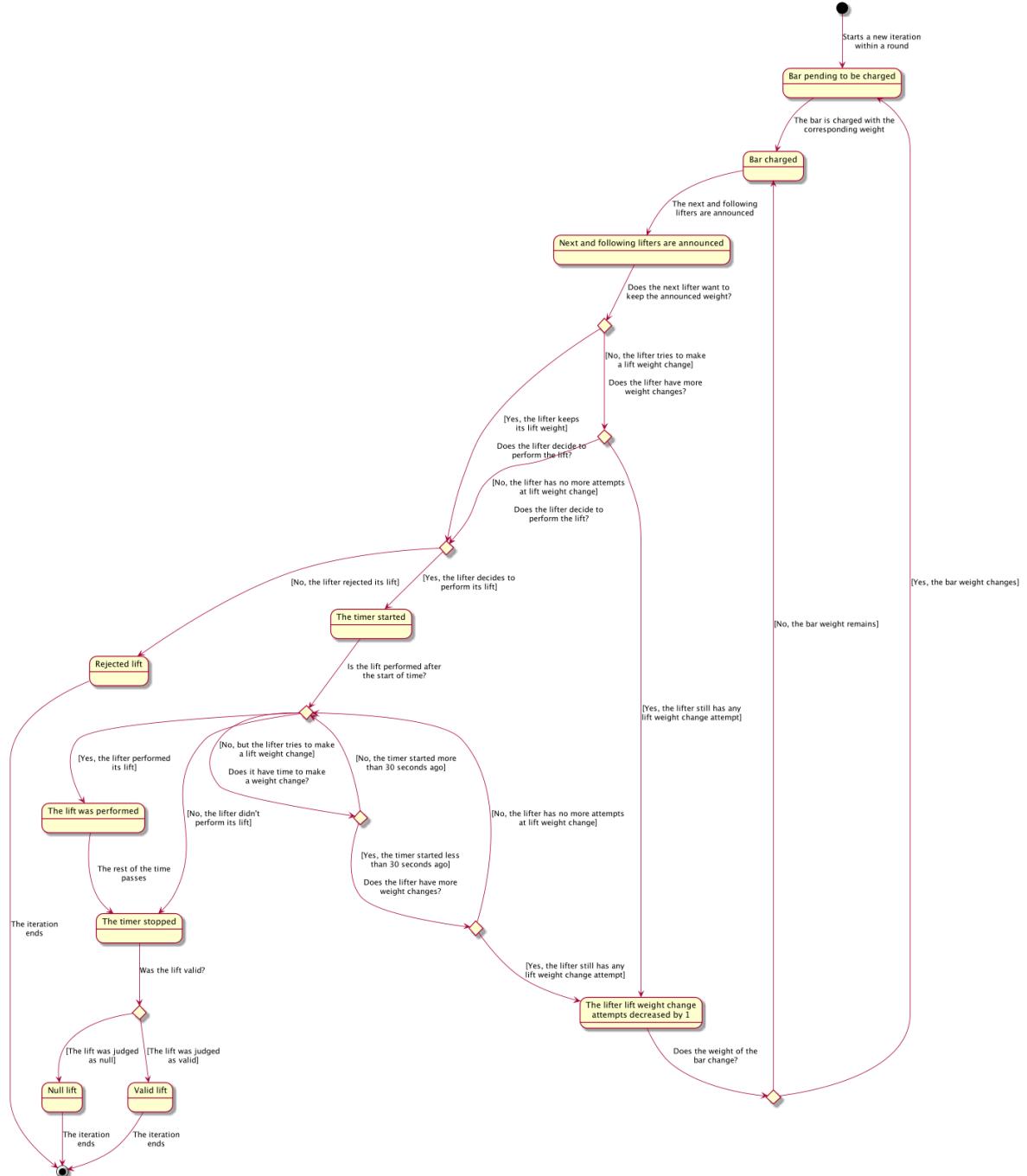


Fig. 7 – Diagrama de estados para una iteración.

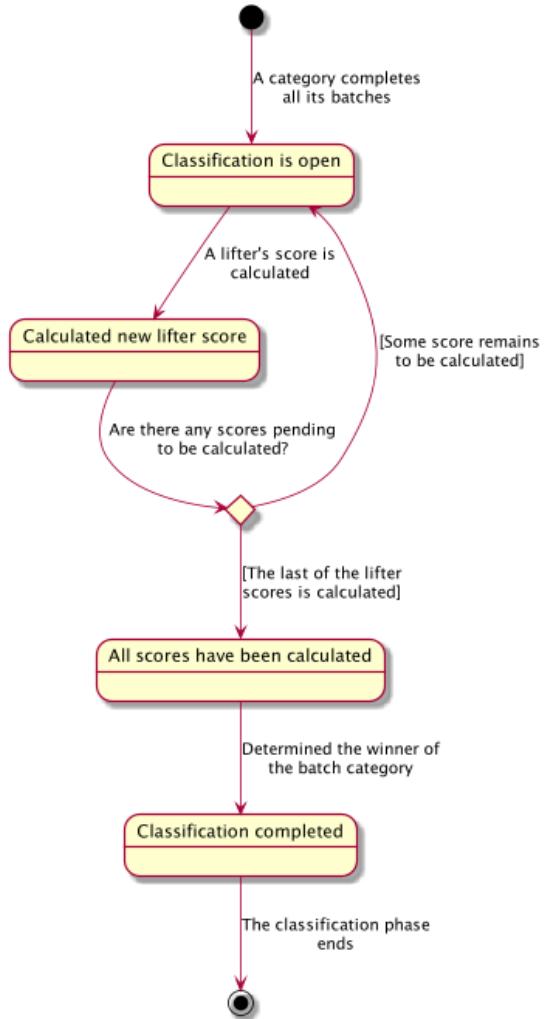


Fig. 8 – Diagrama de estados para la clasificación.

4. Disciplina de requisitos

La disciplina de requisitos es el flujo de trabajo (realización de casos de uso que incluye roles, actividades y artefactos) cuyo principal propósito es dirigir el desarrollo hacia el sistema correcto describiendo los requisitos del sistema de tal manera que se alcance un contrato entre cliente, usuarios y desarrolladores sobre lo que el sistema debe hacer.

Objetivos:

- Establecer y mantener el acuerdo entre clientes y otros implicados sobre lo que el sistema debería hacer.
- Proveer a los desarrolladores una mejor comprensión de los requisitos del sistema.
- Definir los límites del sistema.
- Proveer la base para planificar los contenidos técnicos de cada iteración.

Las actividades recomendadas por RUP en esta disciplina son las siguientes:

1. Encontrar actores y casos de uso.
2. Priorizar casos de uso.
3. Especificación casos de uso.
4. Estructurar el modelo de casos de uso.
5. Prototipado de la interfaz de usuario.

4.1 Actores y casos de uso

- Un actor especifica un rol que adopta una entidad externa cuando interactúa con el sistema directamente (no en respuesta). Si se necesita modelar cosas que suceden en el sistema en un tiempo específico pero que parece que no lo lanza ningún actor, se puede introducir el actor tiempo.
- Un caso de uso es una especificación de secuencias de acciones, incluyendo posibles variaciones, que el sistema puede realizar y que dan un resultado observable de interés a un actor particular.

En la Fig. 9 podemos ver los actores que se han identificado, mientras que de la Fig. 10 a la Fig. 14 se representan los casos de uso identificados y los actores que los realizan.

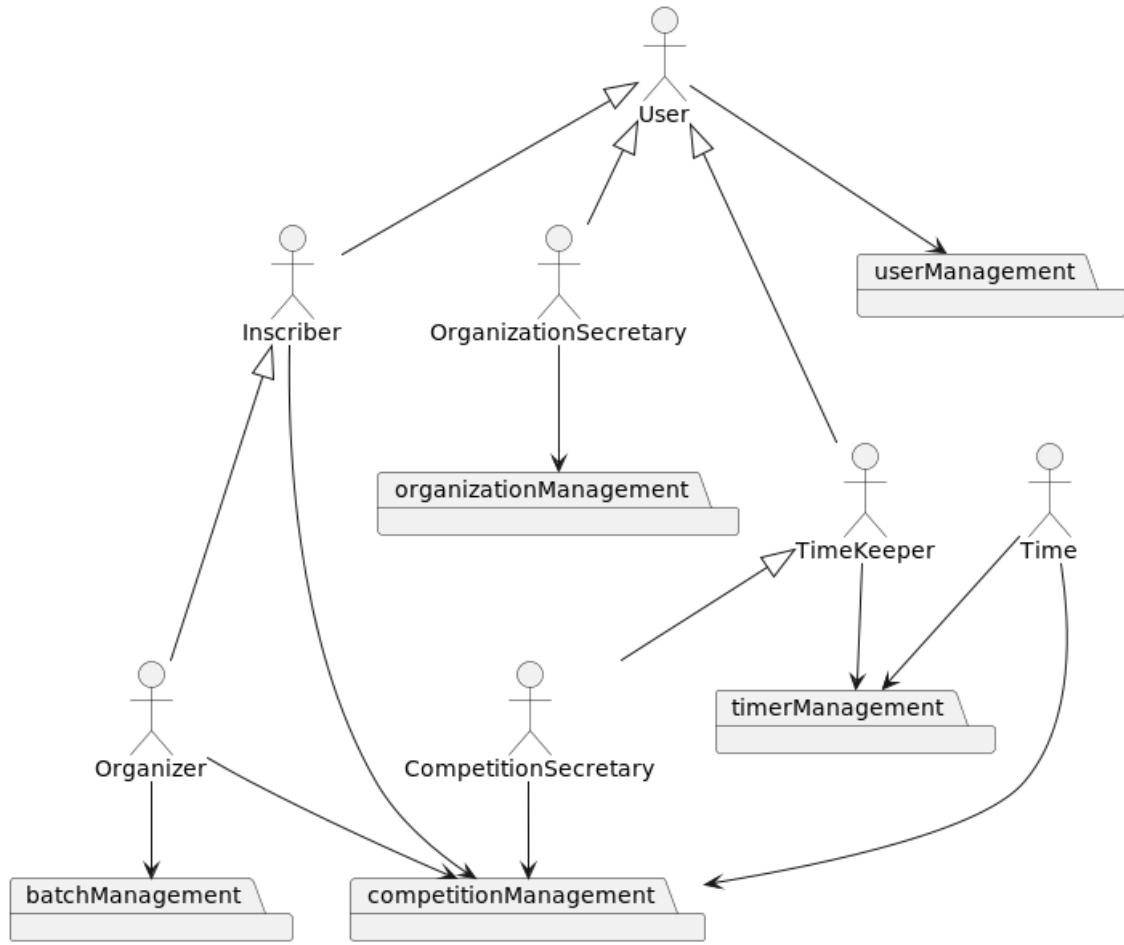


Figura 3. Actores del sistema

Fig. 9 – Actores identificados.

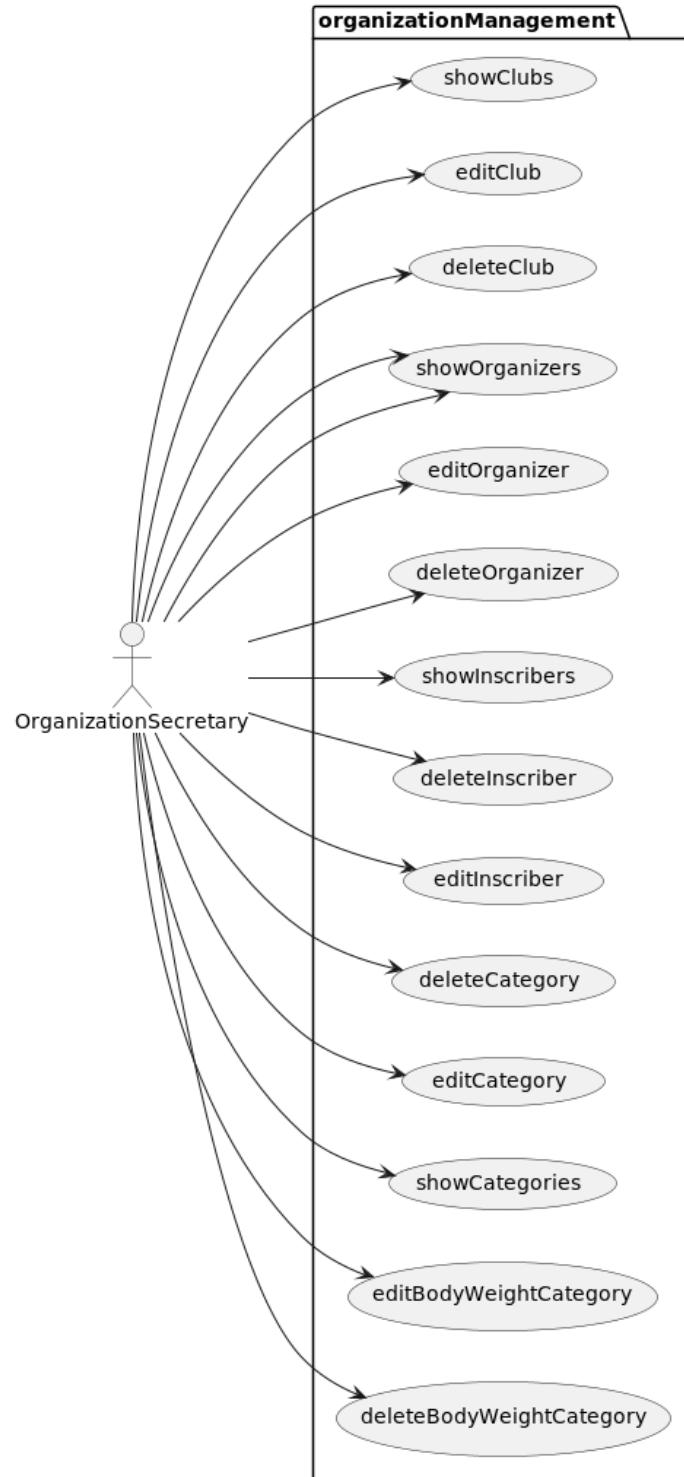


Figura 4. Casos de uso relacionados con el actor secretario de competición

Fig. 10 – Casos de uso del secretario de la organización.

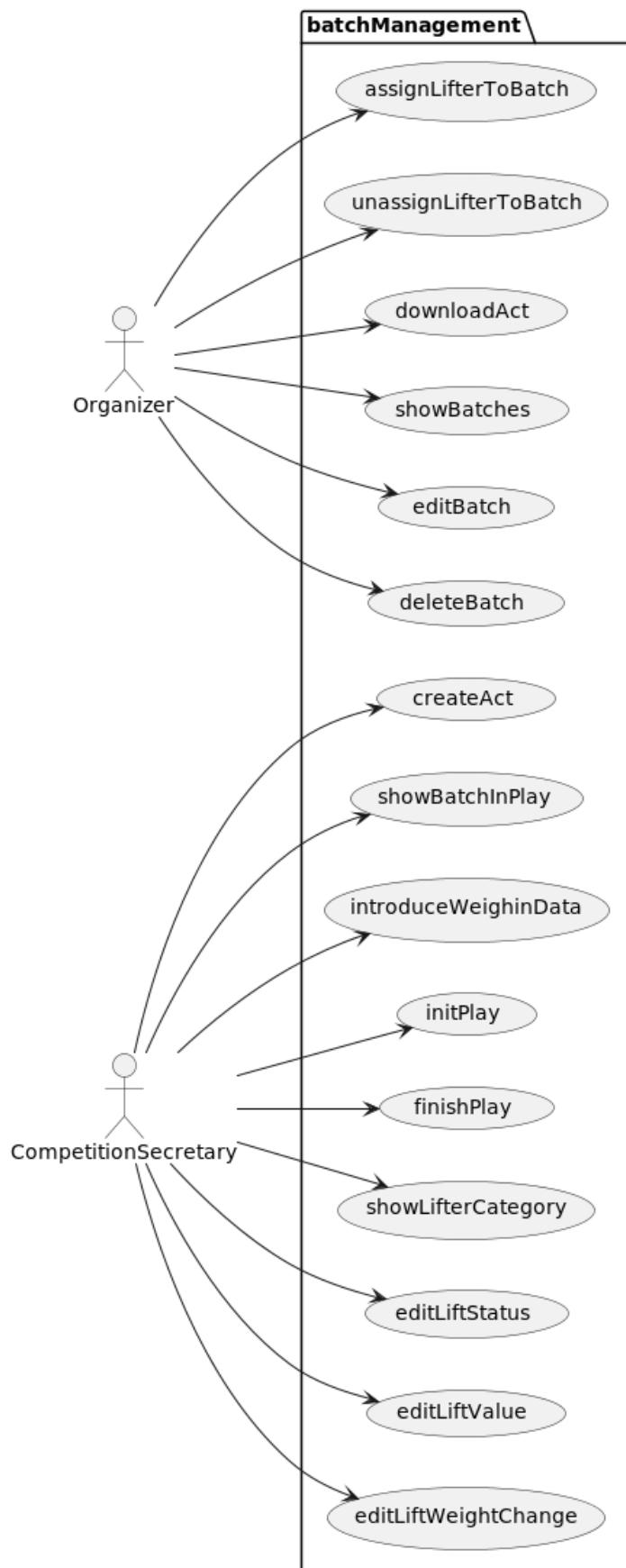


Fig. 11 – Casos de uso del organizador y del secretario de la competición.

