UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA APLICADA

GESTIÓN DE SOCIEDAD DE CAZADORES

PROYECTO FIN DE CARRERA

Autor:

José Antonio Mancebo Requena

Director:

María Carmen Penades Gramage

Julio - 2010

A mis padres, mi hermano y Mª José

Agradecimientos

Me gustaría dar las gracias a mis amigos de toda la vida, a mis amigos de la universidad y a mis compañeros de piso por preocuparse y preguntarme qué tal llevaba el proyecto aunque muchos de ellos no supieran en qué consistía.

Por último quiero dar las gracias a mi tutora, por guiarme a lo largo del proyecto y darme siempre ánimos.

Tabla de contenidos

Lista de figuras

Capítulo 1:

Introducción

Este capítulo introductorio sirve para aclarar de forma concisa los aspectos generales del proyecto, tales como la motivación por la que se realiza este proyecto, los objetivos principales y la organización de la presente memoria comentando brevemente los apartados que la componen.

1.1. Motivación

He decidido realizar este proyecto sobre la gestión de una sociedad de cazadores en primer lugar porque quería sentirme capacitado para realizar el proceso de desarrollo de software de una aplicación por mi mismo tras haber trabajado satisfactoriamente en grupo a lo largo de la carrera.

Y en segundo lugar porque conozco familiares y amigos que practican la caza y sé que en las sociedades de cazadores donde participan no tienen informatizada su gestión y sé que dicha gestión no es tan simple como a priori parece. Deben llevar la

gestión de diversos aspectos a lo largo de las temporadas que organizan: las cuotas que van a pagar los socios, las asambleas que se convocan, las actividades y campeonatos que se realizan, los ingresos y gastos que se producen, etc. Por otro lado también tiene que llevar la gestión de los socios que participan en la sociedad: las modalidades de caza que practican en cada temporada, el estado del pago de las cuotas, etc.

En definitiva me atraía el reto de lograr una gestión más sencilla y cómoda de la práctica de esta actividad tradicional que tanto a evolucionado en los últimos tiempos.

1.2. Objetivos

El objetivo principal de este proyecto es el desarrollo de un sistema de información para la gestión y mantenimiento de una sociedad de cazadores. Centrándose en la gestión de los socios que componen la sociedad de cazadores así como la configuración y seguimiento de las diferentes temporadas que soporta. A continuación se especifican los objetivos más importantes establecidos.

- Gestionar los datos que identifican la sociedad de cazadores.

- Gestionar la información de los socios que han pertenecido a la sociedad, haciendo énfasis en la gestión de las cuotas y de las modalidades de caza que practica cada socio en las diferentes temporadas en las que participa.

- Gestionar las actividades que se organizan por la sociedad las cuales se subdividen en actividades de sociedad, campeonato de cazadores y campeonato de perros.

- Gestionar las actuaciones que se realizan en el coto de la sociedad. Dichas actuaciones pueden ser mejoras para conseguir un buen mantenimiento de las especies cazables o repoblaciones que se hagan de dichas especies por su precario número de población.

- Gestionar la contabilidad de la sociedad, los ingresos y gastos que se producen a lo largo de cada temporada.

- Gestionar las juntas directivas que administran la sociedad de cazadores, tanto su composición como las comisiones que establecen.

A nivel académico, el objetivo del proyecto es realizar una aplicación de escritorio realizada bajo el entorno de desarrollo Visual Studio. Se contará con una

base de datos para almacenar toda la información que la aplicación deberá gestionar. Para ello se deberá de cubrir los siguientes objetivos secundarios:

- Seguir las pautas principales que marca la Metodología RUP.
- Realizar una implementación siguiendo una arquitectura de 3-capas.
- Utilizar el lenguaje de programación C#.
- Trabajar con una base de datos SQL.

1.3. Estructura de la memoria

A continuación se describe brevemente cada uno de los capítulos que componen esta memoria:

- Capítulo 1. Introducción.

  En este capítulo se realiza la presentación del proyecto, se explica cual es la motivación por la cual se lleva a cabo, así como los principales objetivos que se desean lograr a través de él.

- Capítulo 2. Metodología.

  Este capítulo comienza realizando la comparativa entre la metodología tradicional RUP y la metodología ágil XP. A continuación se explican las razones por las cuales se realiza el proyecto bajo la metodología RUP. Para finalizar se define el caso de estudio a resolver.

- Capítulo 3. Modelado conceptual.

  Este capítulo empieza explicando lo que se entiende por modelado conceptual y se define el lenguaje UML. Tras ello se muestran los principales modelos de UML: diagramas de casos de uso, diagrama de clases y diagramas de secuencia asociados al proyecto.

- Capítulo 4. Arquitectura del sistema.

  Este capítulo comienza definiendo el concepto de arquitectura del sistema. Después se explica la arquitectura cliente/servidor de tres capas que es en la que se ha desarrollado el sistema. Para finalizar se muestra como se adapta al proyecto.

- Capítulo 5. Diseño.

  En este capítulo se muestran los diseños de las tres capas que componen la arquitectura del sistema del proyecto: diseño de IGU (capa de presentación), diseño de clases (capa de negocio o lógica) y diseño de base de datos (capa de persistencia).

- Capítulo 6. Implementación.

  Este capítulo empieza explicando la tecnología bajo la que se ha desarrollado la aplicación, el entorno de desarrollo Microsoft Visual Studio 2008 de la plataforma .NET y el lenguaje C#. A continuación se explica cómo se realiza el acceso a datos en esta tecnología mediante el escenario desconectado. Para finalizar se muestran detalles de implementación separados por capas.