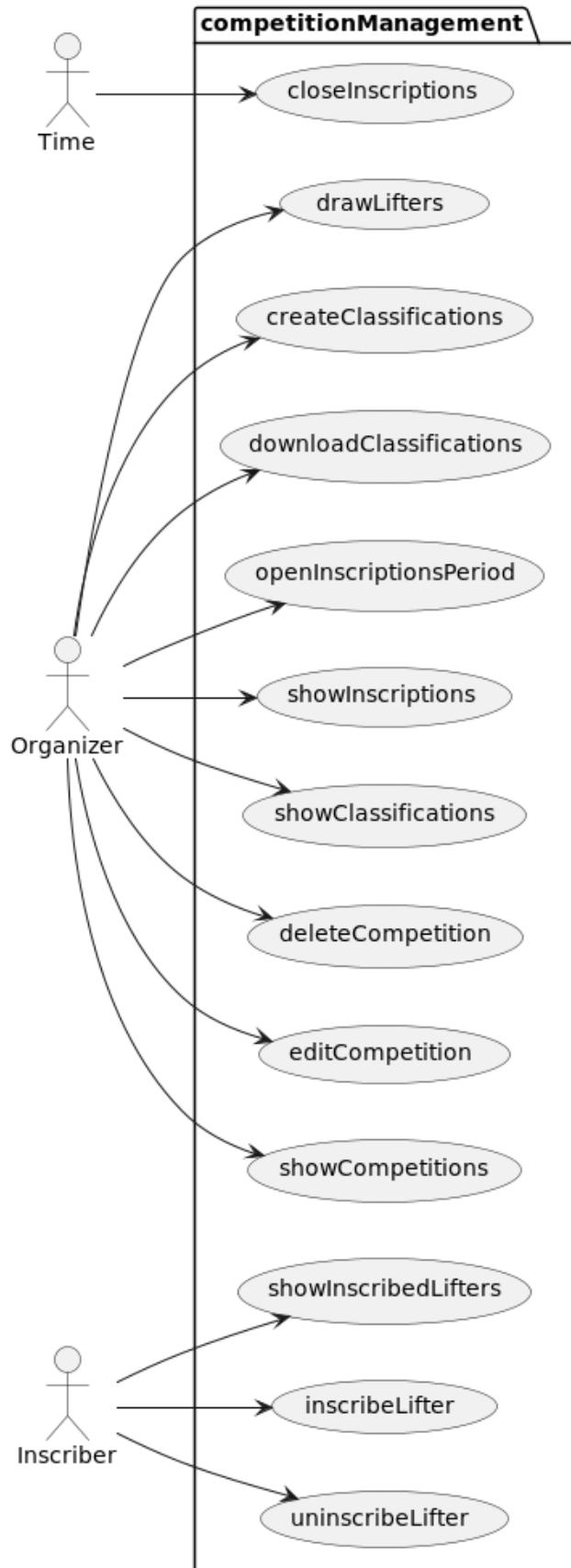


Fig. 12 – Casos de uso del organizador, del inscriptor y del tiempo.

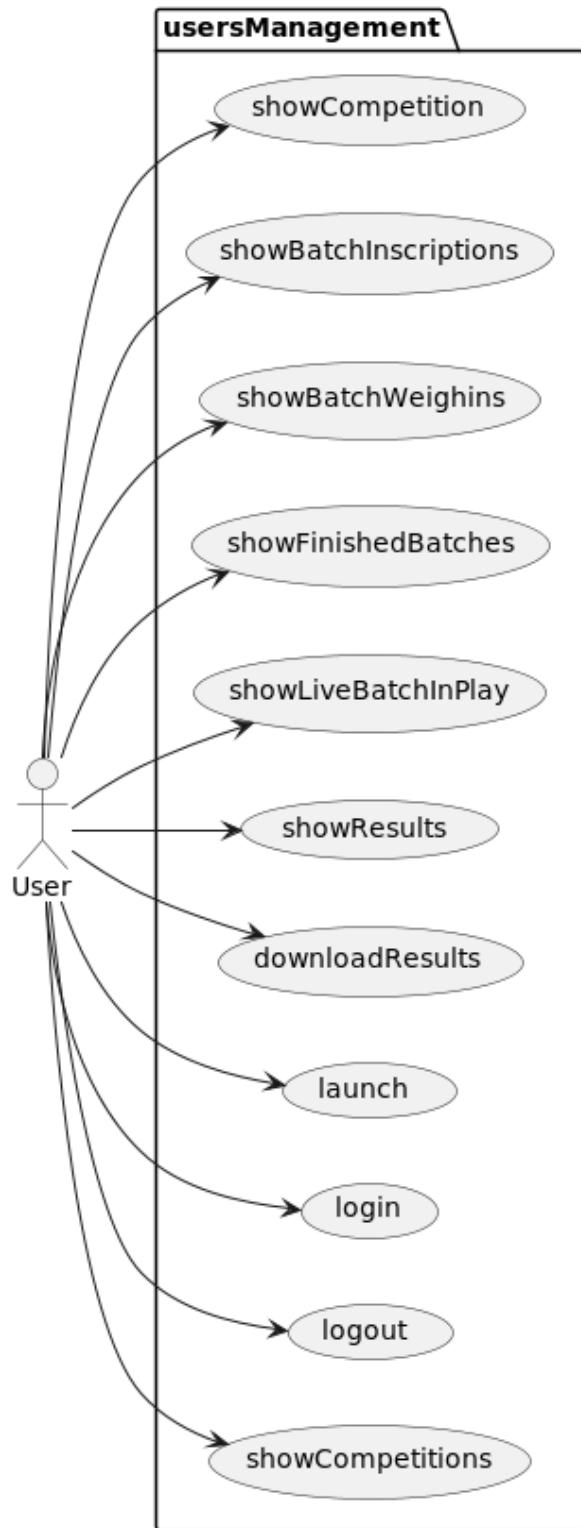


Figura 7. Casos de uso relacionados con el actor

Fig. 13 – Casos de uso del espectador.

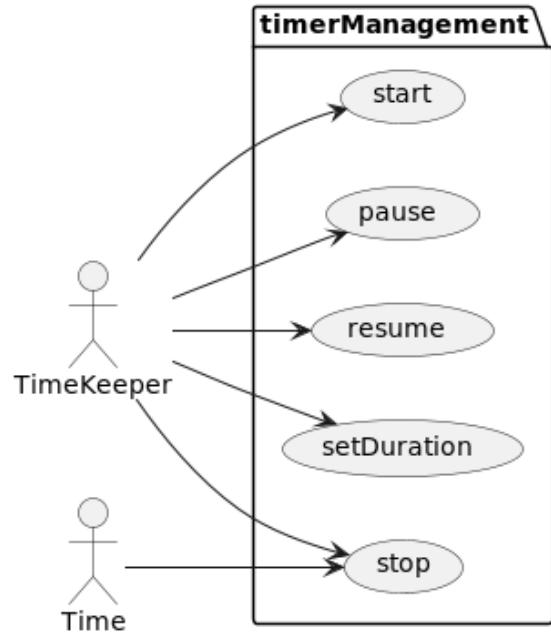


Fig. 14 – Casos de uso relacionados con la gestión del tiempo.

4.2 Priorizar casos de uso

Hay que determinar qué casos de uso serán desarrollados (analizados, diseñados, implementados y probados) en las primeras iteraciones y cuáles se pueden desarrollar en sucesivas iteraciones teniendo en cuenta factores como puedan ser aspectos técnicos, aspectos económicos, aspectos de negocio o cualquier tipo de riesgo.

En nuestro caso hemos decidido que los casos de uso que nos interesaba terminar lo antes posible debido a factores de negocio y al tener una mayor complejidad en su lógica son los relacionados con "jugar la tanda". A partir de ese primer hito, en sucesivas iteraciones se van a ir completando toda la gestión de tandas, competiciones, etc. tal y como se muestran de la Fig. 15 a la Fig. 22.

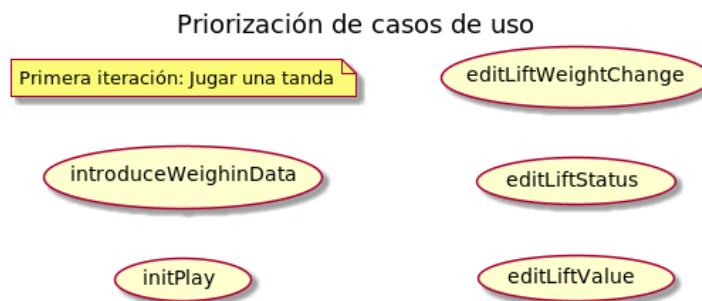


Fig. 15 – Priorización de casos de uso: Primera iteración.

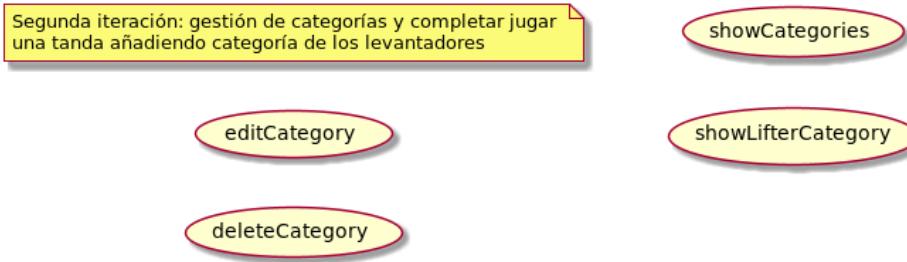


Fig. 16 - Priorización de casos de uso: Segunda iteración.

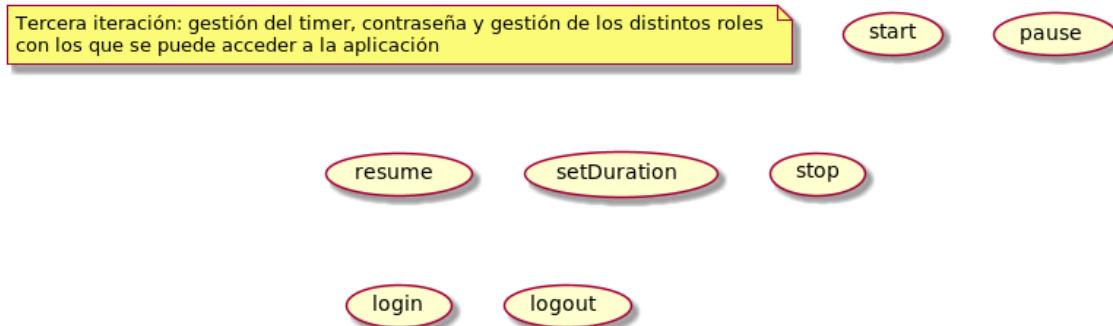


Fig. 17 - Priorización de casos de uso: Tercera iteración.

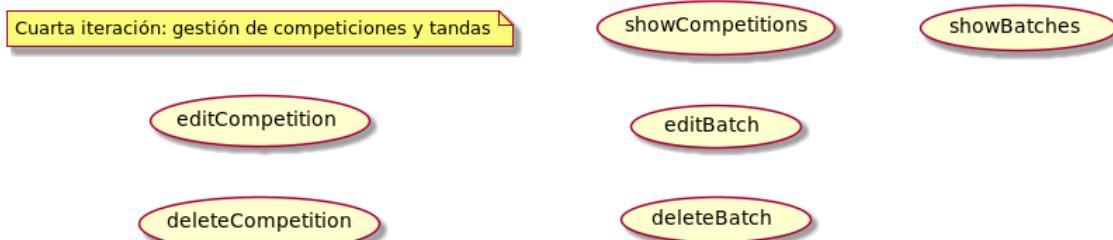


Fig. 18 - Priorización de casos de uso: Cuarta iteración.



Fig. 19 - Priorización de casos de uso: Quinta iteración.

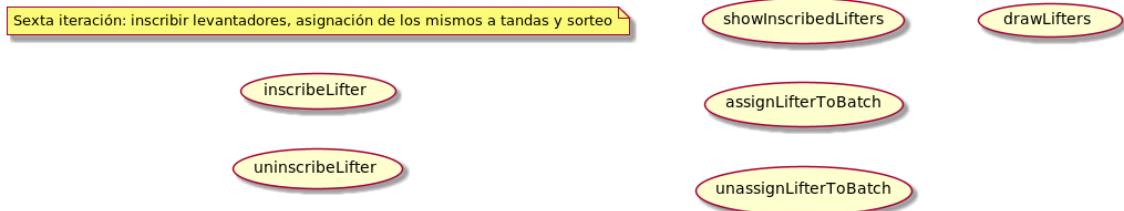


Fig. 20 - Priorización de casos de uso: Sexta iteración.

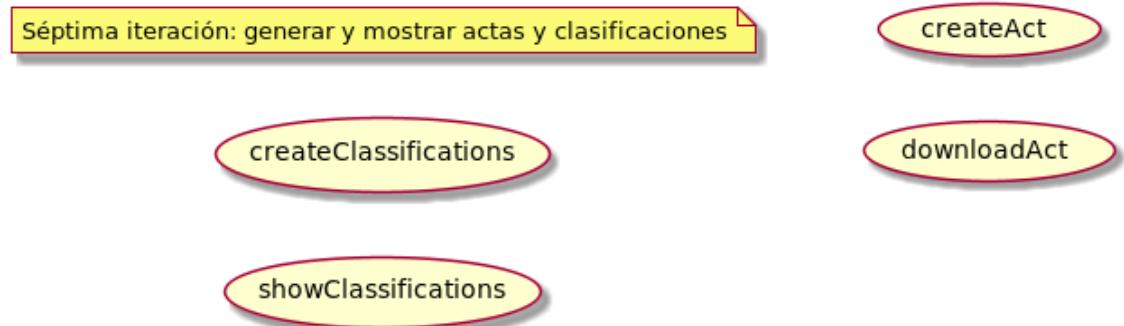


Fig. 21 - Priorización de casos de uso: Séptima iteración.

Octava iteración: resto de casos de uso

Fig. 22 - Priorización de casos de uso: Octava iteración.

4.3 Especificación de casos de uso

Durante esta actividad, nos hemos centrado en los casos de uso correspondientes al secretario de la competición y al secretario de la organización. Mediante la especificación de los casos de uso seleccionados, definiremos de manera minuciosa cada uno de los estados por los que pasa el caso de uso a especificar, y añadiremos los detalles sobre todas las transiciones de un estado a otro.

A nivel funcional, en lo que a un caso de uso se refiere, no nos dejaremos ningún detalle fuera del diagrama de su especificación, y seremos capaces de saber cómo se comporta el sistema esté en el punto en el que esté.