- Capítulo 7. Conclusiones.

  En este capítulo se realiza la valoración del proyecto, los objetivos conseguidos y las posibles evoluciones que se pueden realizar del mismo.

  A continuación se enumeran las referencias bibliográficas y finalmente se muestran 3 anexos:

- Bibliografía.

- Anexo A. Plantillas de casos de uso.

  En este anexo se muestran las plantillas de los casos de uso.

- Anexo B. Manual de instalación.

  En este anexo se explica cómo se realiza la instalación de la aplicación.

- Anexo C. Manual de usuario.

  En este anexo se explica cómo funciona la aplicación.

Capítulo 2:

Metodología

En primer lugar la pregunta que hay que hacerse es ¿Cuál es la metodología orientada a objetos que más conviene para realizar el proyecto?. La metodología tradicional por excelencia, RUP (Rational Unified Process) o la metodología ágil más popular, XP (Extreme Programing).

Para poder elegir la metodología que mejor se puede adaptar a este proyecto a continuación se definen los fundamentos principales de ambas metodologías y se realiza una tabla comparativa entre ambas.

El **RUP** reúne las tres metodologías de desarrollo basadas en el paradigma de la orientación a objetos:

- OOSE: Object Oriented Software Engineering, Ivar Jacobson (Casos de uso).
- OMT: Object Modeling Technique, James Rumbaugh (Análisis).
- Booch, Grady Booch (Diseño).

Es uno de los procesos más generales de los existentes actualmente, ya que está pensando para adaptarse a cualquier proyecto. Está basado en componentes, lo que quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Utiliza el lenguaje UML para expresar gráficamente todos los diagramas que definen el sistema software.

Los aspectos que caracterizan al RUP son tres:

- Dirigido por casos de uso. Una interacción con el usuario es un caso de uso, que captura los requisitos funcionales. Conducen el proceso de desarrollo mediante modelos de diseño e implementación, creados por los desarrolladores, y mediante pruebas. Además estos modelos evolucionan junto a la arquitectura del sistema y se validan para que sean conformes a los caso de uso.

- Centrado en la arquitectura: El concepto de la arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema tales como: plataforma (BBDD, SO, protocolo de comunicación,…), aspectos legales, componentes reutilizables disponibles, requisitos no funcionales, etc. Es una vista del diseño completo que hace visibles las características principales.

- Iterativo e incremental: todo sistema informático es complejo suele durar desde meses hasta años. Por lo que lo más eficiente es dividir el proyecto en pequeñas fases. Al terminar cada fase hay que revisarla y probarla. Las fases se dividen en iteraciones, y cada iteración se basa en la anterior, por lo que se produce un incremento, que no siempre es aditivo.

El ciclo de vida del software se divide en 4 fases, cada una de las cuales tiene varias iteraciones. Una iteración representa un ciclo de desarrollo completo que consta de: requisitos, análisis, diseño, implementación y pruebas. El énfasis en cada flujo de trabajo es diferente dependiendo de la fase en que se encuentre. Las fases son:

- El inicio es definir el alcance del proyecto, definir casos de uso y riesgos.
- La elaboración es proyectar un plan, definir las características y cimentar la arquitectura.
- La construcción es crear el producto.
- La transición es transferir el producto a sus usuarios.

En la figura 1 se puede ver gráficamente la estructura y la distribución del flujo de trabajo.

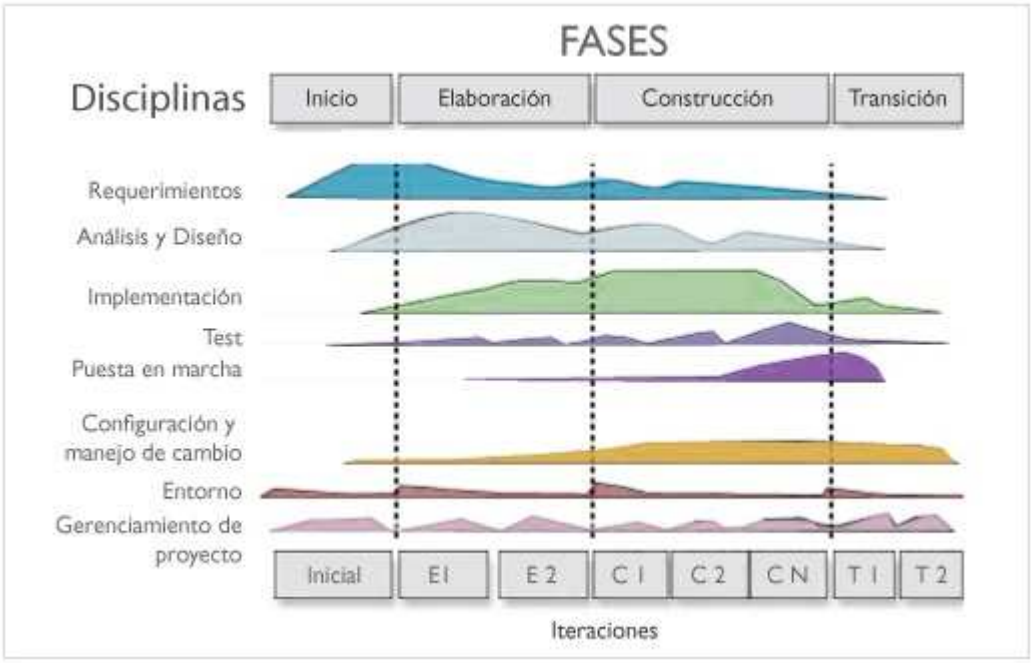

Figura 1. Estructura de RUP

La **XP** es una de las "Metodologías Ágiles" de desarrollo de software más populares hoy en día. Inicialmente fue descrita por Kent Beck cuando trabajaba en la Chrysler Corporation. XP es una metodología donde lo más importante son los individuos y las interacciones entre ellos, siendo clave para el éxito en desarrollo de software, y no las herramientas y los procesos empleados. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