4.3.1. Especificación de casos de uso del secretario de la competición

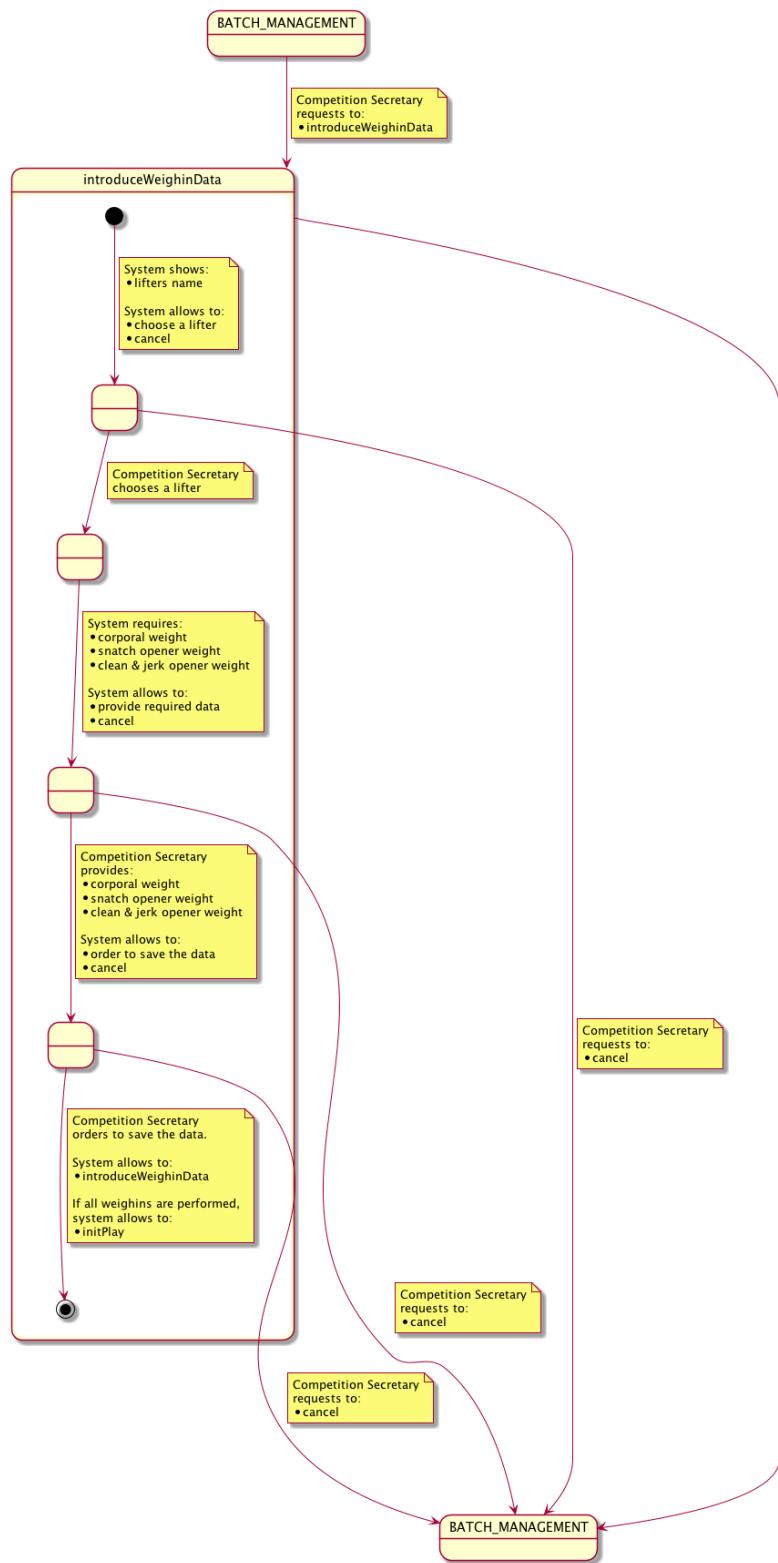
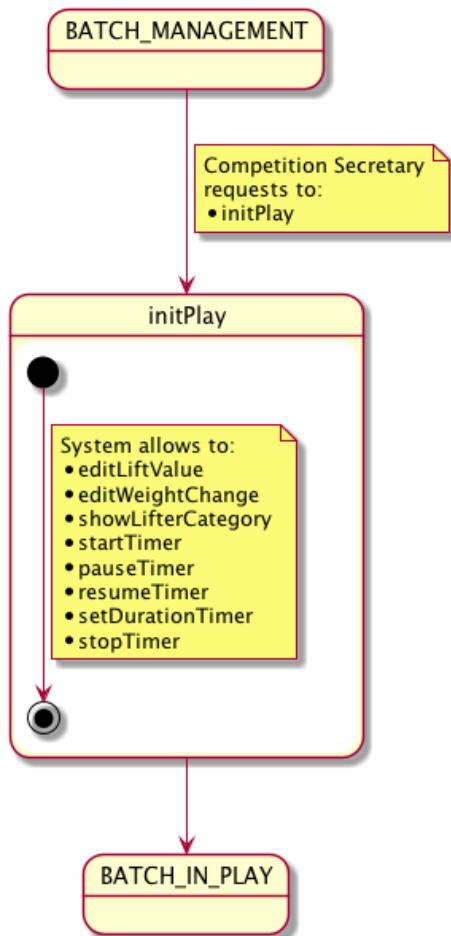
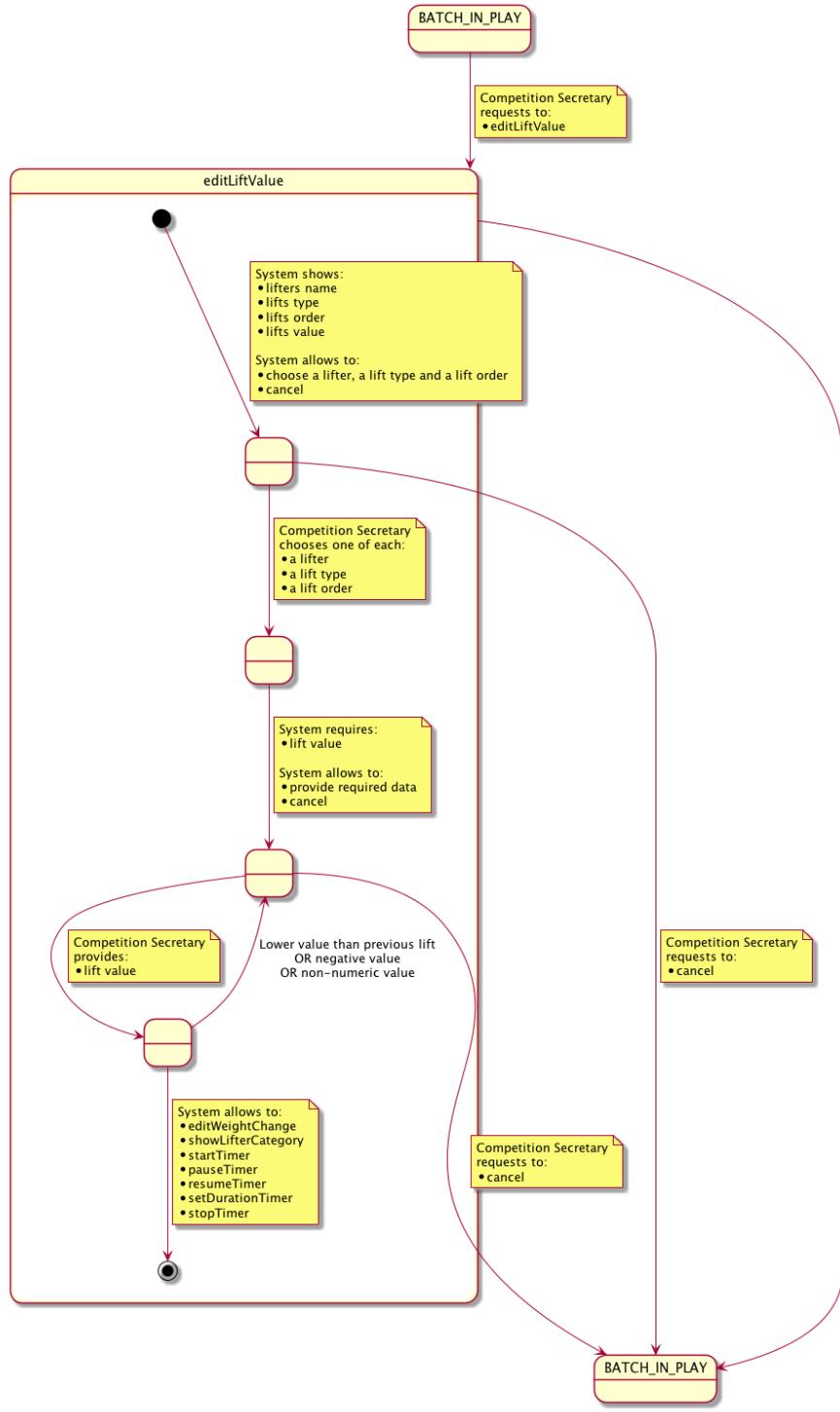


Fig. 23 – Caso de uso: Introducir datos de un pesaje.



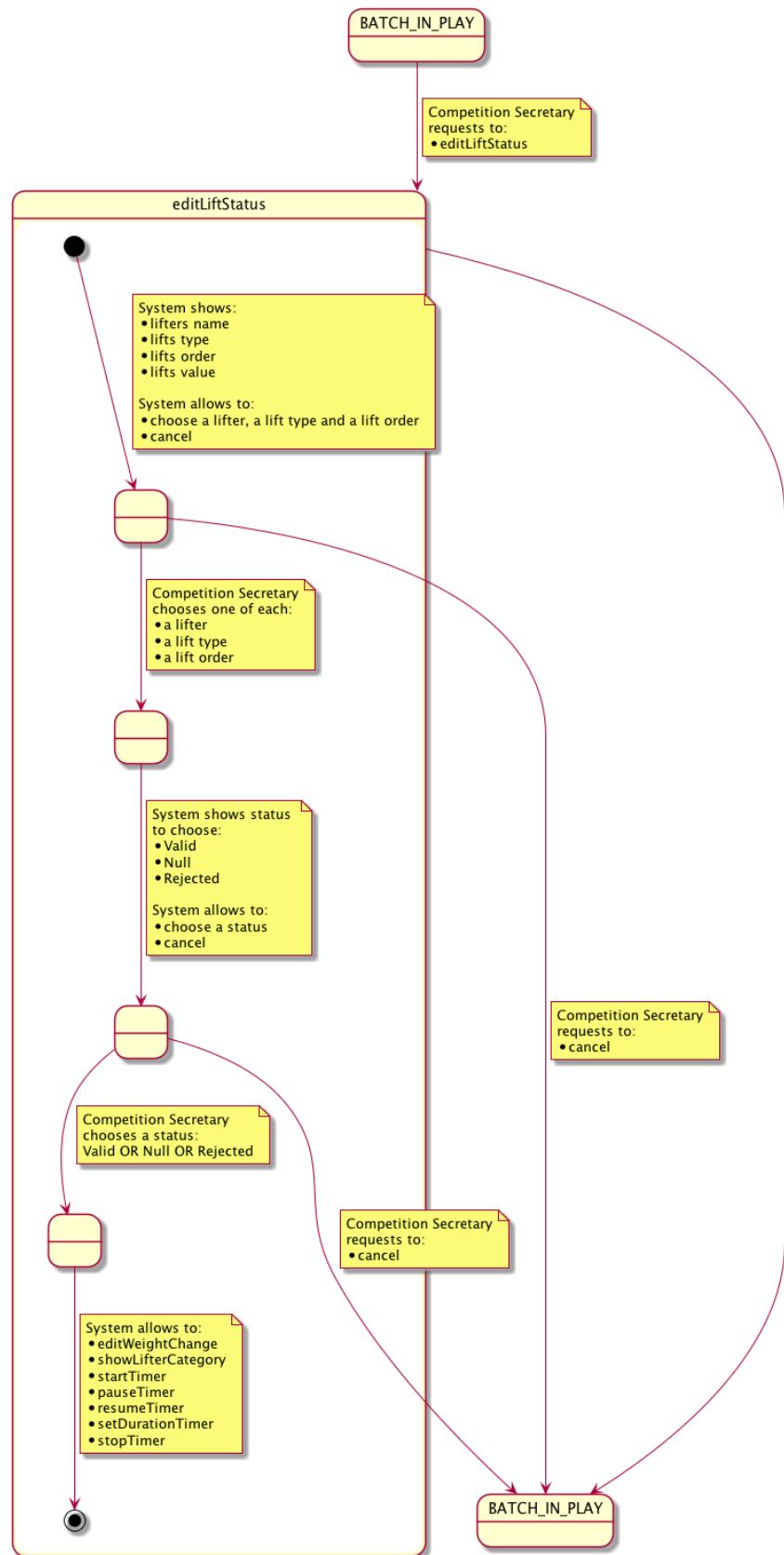
Especificación del caso de uso "initPlay"

Fig. 24 – Caso de uso: Iniciar una ronda.



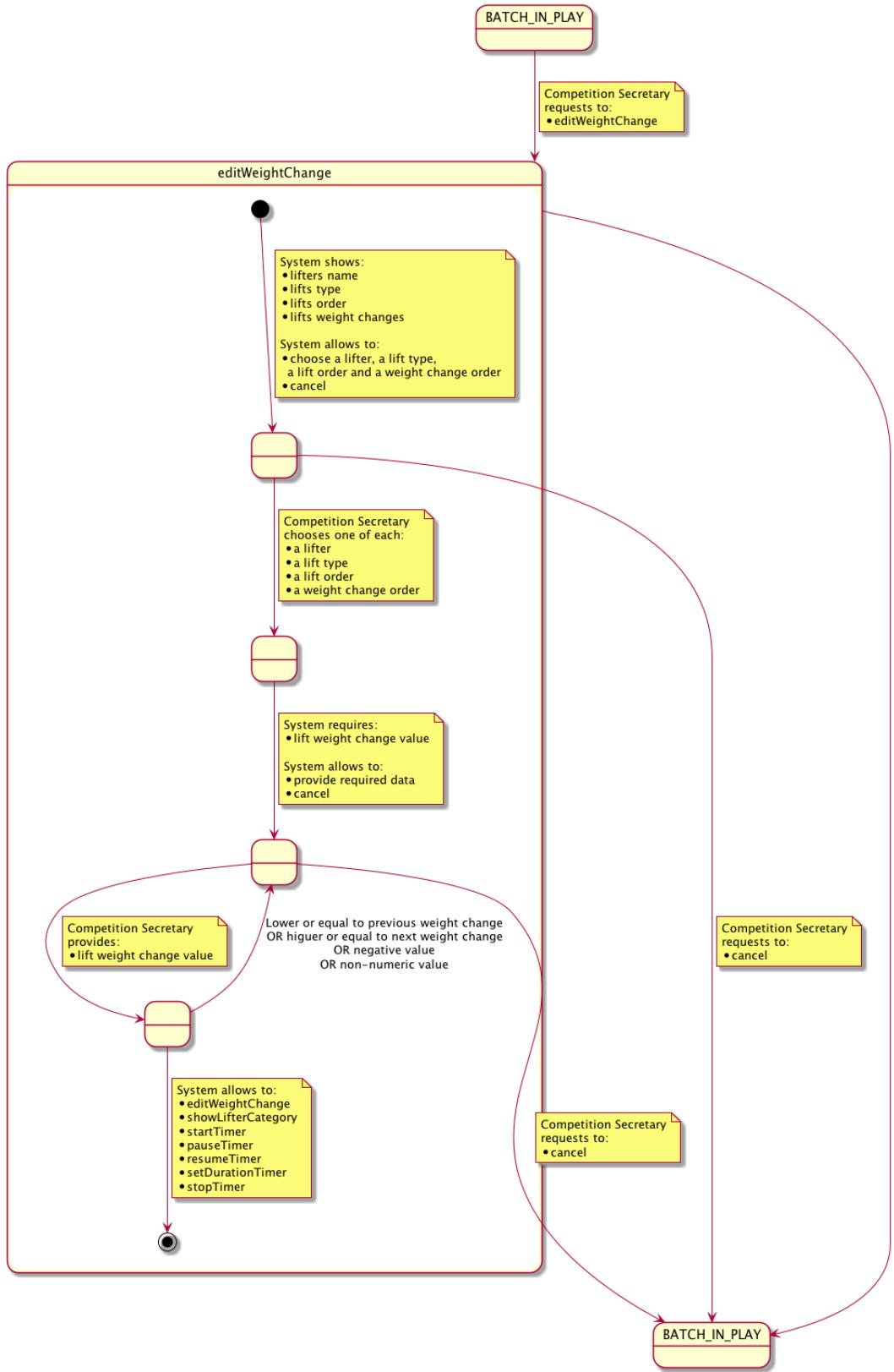
Especificación del caso de uso "editLiftValue"

Fig. 25 – Caso de uso: Añadir o actualizar el peso de un levantamiento.



Especificación del caso de uso "editLiftStatus"

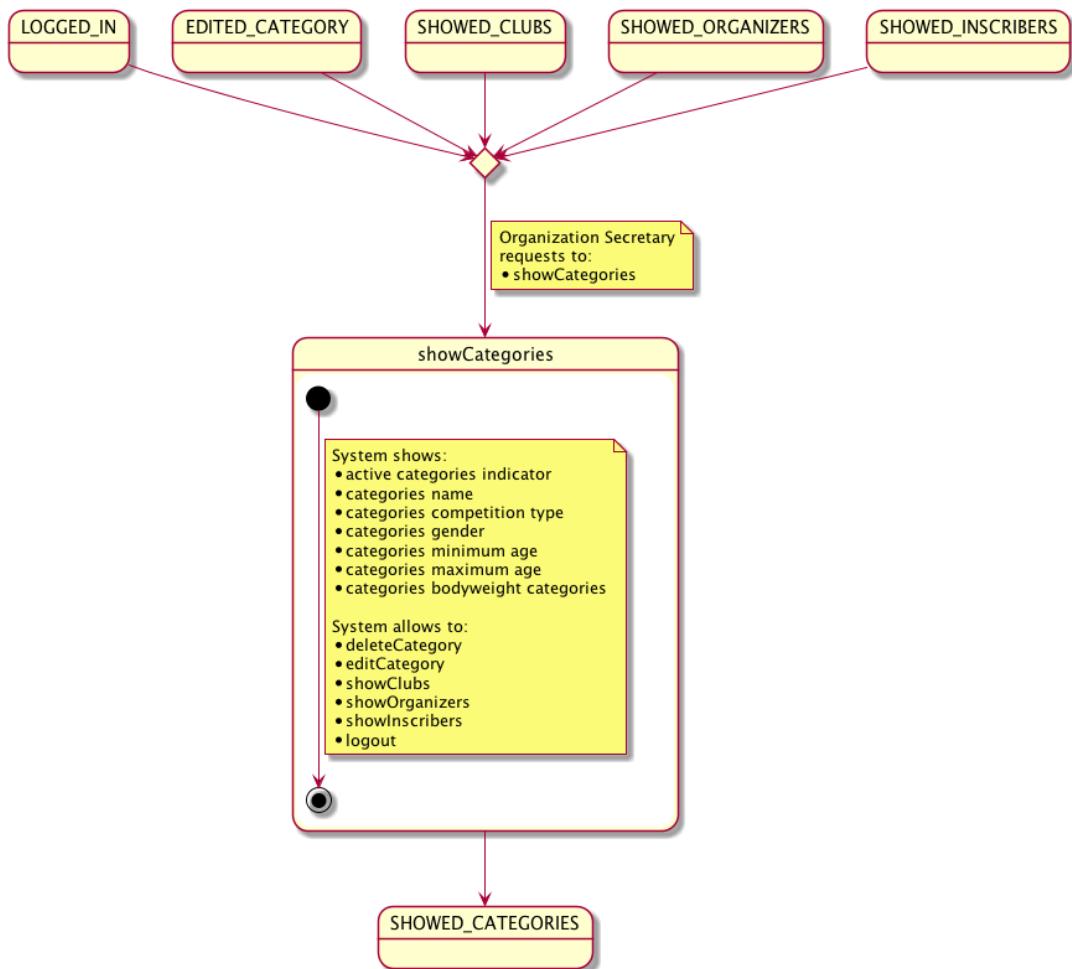
Fig. 26 – Caso de uso: Añadir o actualizar el estado de un levantamiento.



Especificación del caso de uso "editWeightChange"

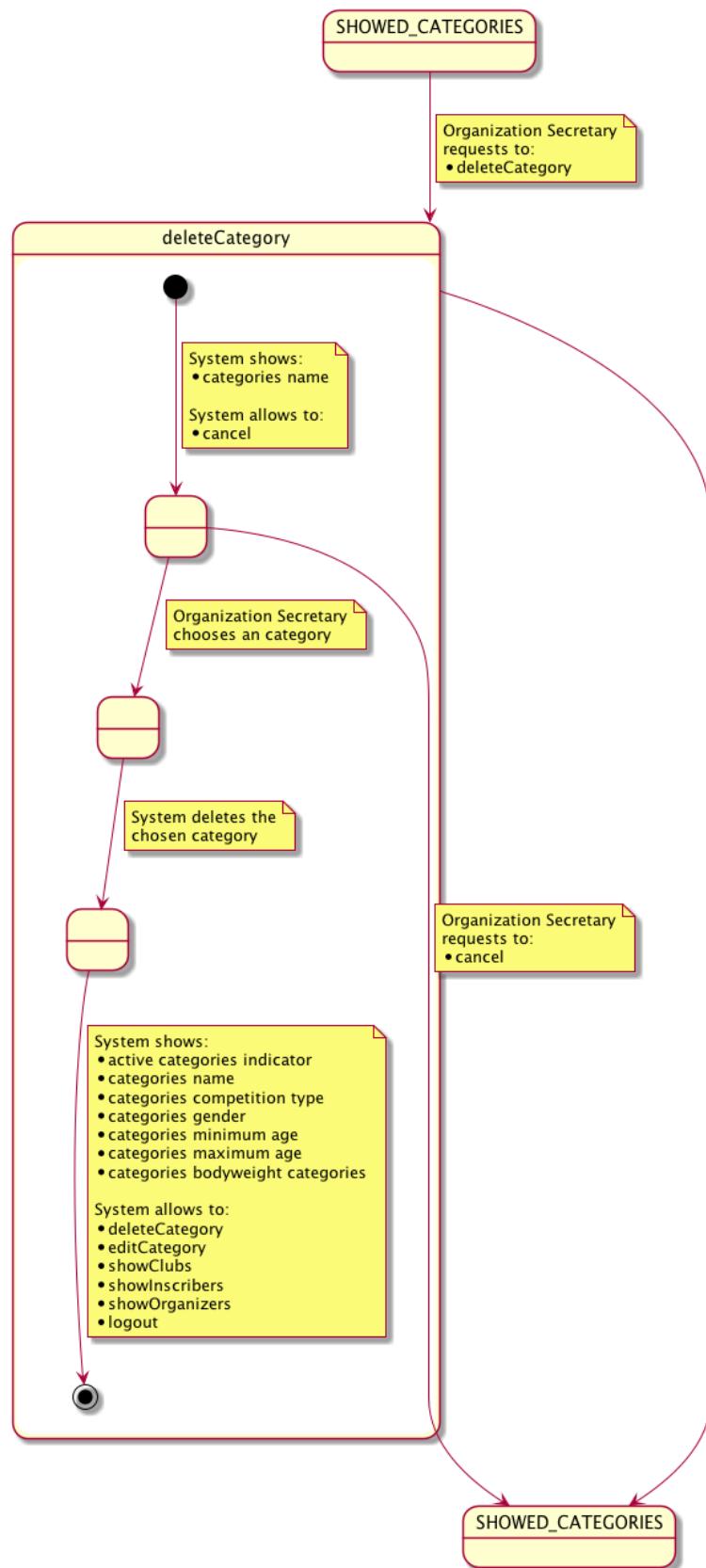
Fig. 27 – Caso de uso: Añadir o actualizar un cambio de peso.