XP define unos valores fundamentales que son los siguientes:

- Comunicación fluida entre todos los participantes.
- Simplicidad en las soluciones implementadas
- Retroalimentación (feedback).
- Coraje para enfrentar los cambios.
- Respeto entre los miembros del equipo y del trabajo que realizan.

La XP agrupa 13 prácticas básicas que se deben cumplir para asegurar el éxito del proyecto. Se pueden ver los nombres originales en la figura:

- Equipo completo.
- Planificación continúa.
- Test del cliente.
- Versiones pequeñas.

- Diseño simple.
- Programación de parejas.
- Desarrollo guiado por pruebas automáticas.
- Mejora continua del diseño.
- Integración continúa.
- El código es de todos.
- Normas de codificación comunes.
- Utilizar metáforas.
- Mantener un ritmo sostenible.

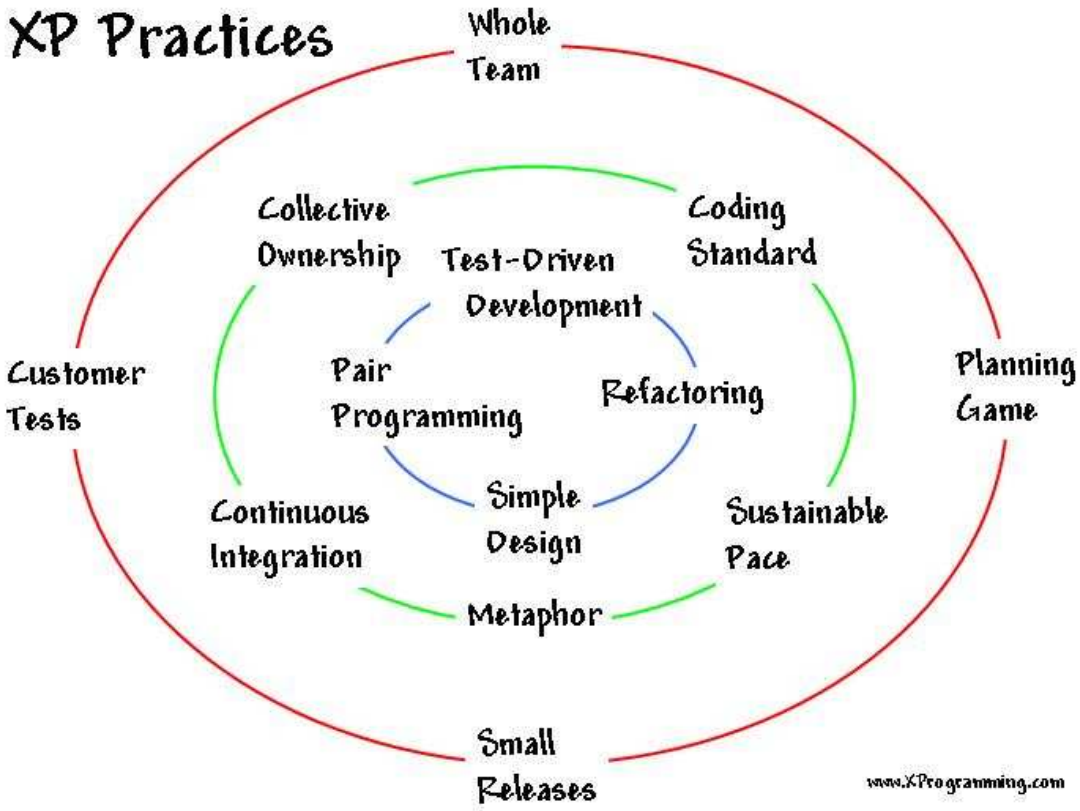

Figura 2. Prácticas de XP

La idea principal de esta metodología consiste en trabajar estrechamente con el cliente, haciéndole pequeñas versiones frecuentemente. El objetivo de estas versiones es conseguir que funcione de forma más simple y con el mínimo código. Cuando el cliente le transmite al equipo de desarrollo lo que quiere, el equipo de desarrollo puede hacer una estimación del tiempo que va a tardar, pero, lógicamente, la planificación deberá revisarse y modificarse continuamente a lo largo del proyecto. Cada vez que se consigue codificar y que funcione una versión de usuario, se le muestra al cliente para la que la pruebe y añada las posibles modificaciones que crea

convenientes. De esta forma se evita perder tiempo desarrollando una aplicación que no sea la que el cliente esperaba. Este ciclo se va repitiendo una y otra vez hasta que el cliente se dé por satisfecho y cierre el proyecto.

La tabla comparativa entre ambas metodologías es la siguiente.

RUP	XP
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software
Menor dificultad para delimitar el alcance del proyecto con nuestro cliente	Mayor dificultad para delimitar el alcance del proyecto con nuestro cliente

Figura 3. Comparativa de RUP y XP

Tras haber estudiado las características de ambas metodologías y viendo como encajan en el proyecto. El proyecto se decide realizar siguiendo la metodología RUP por los siguientes motivos:

- Cuando se comienza el proyecto ya se conoce los requisitos que va a tener que cubrir, no van a sufrir cambios a lo largo del proceso. Por lo que se puede definir la arquitectura del sistema de forma sólida en las primeras iteraciones del mismo.

- Al trabajar una sola persona, es preferible que se te marque un camino a seguir mediante el cual conseguir una buena documentación en la que apoyarse.

- El cliente, en este caso la directora del proyecto, no es parte del equipo de desarrollo sino que interactúa con el equipo de desarrollo, el autor del proyecto, mediante reuniones.

En este proyecto se ha realizado un proceso RUP reducido mediante el cual se debe resolver el caso de estudio definido a continuación.

2.1. Caso de estudio

Se desea desarrollar un sistema informático para gestionar una sociedad de cazadores. Para hacer que dicho caso de estudio sea coherente se ha obtenido información del funcionamiento de sociedades de cazadores a través de internet, las principales fuentes han sido las páginas web de la sociedad de cazadores el Raigous (www.elraigosu.es), la sociedad de cazadores Ecija (www.cazaecija.com) y la sociedad de cazadores San Santurio (www.sociedadsansaturio.com). La especificación que se ha dado del problema es la que sigue.

De la sociedad de cazadores se quiere mantener la siguiente información: CIF, nombre, logotipo, dirección, localidad, provincia y código postal donde se encuentra, así como la cuenta corriente donde se llevara a cabo la gestión contable.

Dicha sociedad de cazadores quiere realizar un seguimiento de:

- El mantenimiento de aspectos necesarios para administrar el resto del sistema.
- Los socios que la integran.
- Las personas no socias que participan en alguna actividad de la misma.
- Los perros que utilizan los socios para cazar o bien los perros que participan en campeonatos organizados por la misma.
- Las juntas directivas que la han regido.
- Las diferentes temporadas que se han afrontado.

Los aspectos que son necesarios para poder administrar el resto del sistema tienen como uno de los campos que lo define el campo estado, el cual puede ser “ACTUAL”, aspecto que está vigente en el sistema, o “HISTORIAL”, aspecto del que se quiere tener constancia aunque actualmente no es necesario. Dichos aspectos son los siguientes:

- Los diferentes tipos de socios en los que la sociedad clasifica a sus socios. De cada tipo de socio se quiere conocer su nombre, descripción, en la que se puede definir el rango de edades que lo comprende así como los derechos y obligaciones que tienen.

- Las modalidades de caza que los socios pueden practicar. Cada modalidad de caza está definida con un nombre, una descripción breve en la que se explique en qué consiste y el tipo de caza a la que pertenece, caza mayor y caza menor.

- El coto de caza de la sociedad del cual se mantiene la siguiente información: el código, el nombre, el número de hectáreas que ocupa y la descripción del mismo, aspecto que no tiene el campo estado. De él también se quiere

saber las zonas en que se divide. De cada zona se quiere guardar su nombre, número de hectáreas, descripción así como el estado.

- Las especies cazables que se pueden cazar. Dichas especies de caza están definidas por su nombre común, nombre científico (opcional), descripción y si se caza en caza menor o en caza mayor.

De toda persona participante en la sociedad se quiere conocer su nombre y apellidos, DNI, teléfono y correo electrónico (opcional).

Además de los socios se quiere conocer el número que lo va identificar dentro de la sociedad, la dirección, localidad, provincia y código postal donde vive, la fecha de nacimiento y el tipo de socio al que pertenece. Además se quiere saber si el socio tiene el pago de las cuotas domiciliado o no, si lo tiene, se debe guardar la información del número de la cuenta corriente donde se realizará. De cada socio también se quiere conocer el estado en el que se encuentra, "ACTUAL", participa en la sociedad actualmente o, "HISTORIAL", ha participado en la sociedad y si quisiera volver en un futuro no debería volver a pagar la cuota de entrada.

Tanto de los socios como de los participantes no socios cuyos perros participan en la sociedad también se quiere llevar el control, de ellos se quiere guardar su número de identificación, su nombre, su especie, su raza, su sexo y su fecha de nacimiento.

Los socios pueden formar parte de la junta directiva que rige la sociedad en los cargos de: presidente, vicepresidente, secretario, tesorero o vocal de una de las comisiones que existen en la sociedad. Una comisión se encargará de la gestión de uno de los apartados en los que se divide la gestión de la sociedad, la cual podrá tener como responsable a uno o más vocales que se pueden apoyar en la ayuda de otros socios denominados colaboradores.

De cada una de las juntas directivas que han formado parte de la sociedad también se quiere conocer su fecha de constitución. Una junta directiva se encargara de gestionar un conjunto de temporadas. Una temporada se identifica por un nombre, por ejemplo, temporada 2009/2010, por su año inicio y el estado en que se encuentra, si esta "EN CONFIGURACIÓN", quiere decir que es el periodo de la temporada en el que se establece las cuotas que los socios deberán pagar y las modalidades de temporada que van a practicar. Una vez finalizada esta etapa, el estado de la temporada cambiará a "EN CURSO", etapa en la que se podrá gestionar las cuotas y modalidades de temporada que los socios eligen, las actividades y actuaciones que va realizar la sociedad así como los movimientos contables que genera la sociedad. Cuando la temporada ha finalizado su estado cambiara a "FINALIZADA".