4.3.2. Especificación de casos de uso del secretario de la organización



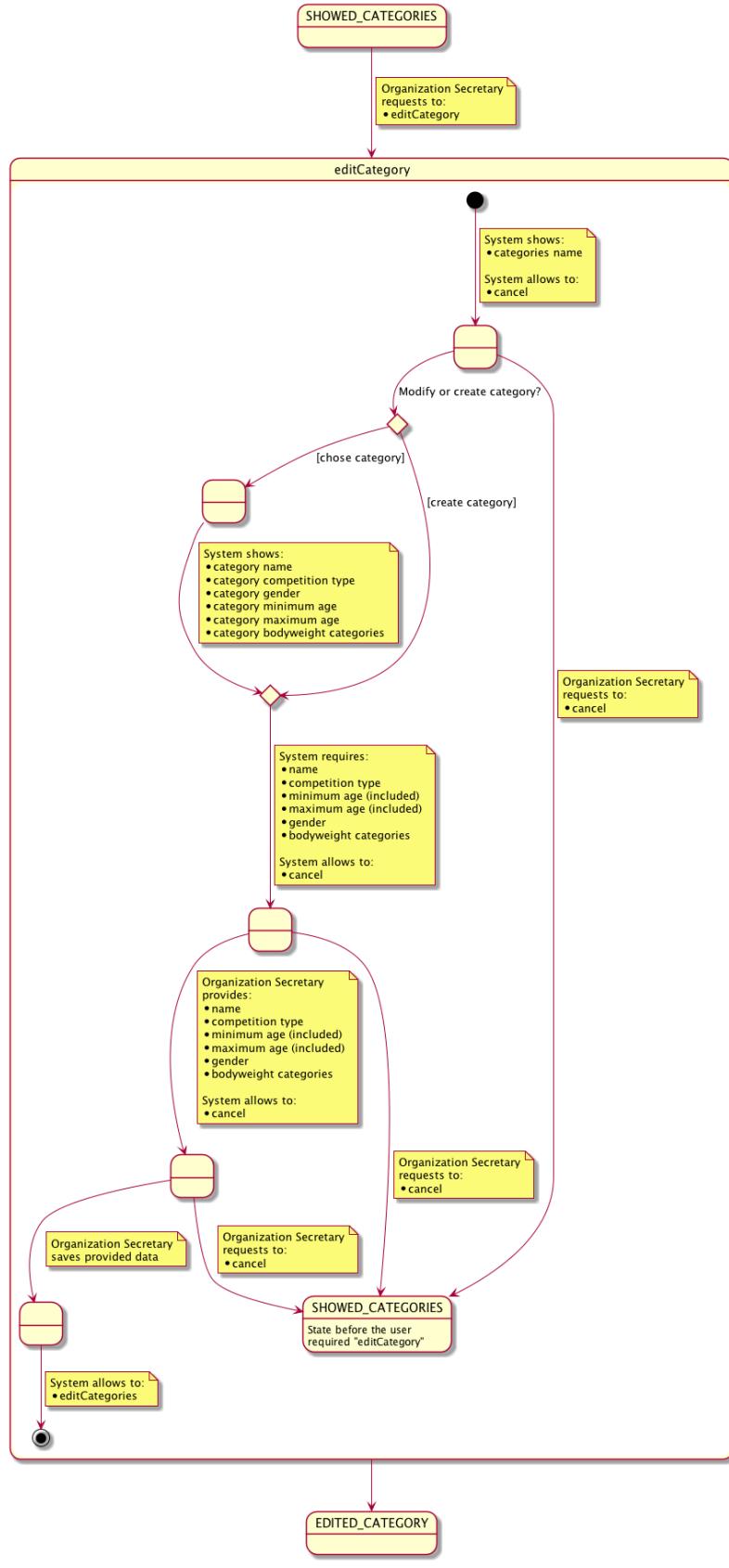
Especificación del caso de uso "showCategories"

Fig. 28 – Caso de uso: Mostrar categorías.



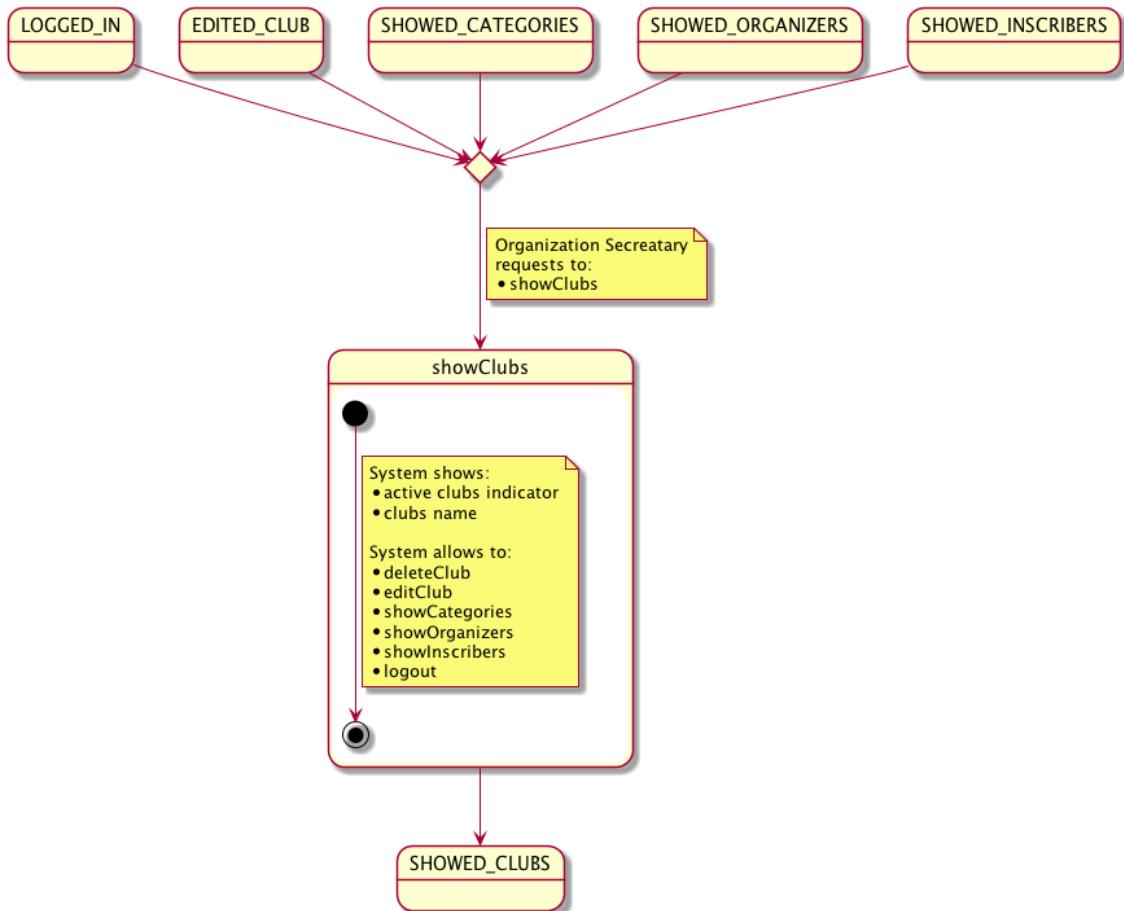
Especificación del caso de uso "deleteCategory"

Fig. 29 – Caso de uso: Eliminar una categoría.



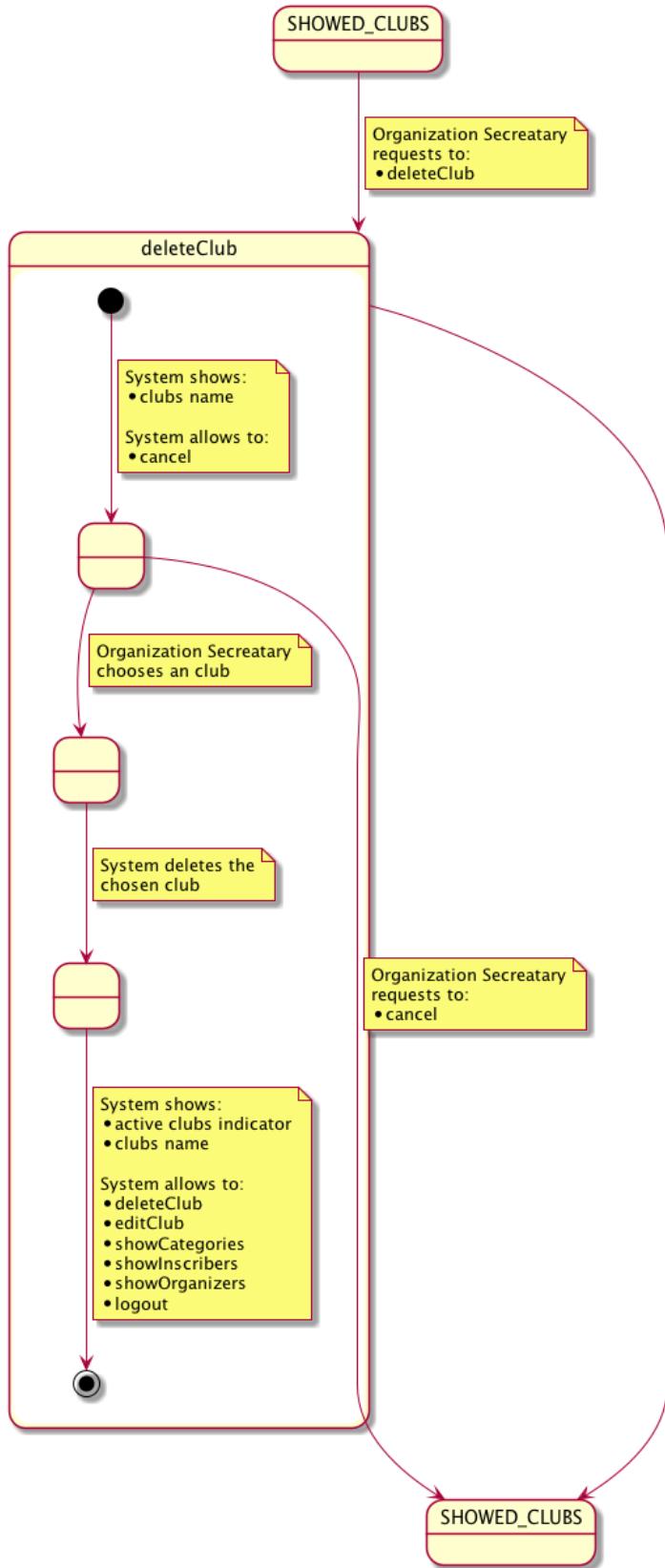
Especificación del caso de uso "editCategory"

Fig. 30 – Caso de uso: Añadir o actualizar una categoría.



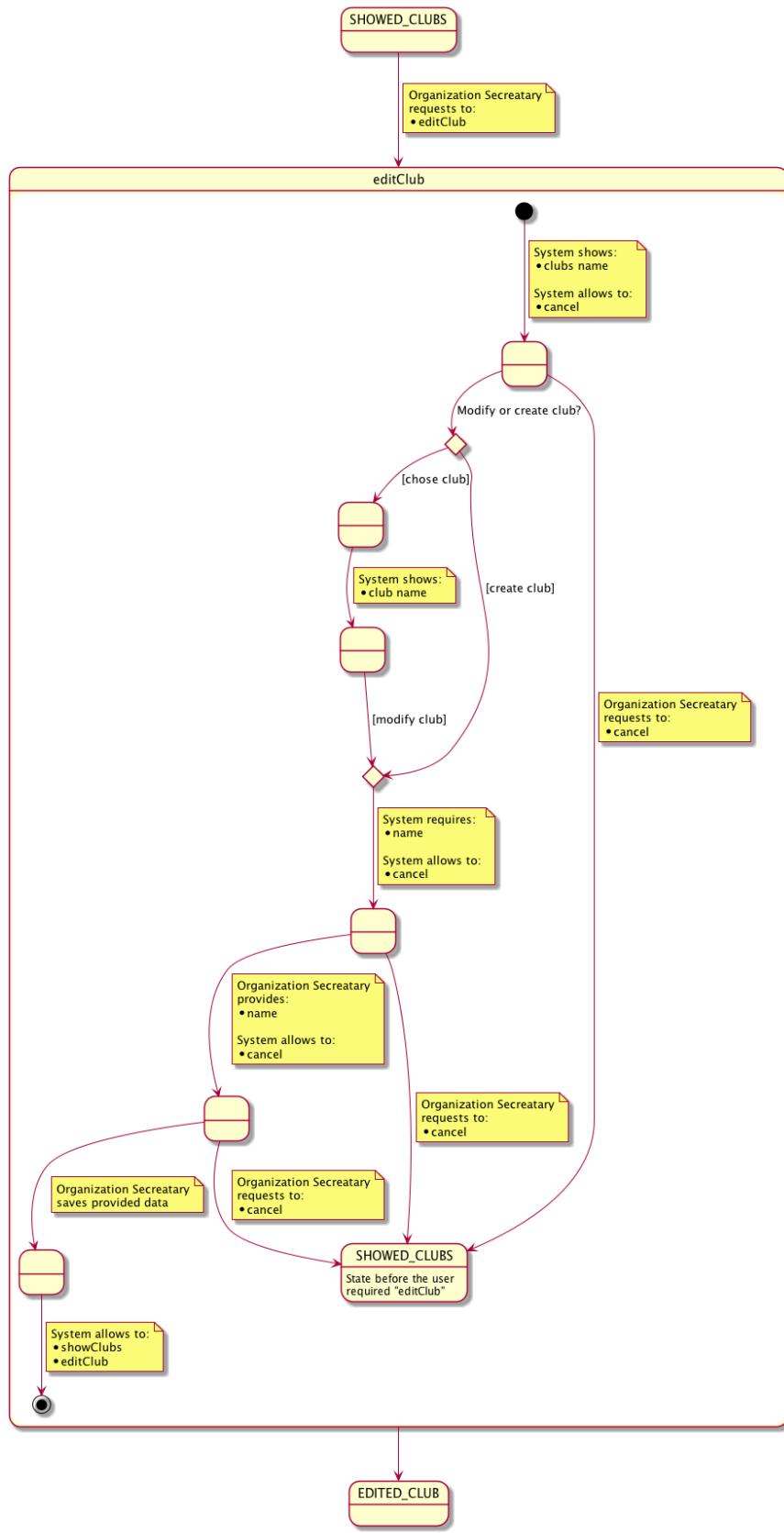
Especificación del caso de uso "showClubs"

Fig. 31 – Caso de uso: Mostrar clubs.



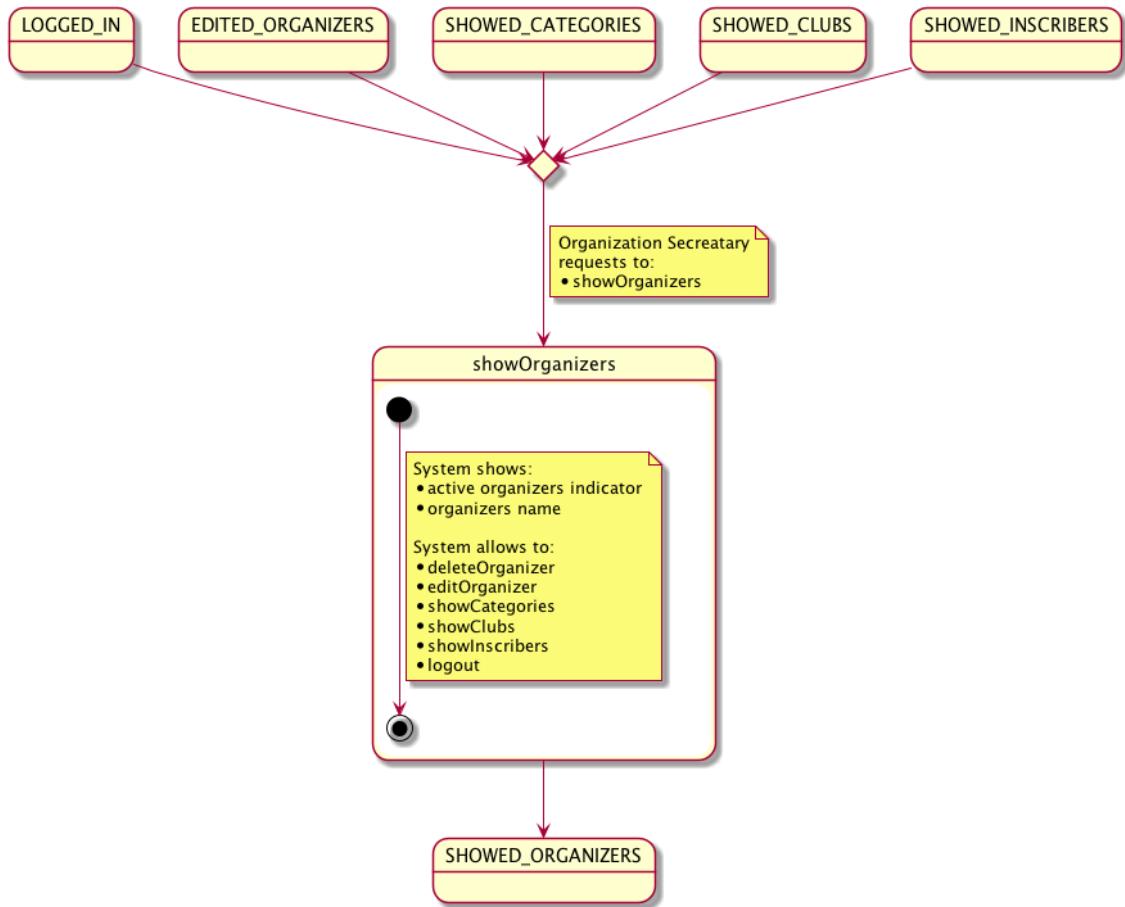
Especificación del caso de uso "deleteClub"

Fig. 32 – Caso de uso: Eliminar un club.



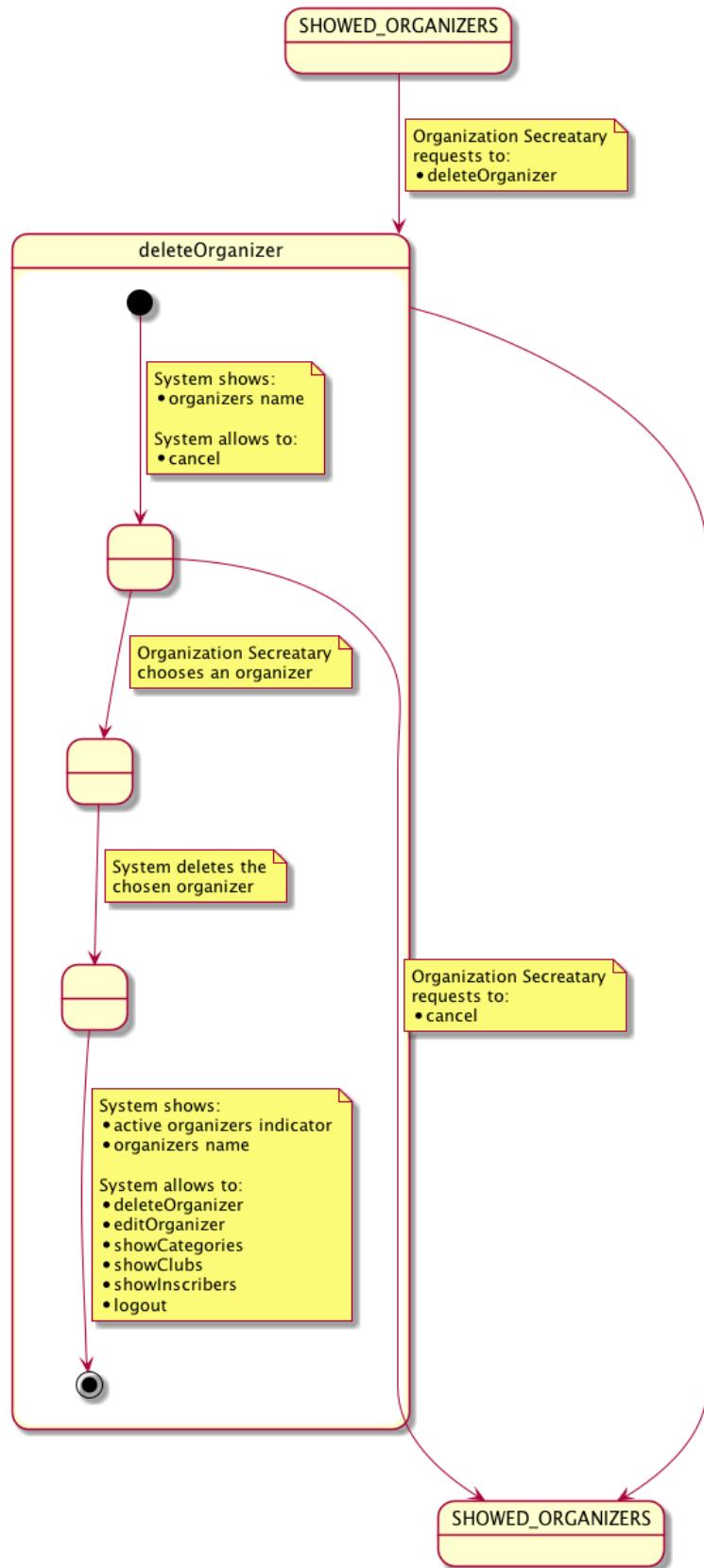
Especificación del caso de uso "editClub"

Fig. 33 – Caso de uso: Añadir o actualizar un club.



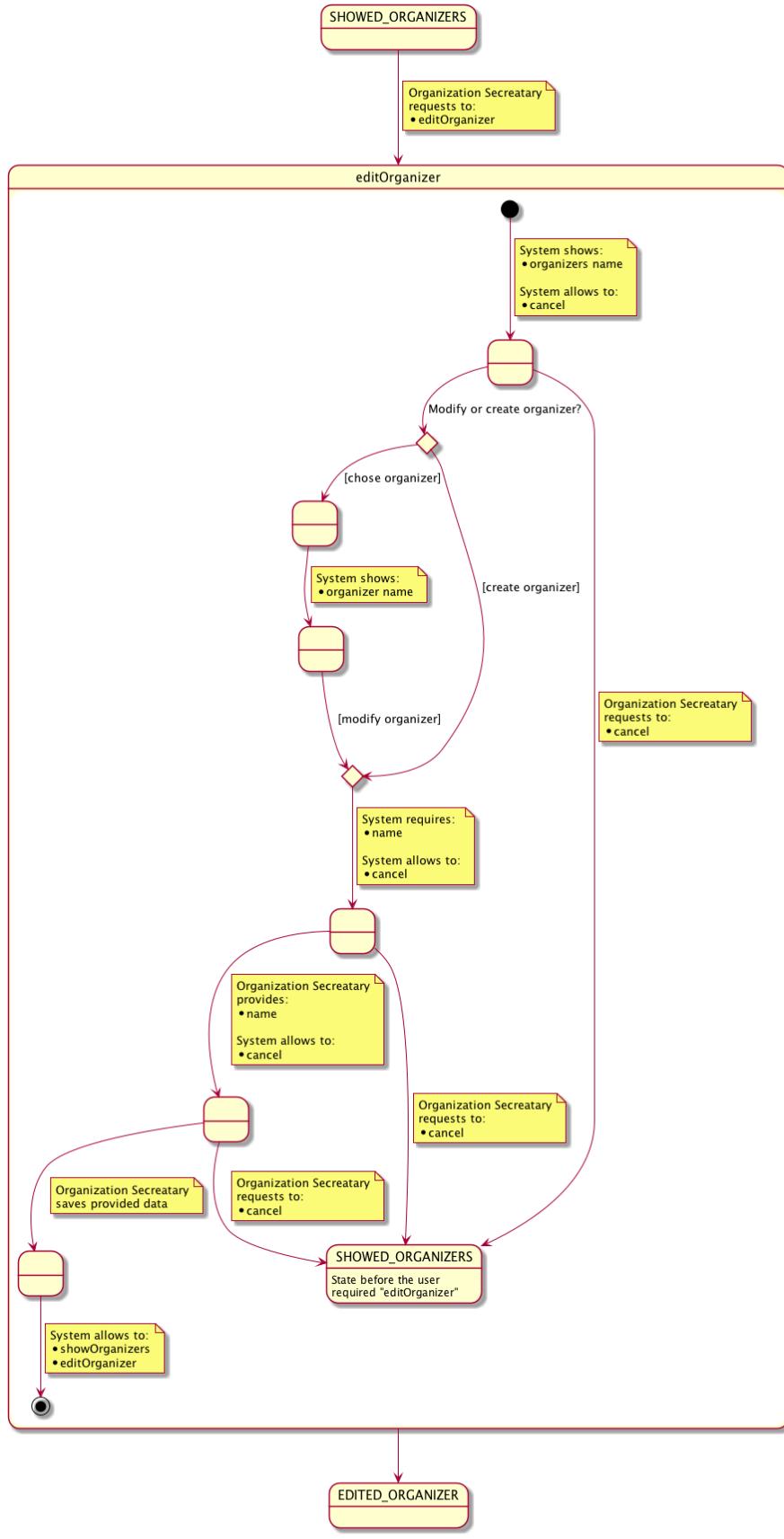
Especificación del caso de uso "showOrganizers"

Fig. 34 – Caso de uso: Mostrar organizadores.



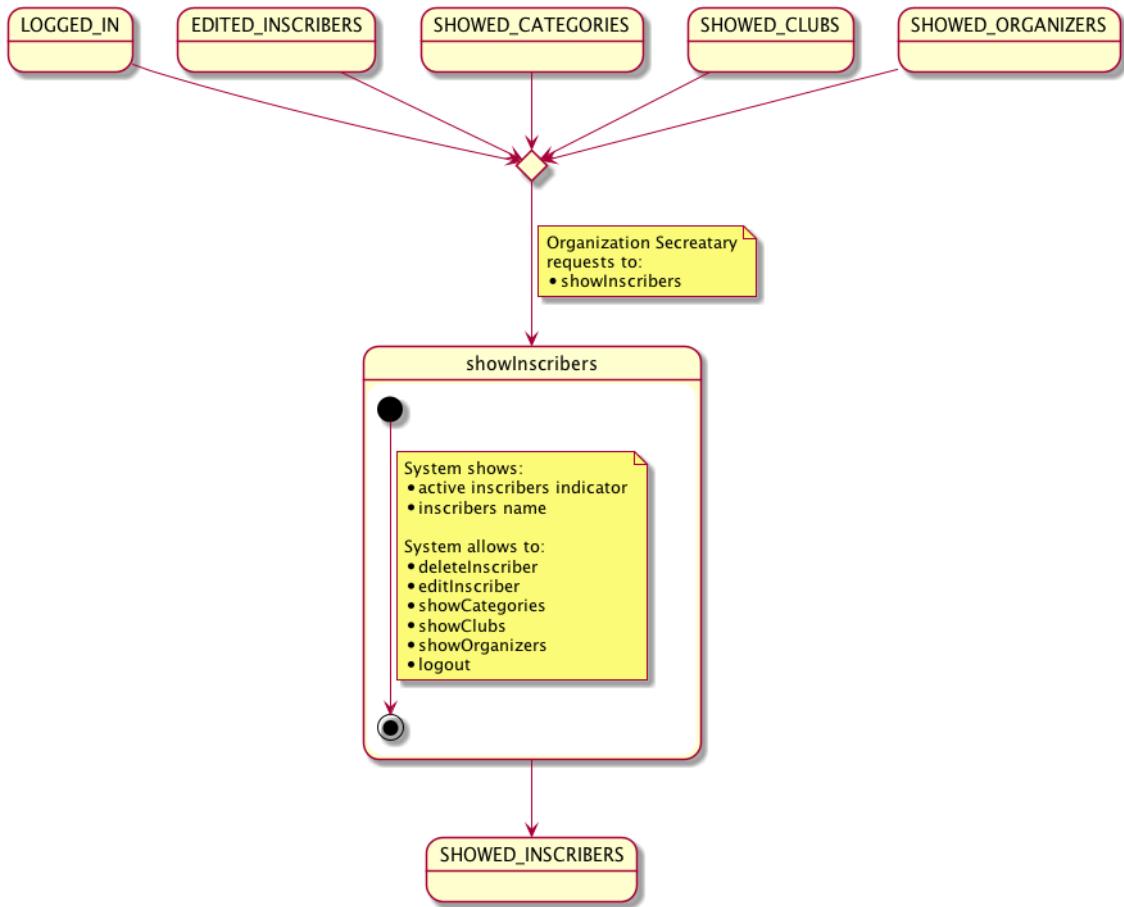
Especificación del caso de uso "deleteOrganizer"

Fig. 35 – Caso de uso: Eliminar un organizador.



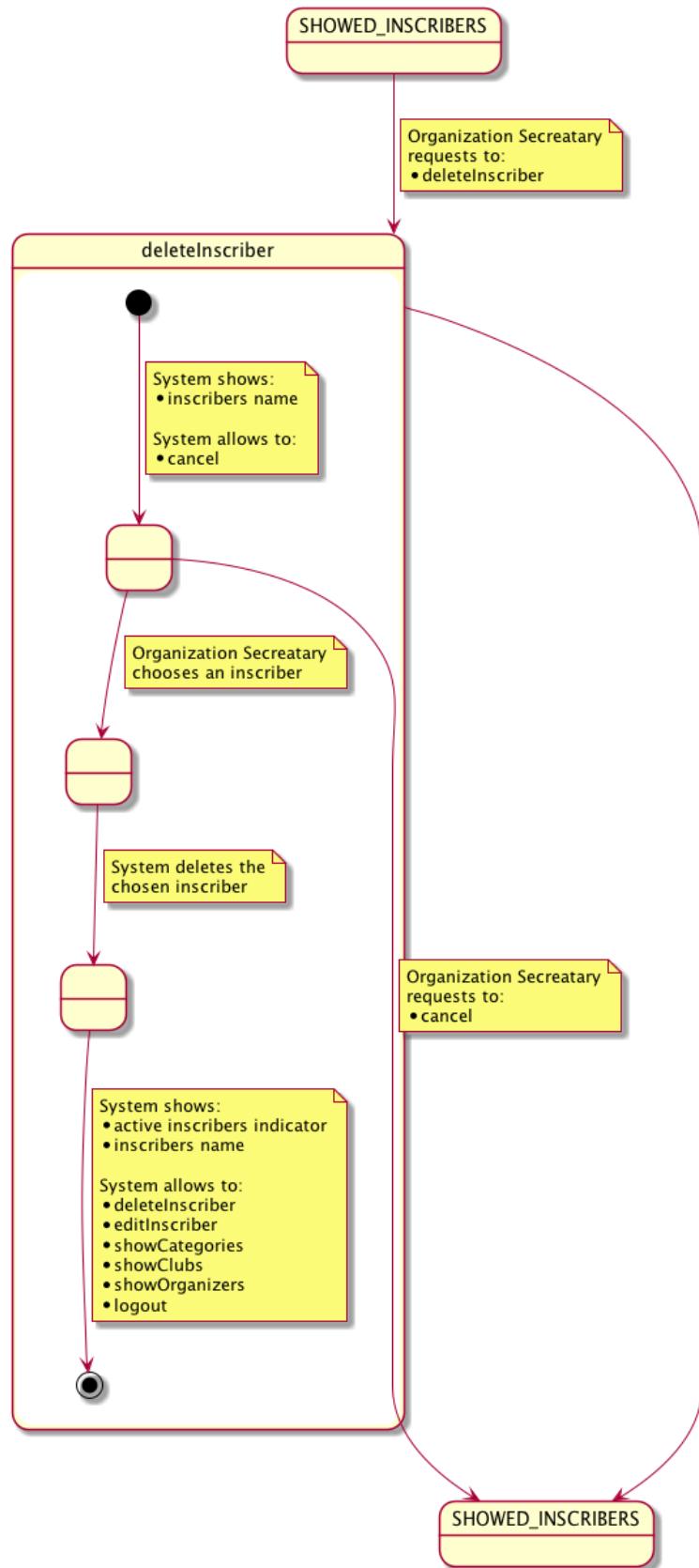
Especificación del caso de uso "editOrganizer"

Fig. 36 – Caso de uso: Añadir o actualizar un organizador.



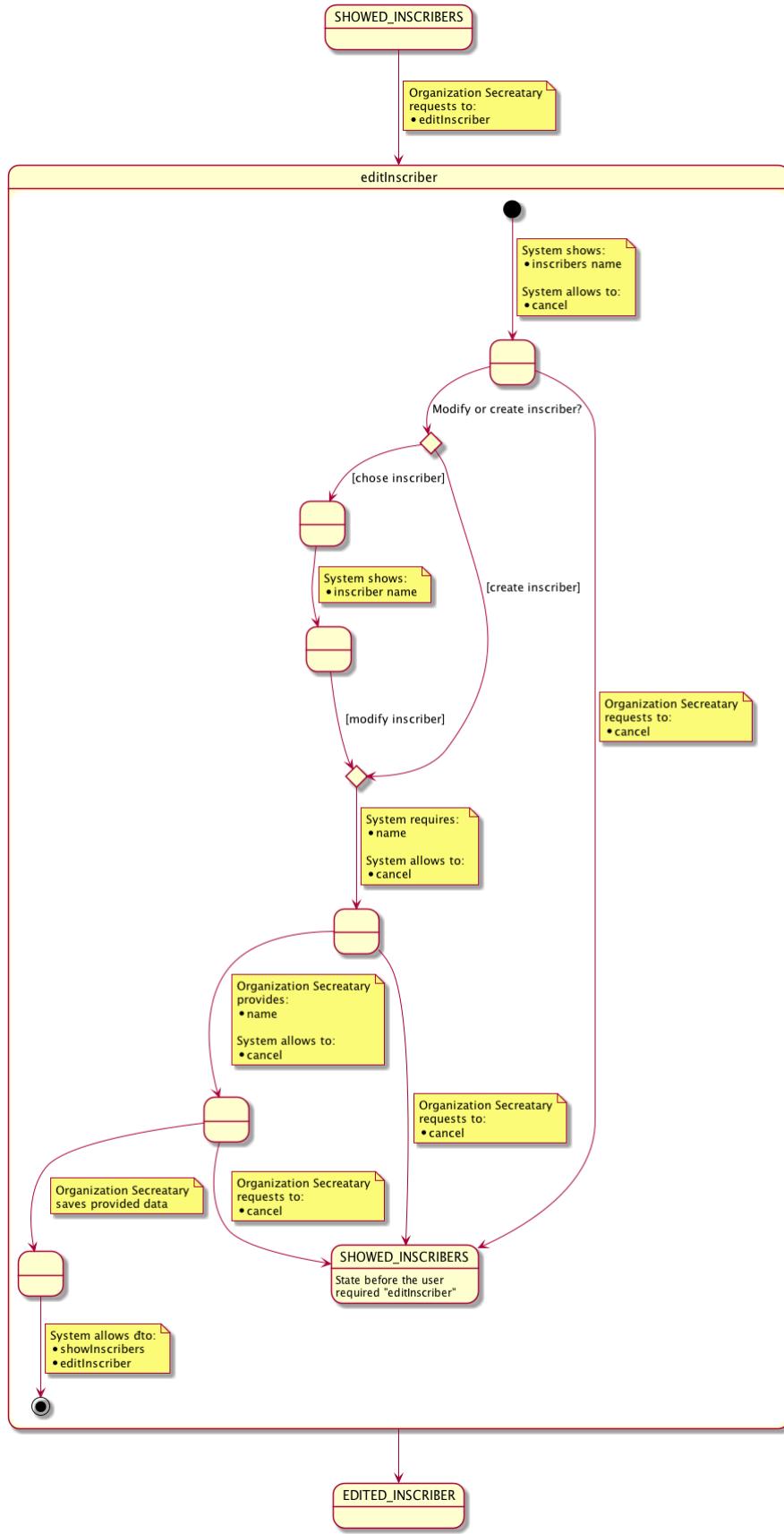
Especificación del caso de uso "showInscribers"

Fig. 37 – Caso de uso: Mostrar inscriptores.