De cada cuota que se determina para la temporada se quiere conocer su nombre y los precios que tiene para cada uno de los tipos de socios actuales. Las cuotas se dividen en cuotas de entrada, de sociedad y de modalidad.

Las cuotas de entrada son las que están obligadas a pagar los socios de nuevo ingreso. Toda temporada tiene que tener una cuota de entrada.

Las cuotas de sociedad son las que hacen referencia al pago del alquiler del local de la sociedad, el mantenimiento del mismo, la derrama para hacer una reforma, etc. Todas las cuotas de sociedad que se determinan en una temporada son de obligado pago por los socios.

Las cuotas de modalidad son las que se asocian a las modalidades que se establecen para cada temporada. De cada una de las modalidades de temporada se quiere conocer su nombre así como el conjunto de modalidades de caza que la componen. Una vez finalizada la etapa de configuración de la temporada el socio podrá determinar las modalidades que desea practicar.

Cada socio tendrá cada temporada una cuota de socio, la cual estará compuesta por un conjunto de cuotas establecidas en dicha temporada. La cuota de entrada, si el socio es de nuevo ingreso, las cuotas de sociedad determinadas, así como las cuotas de modalidad que están relacionadas con las modalidades que el socio ha decidido practicar. El total de dicha cuota a pagar por un socio dependerá de su actual tipo de socio.

Otra de las gestiones que se llevan a cabo en una temporada, es la gestión de las actividades. De cada actividad queremos guardar su nombre, fecha inicio, lugar de realización, descripción, periodo de preinscripción, la cuota que de inscripción del participante socio así como la cuota de inscripción para el participante no socio si se determina que pueden participar y el número máximo de participantes si existiera. Las actividades se dividen en actividades de sociedad, campeonatos de cazadores y campeonatos de perros.

Las actividades de sociedad son las que están relacionadas con la realización de una jornada de tiro al pichón libre, una cena de los miembros de la sociedad o una excursión. De cada participante que se inscribe en una actividad se quiere conocer su D.N.I., nombre, apellidos, teléfono y correo electrónico (opcional).

Los campeonatos de cazadores son las actividades que hacen referencia a competiciones entre cazadores de la cual se quiere gestionar su clasificación. De cada participante que se inscribe en un campeonato se quiere conocer su D.N.I., nombre, apellidos, teléfono, correo electrónico (opcional), dirección, localidad, provincia y código postal donde vive, fecha de nacimiento y si participan participantes no socios,

se quiere conocer el nombre de la sociedad de cazadores a la que pertenece (si pertenece alguna)y por supuesto la clasificación que logre.

Los campeonatos de perros llevan una gestión similar a los campeonatos de cazadores, la diferencia es que la competición es entre perros, como puede ser un campeonato de carreras de galgos. De cada perro que participe se quiere conocer a parte los datos del participante dueño que se conocen en un campeonato de cazadores los datos identificativos del perro, número de identificación, nombre, raza, sexo, fecha nacimiento y la clasificación que logre.

Tanto la decisión de que actividades se va a realizar como otras muchas decisiones se debaten y deciden en las asambleas que convoca la junta directiva a lo largo de la temporada, al menos debe convocar una para informar a los socios de cómo ha ido la gestión contable cuando finaliza la temporada, cada asamblea debe tener un nombre, fecha y lugar donde se va realizar y los puntos del día que se van a tratar, los cuales están definidos por un número y una descripción. Cada uno de los puntos tratados en la asamblea tendrá un acta en la cual se indicara los votos a favor, votos en contra, votos nulos y votos en blanco.

También se quiere tener constancia de las actuaciones en el coto de caza que realiza la sociedad en una temporada. De cada actuación se quiere saber su número (generado por el sistema e identificador de ella), nombre y descripción, en qué consiste y si afecta al coto, a que zonas afecta en concreto. Una actuación puede ser de mejora, colocación de bebederos, suministro de alimentos para los animales o la colocación de cajas-nido. O puede ser una repoblación de una especie cazable, de la especie cazable a repoblar se quiere conocer el número de ejemplares.

Además se quiere tener constancia de las vedas que las instituciones pertinentes establecen para la temporada, de ellas se quiere conocer su nombre, su fecha de inicio y de fin y su descripción.

Para finalizar con la gestión que se realiza de cada temporada, se quiere mantener la información de los movimientos contables que ha generado la sociedad. Se quiere tener conocimiento de todos los gastos e ingresos que se van produciendo a lo largo de la temporada. Cada movimiento es definido por el concepto del gasto o ingreso, la fecha en que se realizó y el importe del mismo.

Capítulo 3:

Modelado Conceptual

¿Por qué modelamos? Porque necesitamos construir modelos para comprender mejor el sistema que estamos desarrollando, los modelos nos:

- Visualizan como es o queremos que sea un sistema.
- Especifican la estructura o el comportamiento de un sistema.
- Proporcionan plantillas que nos guían a la construcción de un sistema
- Documentan las decisiones que hemos adoptado.

El modelado conceptual que se ha realizado es orientado a objetos (O.O.) que se define como *"Un proceso que examina los requisitos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema"* (Booch G. , 1994), cuyas principales características son:

- Próximo a los mecanismos cognitivos humanos.
- Desarrollo incremental bajo una noción común de objeto.