Especificación del caso de uso "deleteInscriber"

Fig. 38 – Caso de uso: Eliminar un inscriptor.



Especificación del caso de uso "editInscriber"

Fig. 39 – Caso de uso: Añadir o actualizar un inscriptor.

4.4 Prototipado interfaz de usuario

A continuación se detallan los prototipos de las interfaces del usuario que permiten el intercambio de información detallado en los diagramas de especificación de casos de uso.

Para cada uno de los actores del sistema se ha generado un *webmap* donde se puede ver como se interconectan las distintas pantallas de prototipo y los casos de uso que se cubren.

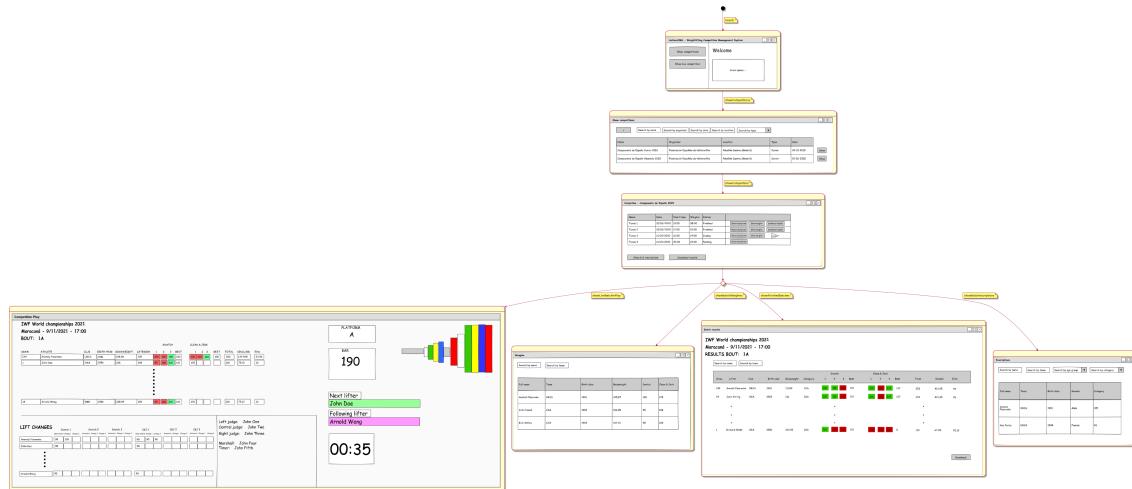


Fig. 40 – Webmap del espectador.

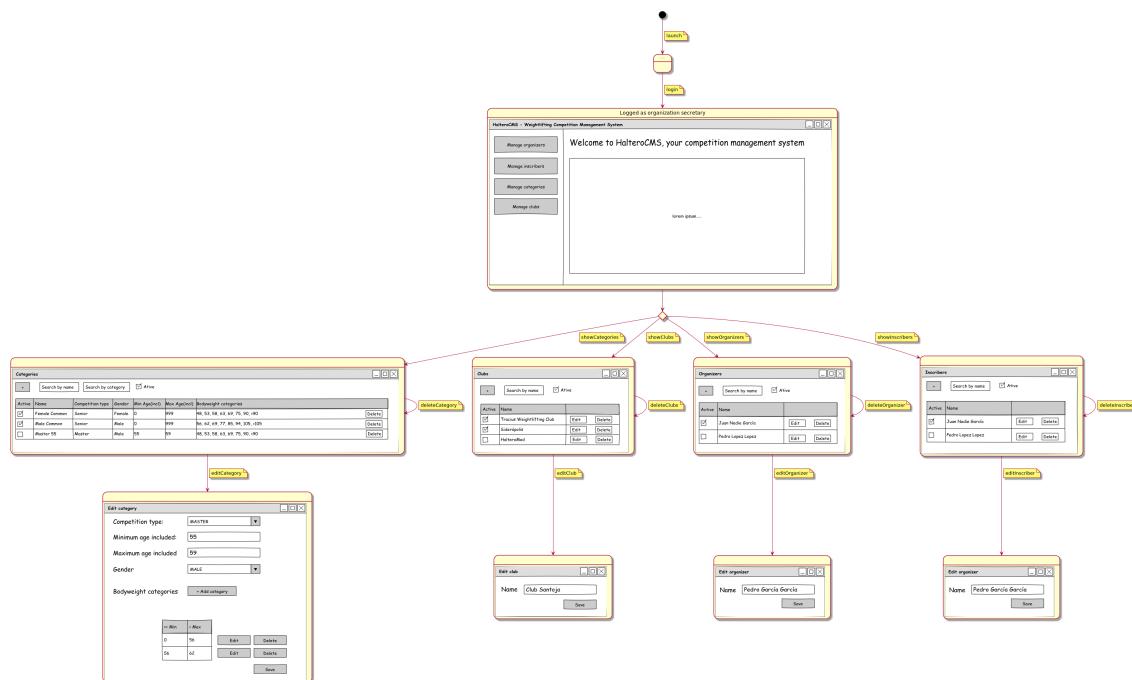


Fig. 41 – Webmap del secretario de la organización.

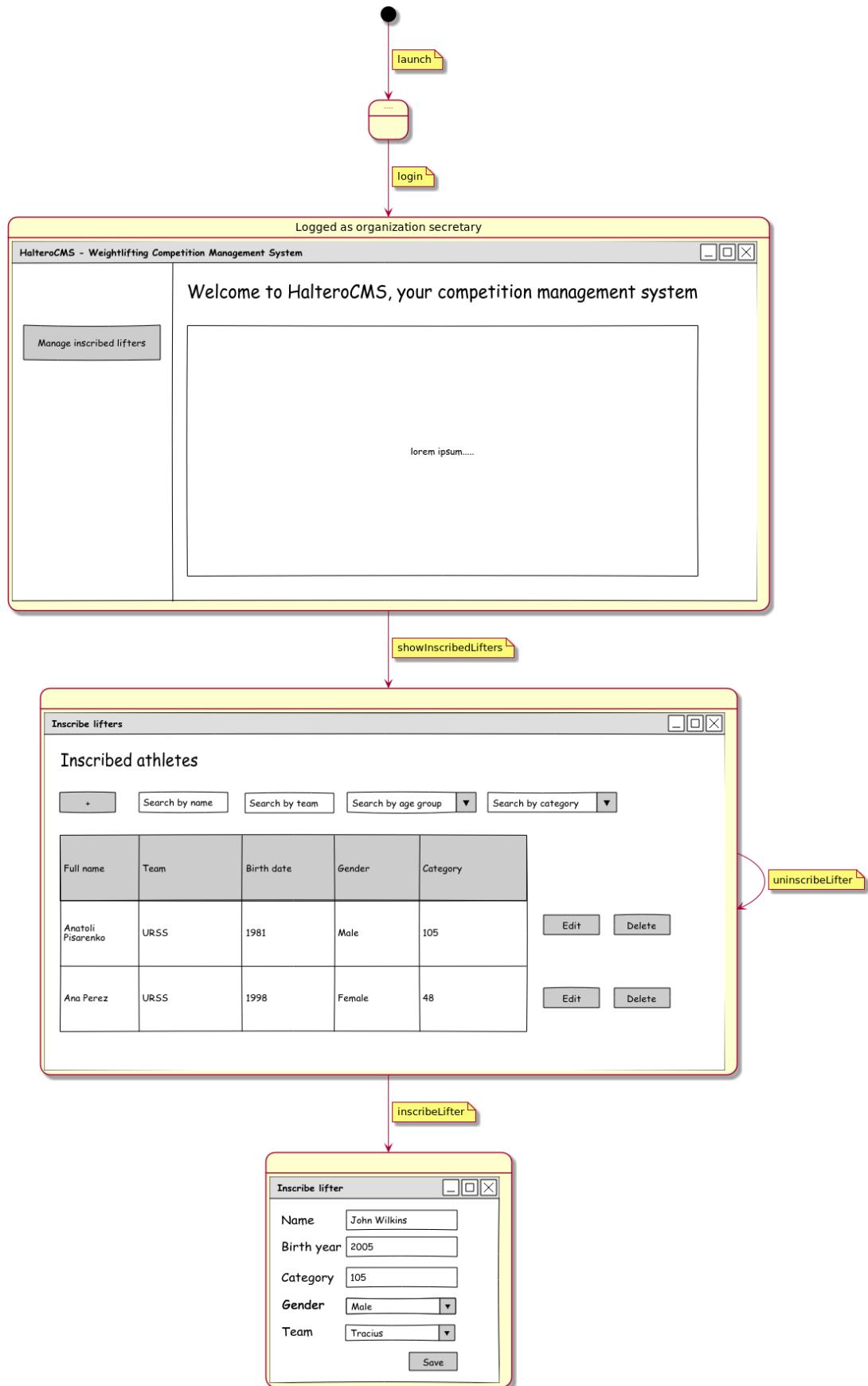


Fig. 42 – Webmap del inscriptor.

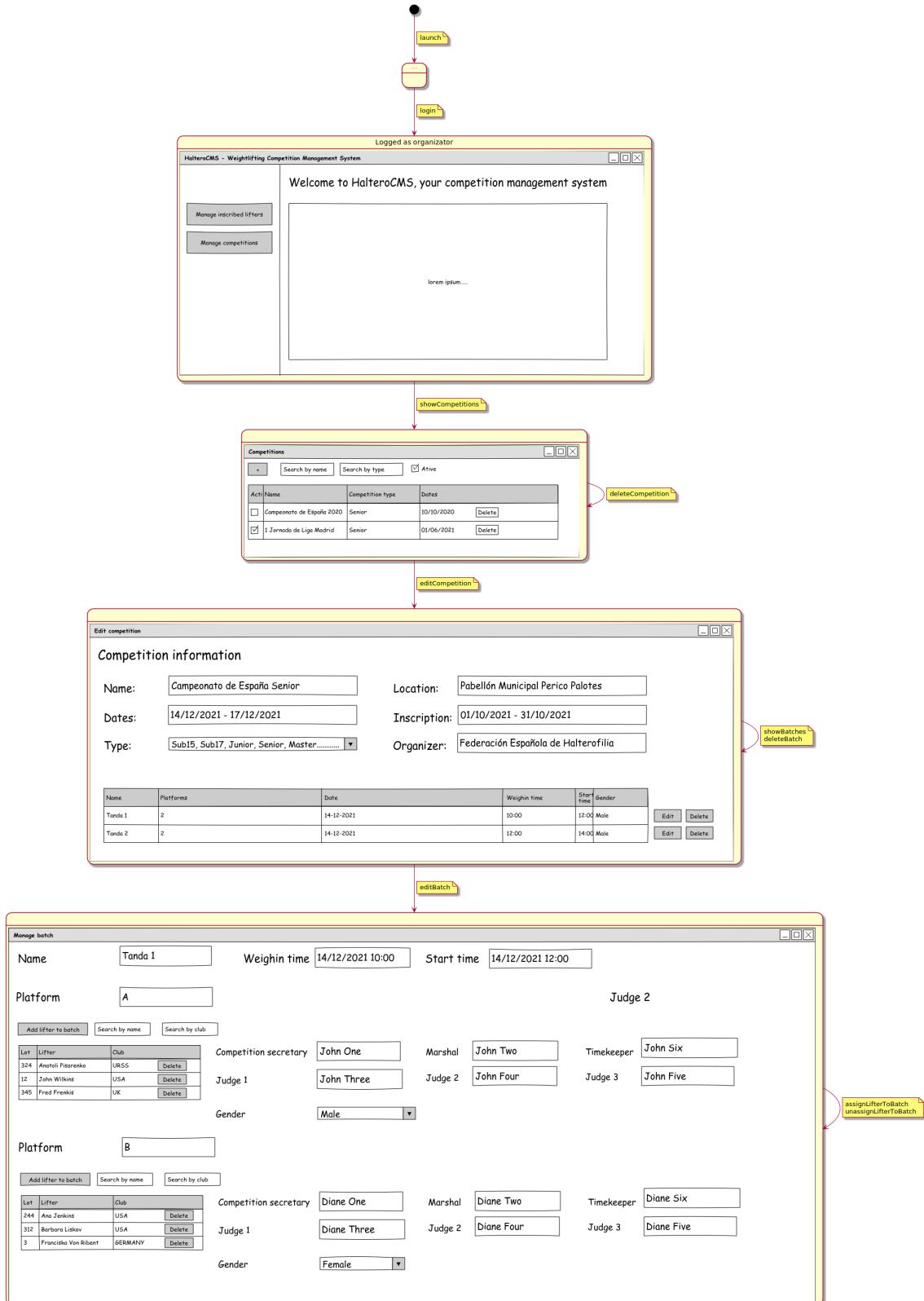


Fig. 43 – Webmap del organizador.

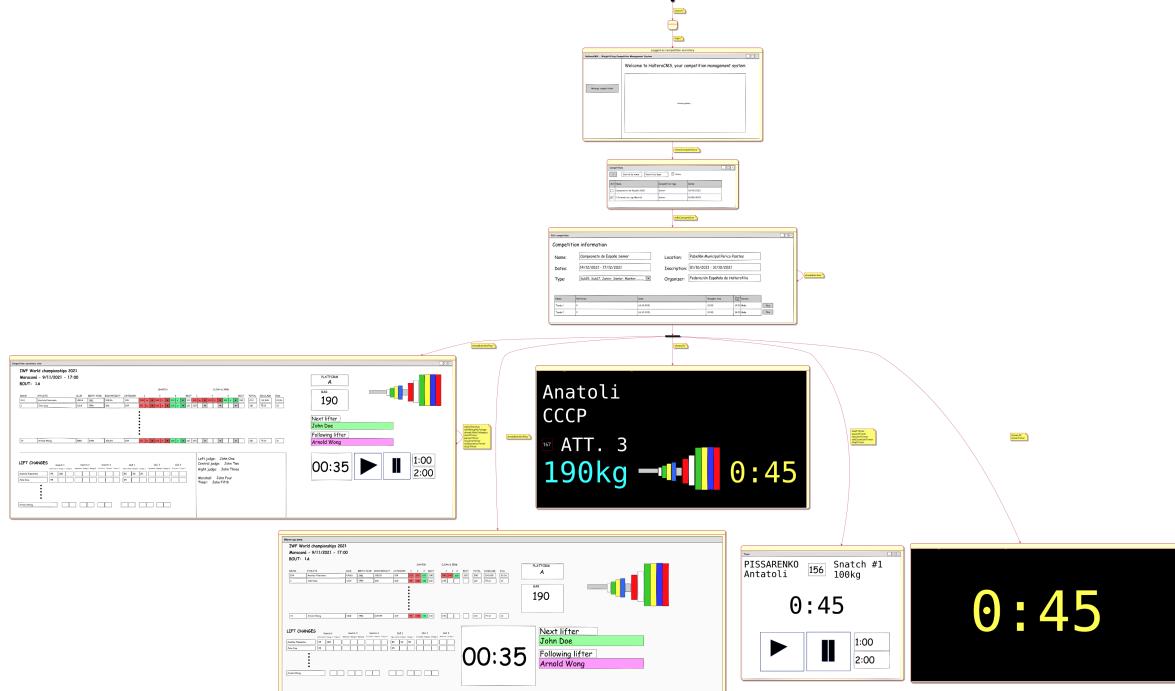


Fig. 44 – Webmap del secretario de la competición.

5. Disciplina de análisis

La finalidad de la disciplina de análisis es el flujo de trabajo (realización de casos de uso que incluyen trabajadores, actividades y diagramas) cuyo propósito principal es analizar los requisitos de la captura de requisitos a través de su refinamiento y la estructura para lograr una comprensión más precisa de los requisitos, una descripción de los requisitos que es fácil de mantener y nos ayudan a estructurar el sistema.

Las actividades que RUP recomienda para esta disciplina son las siguientes:

- Análisis de la arquitectura.
- Análisis de casos de uso.
- Análisis de clases.
- Análisis de paquetes.

Nosotros, en este proyecto, hemos trabajado en las actividades de análisis de arquitectura y de análisis de casos de uso.

5.1 Análisis de la arquitectura

Esta aplicación se ha estructurado por capas de presentación, capa de negocio y datos.

En este punto se organiza el sistema software, en clases modelo vista y controlador, se pintan algunas de las relaciones más importantes.