Las actividades del modelado nos van a permitir conseguir definir la estructura y comportamiento del sistema:

- Definición del problema y recogida de datos.
- Identificación de las clases y la estructura del dominio del problema.
- Identificación del comportamiento de los objetos.
- Identificación de los patrones de interacción entre objetos.
- Integrar y revisar Diagramas (redefinir si es necesario).

Este sistema se ha modelado atreves del lenguaje UML (Unified Modeling Language) el cual se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables (www.wikipedia.org).

3.1. Casos de uso

Los diagramas de casos de uso documentan el comportamiento de un sistema desde el punto de vista del usuario. Por tanto los casos de uso determinan los requisitos funcionales del sistema. Se pueden usar durante las siguientes fases del desarrollo:

- Captura de requisitos.
- Planificación de iteraciones de desarrollo.
- Validación del sistema.

Su ventaja principal es la facilidad para interpretarlos, lo que hace que sean especialmente útiles en la comunicación con el cliente.

Estos diagramas permiten una representación gráfica de las interacciones entre actores (usuarios o aplicaciones externas que podrán demandar la utilización de funciones ofrecidas por el sistema) y casos de uso (forma concreta de utilizar parte de la funcionalidad del sistema).

A continuación se muestran los diagramas de casos de uso del proyecto desarrollados con la herramienta Rational Rose 98 (IBM Rational Rose). Además se

muestran las plantillas de descripción de los casos de uso, dos a modo de ejemplo en presente apartado y el resto en el anexo A.

Diagrama de caso de uso de contexto

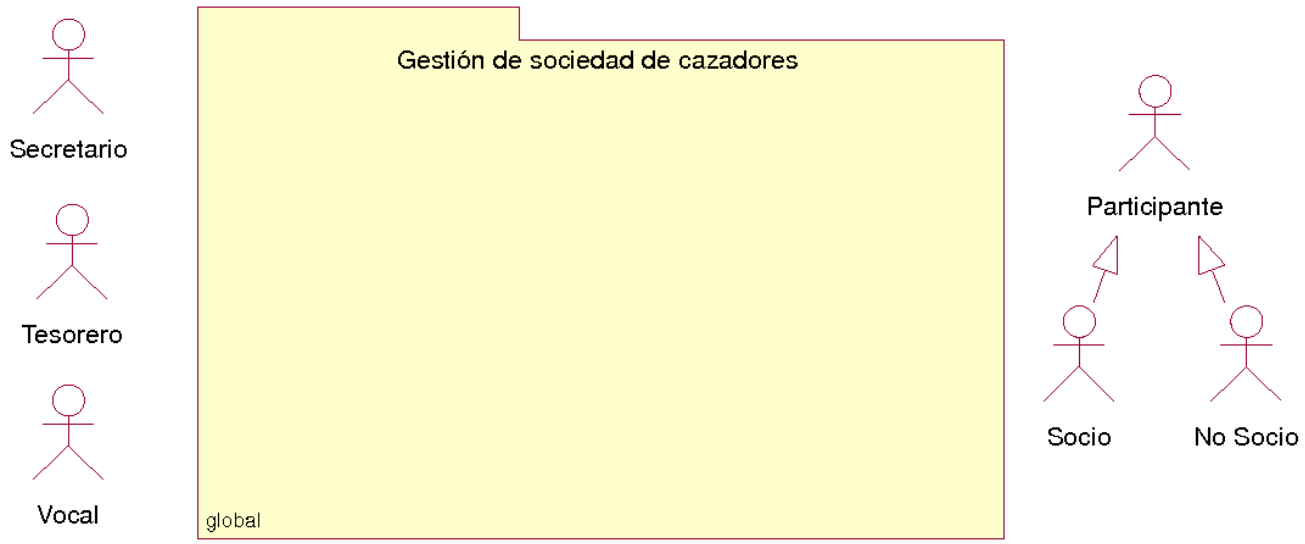

Figura 4. Diagrama de contexto.

Diagrama de caso de uso inicial

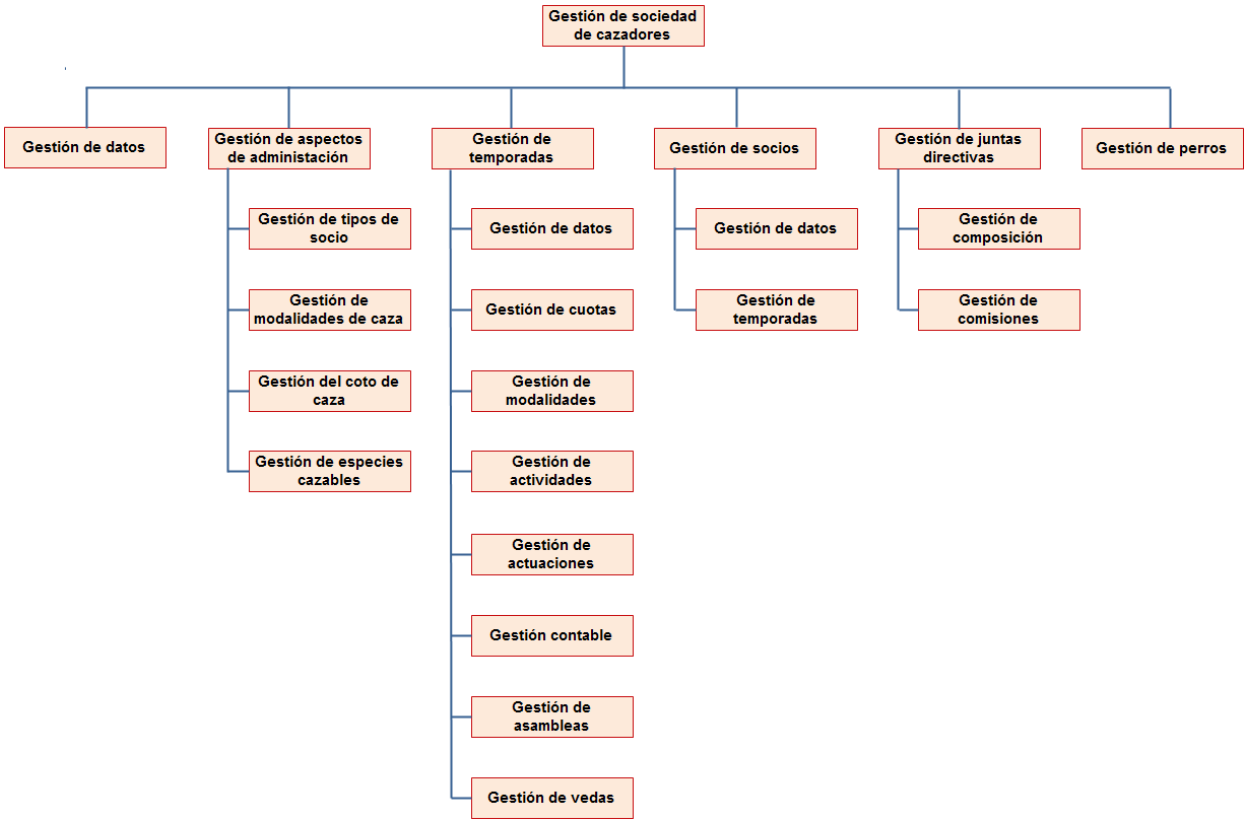

Figura 5. Diagrama inicial

Diagrama de caso de uso del subsistema gestión de datos de sociedad de cazadores

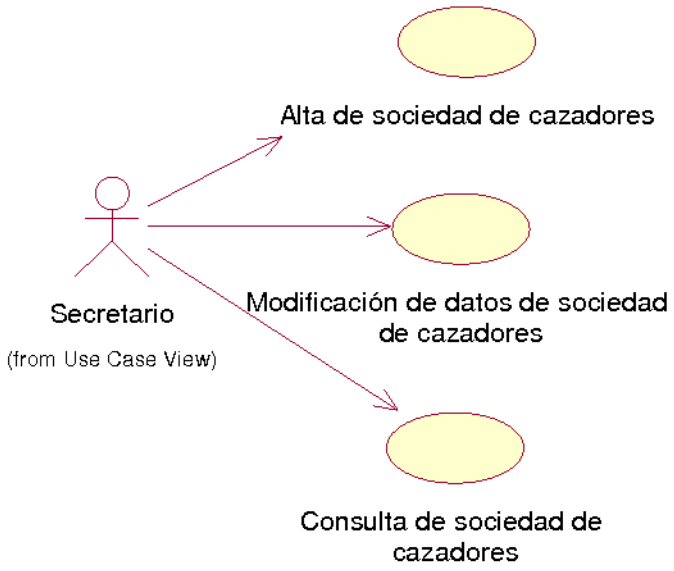

Figura 6. Diagrama de gestión de datos de sociedad de cazadores.

Diagrama de caso de uso del subsistema gestión de aspectos de administración

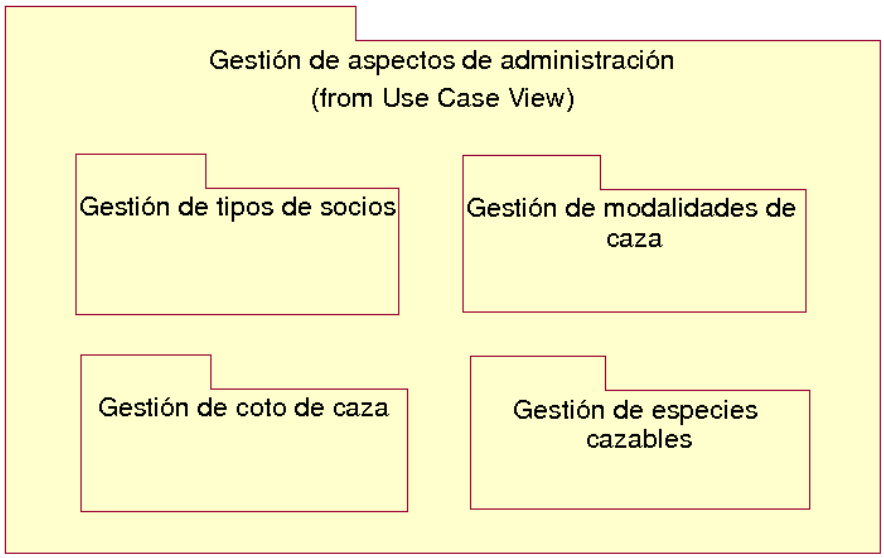

Figura 7. Diagrama de gestión de aspectos de administración.