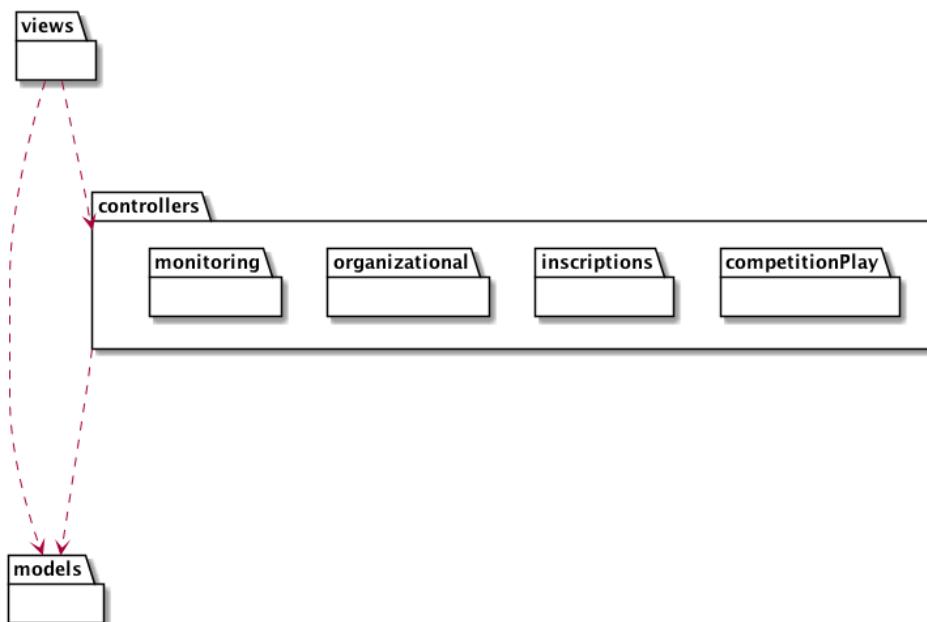


Fig. 45 – Análisis global de la arquitectura.

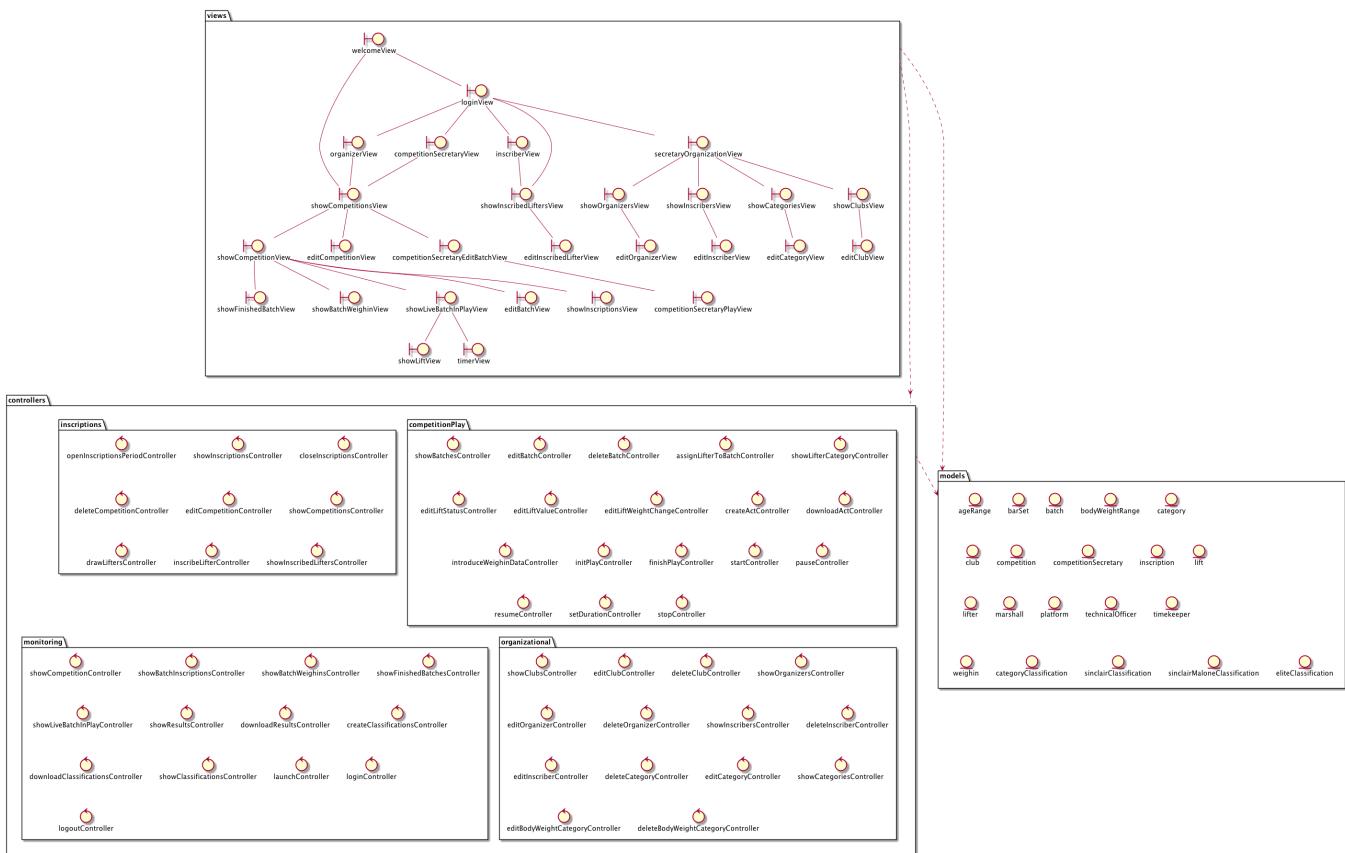


Fig. 46 – Análisis de la arquitectura al completo.

5.2 Análisis de casos de uso

Tras realizar la actividad anterior, análisis de la arquitectura, podemos comenzar el análisis de los casos de uso. En esta actividad trataremos de recoger qué componentes de los que hemos obtenido en el análisis de la arquitectura están involucrados en cada caso de uso, y de qué forma se relacionan y a través de qué mensajes.

Nos hemos centrado en esta ocasión en dos casos de uso distintos: 'introduceWeighinData' y 'showBatchInPlay', o lo que es lo mismo, añadir un nuevo pesaje y mostrar la tanda en curso. A continuación, las Fig. 47 y 48 muestran el análisis de estos dos casos de uso.

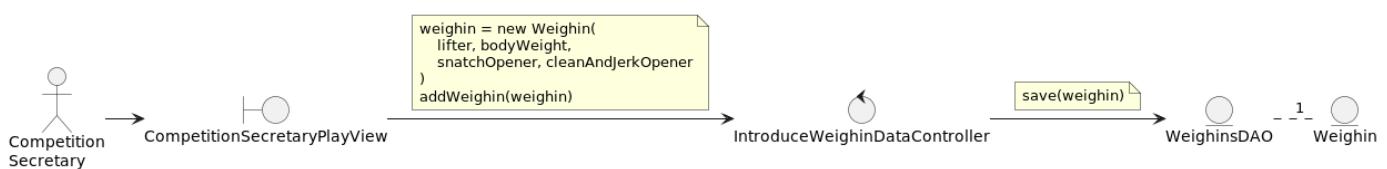


Fig. 47 – Análisis del caso de uso 'introduceWeighinData'.



Fig. 48 – Análisis del caso de uso 'showBatchInPlay'.

6. Disciplina de diseño

El principal propósito de la disciplina de diseño es desarrollar modelos enfocados sobre los requisitos no funcionales y el dominio de la solución, y prepararnos para la implementación y las pruebas del sistema.

En esta fase, identificaremos los distintos subsistemas pero también dejaremos claro cómo se conectan entre sí, es decir, definiremos las posibles interfaces entre los mismos. Desgranaremos a su vez todo el trabajo de implementación en unidades más pequeñas, para que pueda ser asumida cada una por distintos equipos de trabajo. Y además profundizaremos en los requisitos no funcionales, detectando toda limitación que pueda existir en las tecnologías de las cuáles vayamos a hacer uso.

Las actividades recomendadas por RUP para esta disciplina son las siguientes:

1. Diseñar la arquitectura.
2. Diseñar casos de uso.
3. Diseñar clases.
4. Diseñar subsistemas.

En este proyecto hemos trabajado en las tres primeras actividades, habiendo sido generados una serie de diagramas que vamos a explicar a continuación.

6.1. Diseñar la arquitectura

Los objetivos del diseño de la arquitectura son identificar clases arquitectónicas significativas de diseño, subsistemas específicos de la aplicación e identificar el software y las tecnologías en las que se va a apoyar. Este software pueden ser servidores, navegadores web o componentes software, entre otros.

En el diseño de la arquitectura, se han identificado los nodos y conexiones reflejados en la siguiente figura.

La aplicación está estructurada por capas y se organiza en una capa de presentación, una capa de negocio y una capa de datos. Se profundizará en ello más adelante. Además, es una aplicación multi-página, es decir, en la parte del cliente está el navegador y es el servidor el que se encarga de generar todas las vistas en html, css y javascript que manda posteriormente al cliente.

Estamos usando una arquitectura MVP Controlador Supervisor:

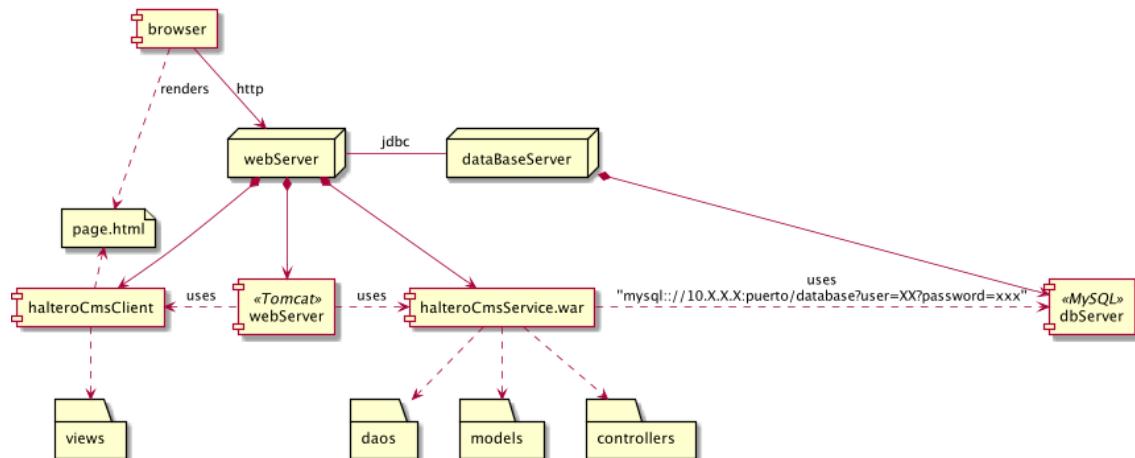


Fig. 49 – Diagrama de despliegue.

En primer lugar, vemos que existe un webServer que se compone del cliente web (halteroCmsClient) y de los servicios del backend (halteroCmsService.war). Debido a nuestra escasa experiencia desarrollando con tecnologías front, hemos decidido utilizar un motor de plantillas (Thymeleaf en este caso), y por eso el servidor Tomcat usa ambos.

El cliente renderiza las plantillas y genera una serie de páginas html, las cuales son consumidas por el navegador web. Y por otro lado, el servidor se conecta con una base de datos MySQL.

Vemos que el cliente hace uso del paquete del código correspondiente a las vistas, y que el war de los servicios hace uso de los controladores, modelos y DAOs.

Por último, tenemos que añadir que la aplicación se trata de un monolito y que por tanto, se genera un único artefacto que contiene la aplicación. Este tipo de aplicación tiene como ventajas principales su facilidad a la hora de gestionar versiones y despliegues.

6.2. Diseñar casos de uso

Para el desempeño de esta actividad, hemos seleccionado la herramienta de los diagramas de secuencia, especificando en ellos componentes concretos (vistas html, clases java, etc.) y los mensajes que éstos intercambiarán.

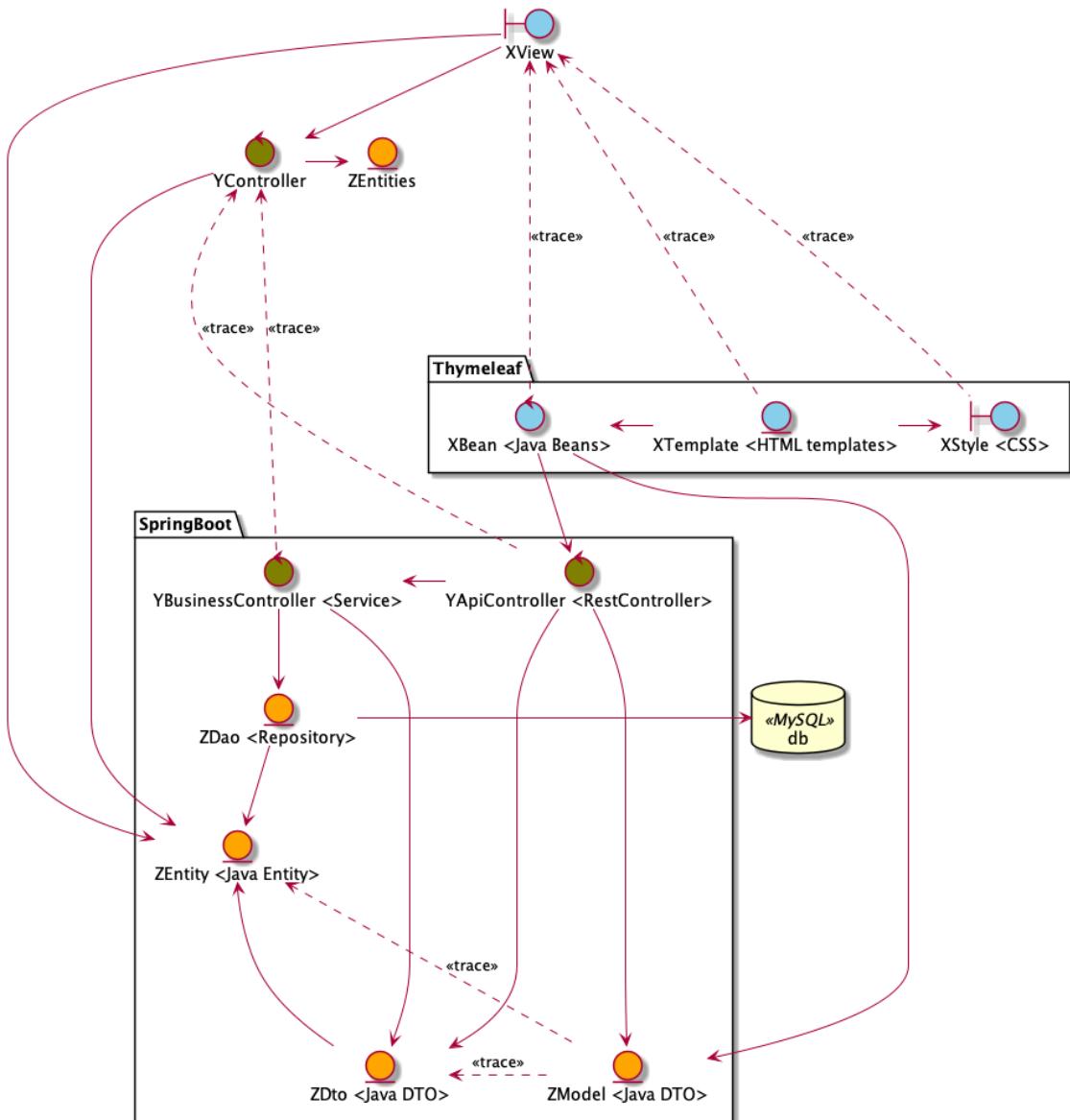


Fig. 50 – Diseño para cualquiera de los casos de uso.

- Clases View: se corresponden con una clase Template, Style y Bean. La relación que hay entre estas clases de diseño se podría ver como las relaciones entre clases que hay en un patrón MVP (Modelo Vista Presentador) en la capa de presentación: donde las clases Template serían el modelo, las clases CSS, serían la Vista o representación visual de los modelos, y los Beans, los presentadores encargados de presentar los modelos.
- Clases Controller: estas clases se encargan de desacoplar las clases de negocio de las clases de presentación, de esta manera, es muy fácil cambiar de tecnologías ya que la lógica de negocio se ve impactada a muy bajo nivel.
- Clases DAO: la funcionalidad de estas clases es conectarse con la base de datos y realizar todas las consultas necesarias para recuperar las entidades.

- Clases Entity: se corresponden con el objeto de base de datos desacoplado de cualquier tecnología.

Únicamente hemos diseñado dos casos de uso, pero si nos pusiésemos a diseñar más, los diagramas de secuencia serían bastante similares entre ellos. Por este motivo, incluimos a continuación un diagrama de secuencia genérico que podría aplicarse a cualquier diseño de caso de uso.

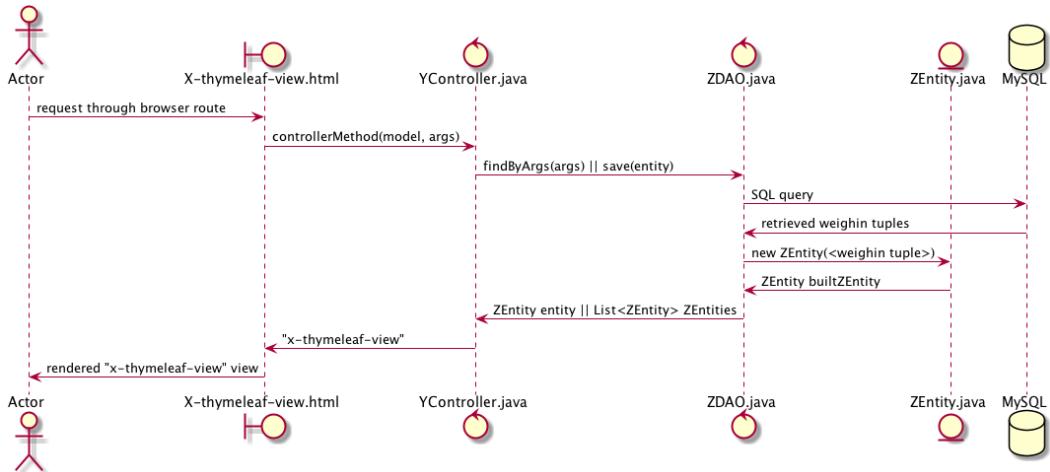


Fig. 51 – Diagrama de secuencia genérico para cualquier caso de uso.

6.2.1. Diseño del caso de uso 'introduceWeighinData'

Esta disciplina nos ayuda a identificar las clases de diseño y subsistemas necesarios para realizar el caso de uso. En la siguiente figura se muestra el diagrama de secuencia para el caso de uso 'introduceWeighinData', correspondiente a la operación de introducir los datos de un pesaje (que son el peso corporal del levantador y los openers).

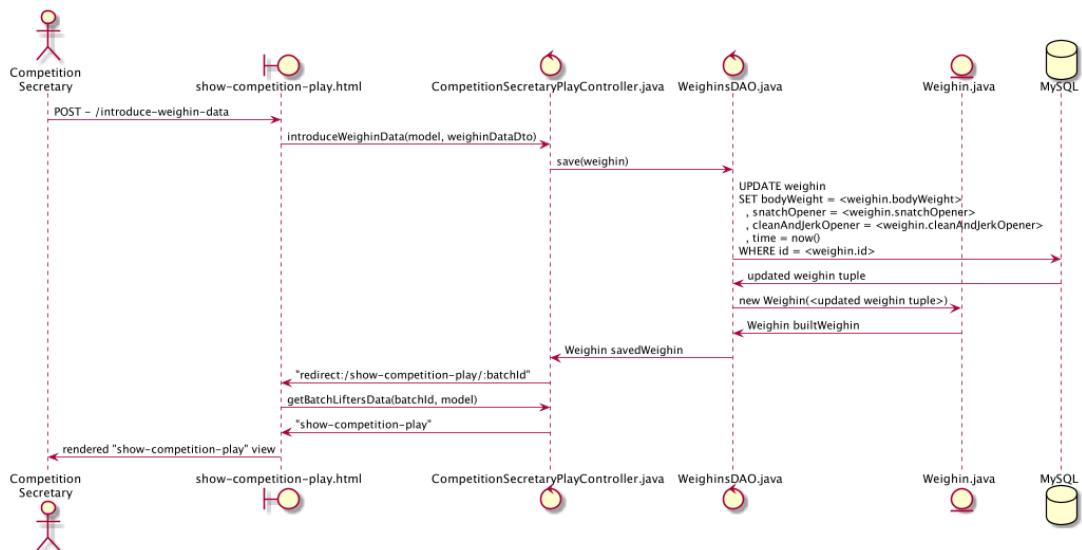


Fig. 52 – Diagrama de secuencia para el caso de uso 'introduceWeighinData'.

El detonante de este caso de uso es el Secretario de la Competición, quien interactuará con la vista show-competition-play a través del navegador. Pero debemos hacer un apunte en este diagrama, y es que a la hora de implementar este caso de uso, gracias al framework de desarrollo, todo lo que haya a la derecha del componente WeighinsDAO será transparente al programador. El programador se encargará de definir una interfaz en WeighinsDAO.java y el framework hará el resto del trabajo (componer la query, consultar la query a MySQL, llamar al constructor de Weighin, etc.).

6.2.2. Diseño del caso de uso 'showBatchInPlay'

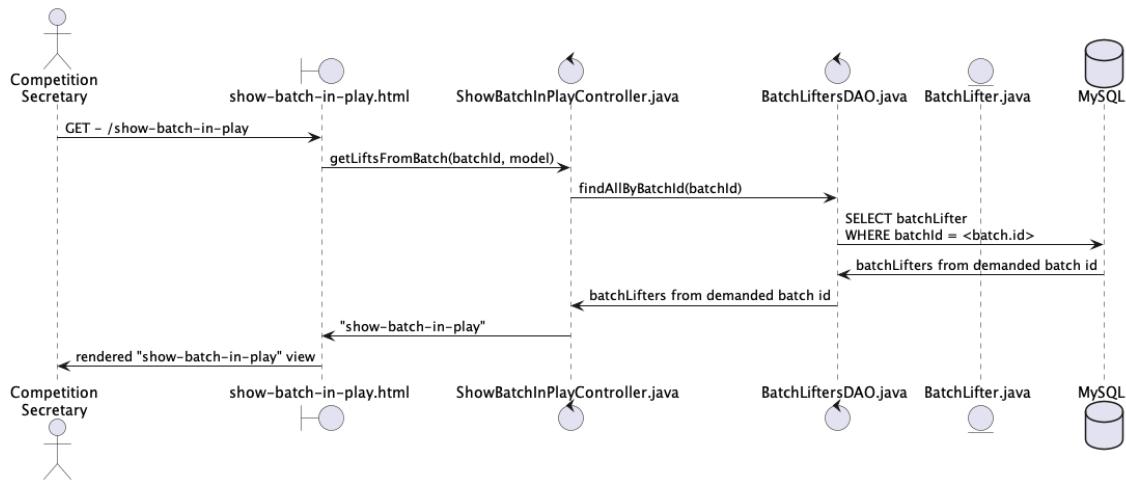


Fig. 53 - Diagrama de secuencia para el caso de uso 'showBatchInPlay'.

El detonante de este caso de uso es el Secretario de Competición, que a través de la vista correspondiente solicita obtener los levantadores y levantamientos correspondientes a la tanda en juego.

6.3. Diseñar clases

Inspirándonos principalmente en el modelo del dominio, y basándonos en todo el trabajo realizado en posteriores disciplinas, hemos diseñado las clases que formarán parte de nuestro desarrollo. Quedan recogidas en el siguiente diagrama de clases.

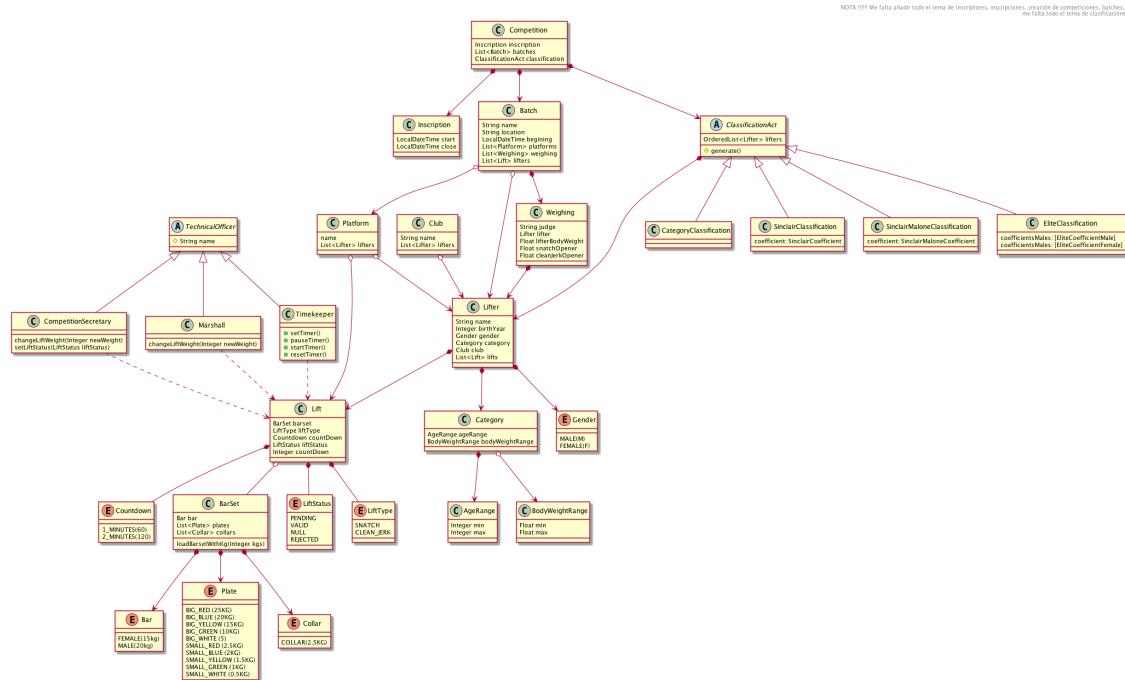


Fig. 54 – Diagrama de clases.

Al contrario que en el modelo del dominio, esta división en clases ya está preparada para ser llevada directamente al código java, y cada una de ellas ya dispone de atributos tipados e incluso algunos métodos.

7. Disciplina de implementación

La disciplina de implementación, es el flujo de trabajo cuyo propósito principal es implementar el sistema en términos de componentes, como pueden ser el código, los fichero binarios, ejecutables, vistas renderizadas, etc.

El objetivo que se debe conseguir con esta disciplina es la generación del código necesario para la implementación de las clases y objetos en términos de componentes, definiéndose de forma previa su organización en términos de subsistemas divididos en capas, y cubriendose esta implementación con las necesarias pruebas unitarias.

Las actividades recomendadas por RUP para esta disciplina son las siguientes:

1. Implementación de la arquitectura.
2. Integración de los sistemas.
3. Implementación de un subsistema.
4. Implementación de una clase.
5. Realización de las pruebas de unidad.

Tras todo el trabajo realizado en disciplinas anteriores, el cometido de esta disciplina resulta bastante mecánico e instantáneo. Simplemente, basta con plasmar en el código las interacciones definidas en los diagramas de diseño esbozados en la disciplina anterior, siguiendo unas buenas prácticas de desarrollo para que el código quede lo más limpio posible.

7.1. Casos de uso implementados

Puesto que en la disciplina de diseño nos hemos centrado en los casos de uso 'introduceWeighinData' y 'showBatchInPlay', hemos querido cubrirlos también desde el punto de vista de la disciplina de desarrollo. Es la primera vez que hacemos un análisis tan exhaustivo del proyecto de forma previa a la implementación, por lo que nos ha sorprendido bastante la rapidez con la que hemos conseguido completar los desarrollos.

Apenas hemos necesitado invertir tiempo en pensar mucho, porque la gran mayoría de los problemas que podrían surgir ya fueron planteados previamente.

7.2. Test-First Development

Para nosotros los tests son imprescindibles en cualquier tipo de proyecto software. Por tanto, durante la ejecución de la Disciplina de Implementación hemos aplicado Test-First Development (TFD), que consiste en implementar las pruebas de forma previa a la funcionalidad. Esta filosofía de desarrollo funciona bajo la premisa de

que los tests son parte de la aplicación, no un añadido del que se pueda prescindir. Sin los tests, el código no está completo.

La razón de más peso por la cuál hemos decidido hacer TFD es que, al igual que TDD (Test Driven Development), es una muy buena práctica para no desarrollar "código de más". De esta forma, aprovechando nuestra experiencia a la hora de hacer tests, se ha invertido mucho menos tiempo en implementar los casos de uso desarrollados que si hubiésemos programado el código funcional al principio del todo.

Además, teniendo descrita la especificación de los casos de uso en la Disciplina de Requisitos, ha sido muy fácil enumerar los distintos tests que la plataforma debe pasar para dar por concluido el desarrollo. En este punto queremos indicar que los únicos tests que se han llevado a cabo son los test unitarios, que son los indicados por RUP en esta Disciplina.

Al haber utilizado RUP no hemos podido aplicar TDD, ya que TDD va un paso más allá e implica diseñar mediante los tests. Existiendo un diseño previo a la Disciplina de Implementación, TDD resulta totalmente incompatible con este proyecto, al contrario que TFD que no debe ser menospreciado en absoluto.

Siguiendo TFD se podrían desarrollar de forma previa los test end to end, aunque entre todos los posibles tipos de tests que se pueden añadir, los test unitarios son los que más útiles nos resultan para completar la implementación de nuestros casos de uso.

7.3. Arquitectura por capas vs. arquitectura hexagonal

Aunque el framework de desarrollo RUP no establece ninguna pauta acerca de cómo estructurar el código, hemos querido dedicar en este punto algunas breves líneas a la arquitectura hexagonal, que aunque está muy de moda y se ofrece como algo novedoso, bajo nuestro punto de vista es algo que ya existía desde hace mucho tiempo.

Para estructurar el código de la aplicación, nosotros hemos elegido hacerlo a través un modelo por capas. Pero a la vez, nos hemos inspirado en algunos elementos de la arquitectura hexagonal, y aunque no hemos sido fieles a la terminología de la misma, sí que podemos intuir algunos de sus elementos principales. Aun así, como no la hemos seguido estrictamente no podemos decir que la hayamos utilizado.

Para empezar, en las clases correspondientes al dominio, hemos tratado de ser lo más agnósticos posible a la tecnología. Un ejemplo de esto es que no hemos usado Lombok, una tecnología que se utiliza mucho para generar código (getters, setters, builders, etc.). Pero el dominio está acoplado al modelo de base de datos, y depende del framework de persistencia, por tanto no podemos decir que esto se trate de una arquitectura hexagonal. Para desacoplar esto, habría que utilizar

adaptadores, pero no los hemos implementado porque no nos aportaba ninguna ventaja, solo complejidad.

Pueden distinguirse, por el contrario, adaptadores primarios en la capa controlador y puertos secundarios en la capa DAO.

Desde nuestro punto de vista, la arquitectura hexagonal puede considerarse una arquitectura por capas algo vitaminada, pero dibujada de otra forma. Por esta razón enunciamos que no es algo novedoso. Debe utilizarse con criterio y siempre que exista alguna razón de peso para ello, pues de no ser así el código de nuestro proyecto podría convertirse en algo muy complejo innecesariamente. Sigamos la regla KISS siempre que podamos.

8. Disciplina de pruebas

La disciplina de pruebas, es el flujo de trabajo cuyo principal propósito es comprobar el resultado de la implementación probando cada construcción, incluyendo construcciones de versiones intermedias y final que se entregará.

Los objetivos que esta disciplina persigue, consisten en la validación de la implementación llevada a cabo acerca del producto software, ya sea desde el lado funcional (es decir, que los requisitos se cumplen al 100%) como desde parte no funcional (la calidad que el sistema ofrece). Además, mediante las pruebas trataremos de localizar errores en nuestro desarrollo, documentándolos para poder solventarlos posteriormente.

Las actividades recomendadas por RUP para esta disciplina son las siguientes:

1. Planificación de las pruebas de software.
2. Diseño de dichas pruebas.
3. Implementación de las mismas.
4. Realización de pruebas de integración.
5. Realización de pruebas de sistema.
6. Evaluación de las pruebas realizadas.

Durante la ejecución de este proyecto no se ha podido llevar a cabo la Disciplina de Pruebas de forma seria y procedimentada, pues necesitaríamos muchísimo más tiempo y experiencia para aplicar todas las disciplinas al completo.

Aun así, hemos realizado algunas pruebas funcionales de forma manual, siguiendo los distintos caminos marcados en los diagramas de la especificación de los casos de uso que hemos implementado. Consideramos que el conjunto de estas pruebas sumadas a los test unitarios desarrollados en la Disciplina anterior, proporcionan una red de seguridad mínima con la que al menos nos sentimos cómodos al entregar este Proyecto.

Conclusiones y trabajo futuro

Se han cumplido los objetivos propuestos puesto que hemos aplicado RUP al desarrollo de un proyecto real, focalizándonos en las disciplinas de requisitos, análisis y diseño. Se ha hecho menos hincapié en el carácter iterativo ya que no hemos dispuesto del tiempo necesario para realizar un mayor número de casos de uso. Este último punto también ha hecho que la fase de gestión también haya sido simple, aún así de cara a un futuro estas disciplinas se podrían retomar de una manera sencilla.

Puntos positivos que se han observado:

- Uso de vocabulario común, otorgando la importancia que se merece al dominio de la aplicación. Si bien es cierto que no hemos aplicado DDD, el lenguaje ubicuo ha estado presente de principio a fin durante todas las fases del proyecto, cuidándose al máximo la definición y el uso de cada uno de los identificadores empleados. Aunque en un primer momento no fue una tarea del todo sencilla, ya que no estamos acostumbrados a poner tanto detalle en el dominio, nos ha resultado muy beneficioso a la hora de hacernos entender plasmando nuestras ideas tanto en la documentación como en las distintas conversaciones entabladas.
- La obtención y clasificación de casos de uso hacen que sea francamente fácil el asignar tareas a distintas personas del equipo con muy distinto nivel de experiencia, siendo muy fácil poder darle a perfiles junior o senior tareas acordes.
- El haber hecho un análisis de los casos de uso ha hecho que aplicar la técnica de TFD sea realmente sencillo ya que los casos a probar han salido de una manera mucho más natural y sencilla.
- La partición en casos de uso y su posterior análisis han hecho que toda la "lógica" de la funcionalidad se haya pensado y analizado previa al desarrollo consiguiendo de esta manera que a la hora de hacer el código todos los esfuerzos vayan a generar un código de calidad.
- Gracias a la documentación en forma de diagramas es muchísimo más fácil la entrada a un proyecto, ya que desde el día uno la gente dispone de un modelo de dominio y unas bases sobre las que seguir avanzando.
- La planificación una vez hechos los primeros casos de uso es muy sencilla y bastante aproximados los tiempos estimados con los tiempos en los que realmente se ha realizado la tarea.

Líneas futuras y próximos pasos:

- Acabar con la implementación de todos los casos de usos a los que se han llegado en el PFM.
- Realizar un análisis de como todo el proceso que se ha realizado se podría agilizar.
- Mejorar la disciplina de gestión.

A medida que se vayan desarrollando nuevas funcionalidades se podría realizar un estudio de como llevarlo a microservicios si se diesen las condiciones necesarias para su cumplimiento.

Bibliografía

- Kanban proyecto [webpage online]. Available:
https://github.com/zuldare/mastercloud_pfm_halterocms/projects/1
- Github actions [webpage online]. Available:
<https://github.com/features/actions>
- PlantUml [webpage online]. Available: <https://plantuml.com/es/>
- Pencil [herramienta]. Available: <https://pencil.evolus.vn/>