Diagrama de caso de uso del subsistema gestión de tipos de socio

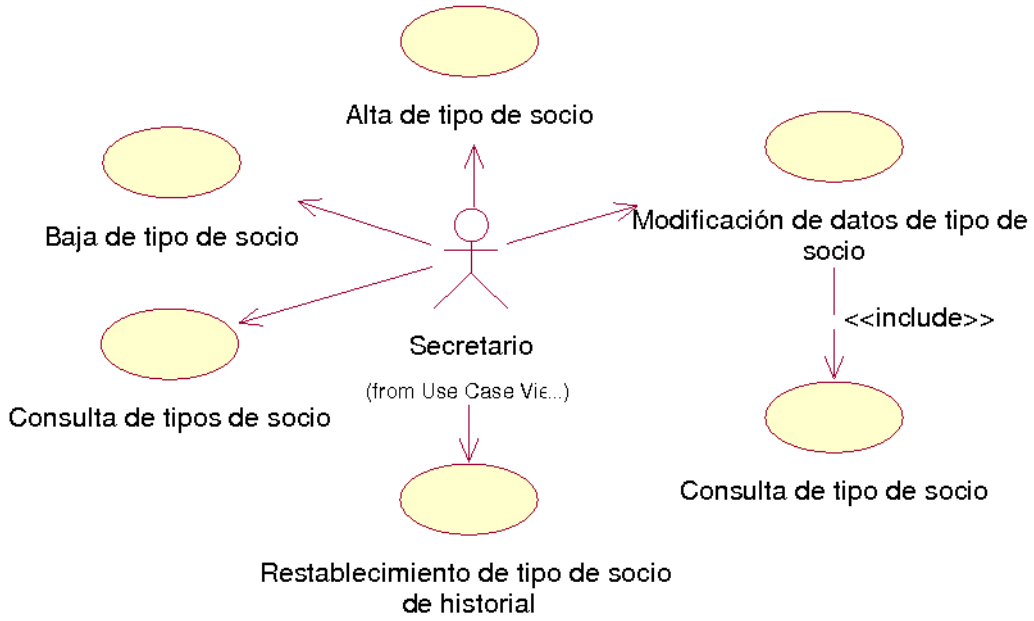

Figura 8. Diagrama gestión de tipos de socio.

Diagrama de caso de uso del subsistema gestión de modalidades de caza

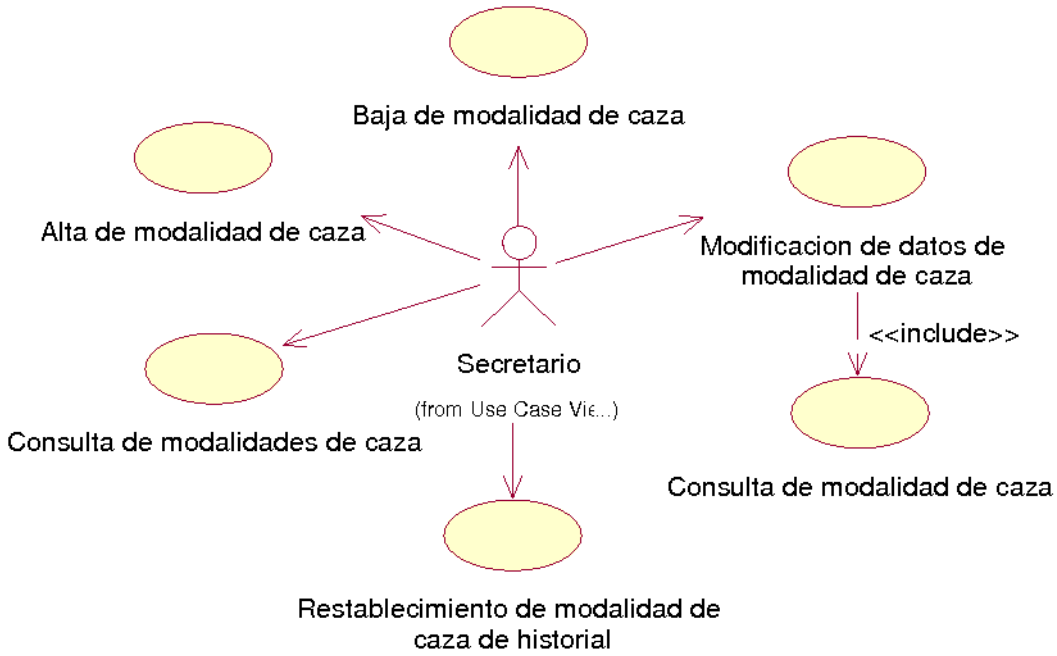

Figura 9. Diagrama gestión de modalidades de caza.

Caso de uso	Baja de modalidad de caza.
Actores	Secretario.
Propósito	Dar de baja una modalidad de caza.
Resumen	El secretario da de baja una modalidad de caza.
Precondiciones	Se ha realizado el caso de uso Consulta de modalidades de caza mostrando las que son actuales.
Postcondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja una modalidad de caza.	
2. El secretario selecciona la modalidad de caza a dar de baja.	
3. El secretario solicita dar de baja la modalidad de caza seleccionada.	4. Busca la modalidad de caza seleccionada.
	5. Comprueba que la modalidad de caza no ha pertenecido a ninguna modalidad de temporada.
	6. Si la modalidad de caza no ha pertenecido a ninguna modalidad de temporada, elimina la modalidad de caza del sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 5 la modalidad de caza ha pertenecido alguna modalidad de temporada.	
	1.1. Modifica el estado de la modalidad de caza a “HISTORIAL” y lo registra en el sistema.
	1.2. Muestra mensaje de información.
Extensiones asíncronas	
Ninguna.	

Diagrama de caso de uso del subsistema gestión de especies cazables

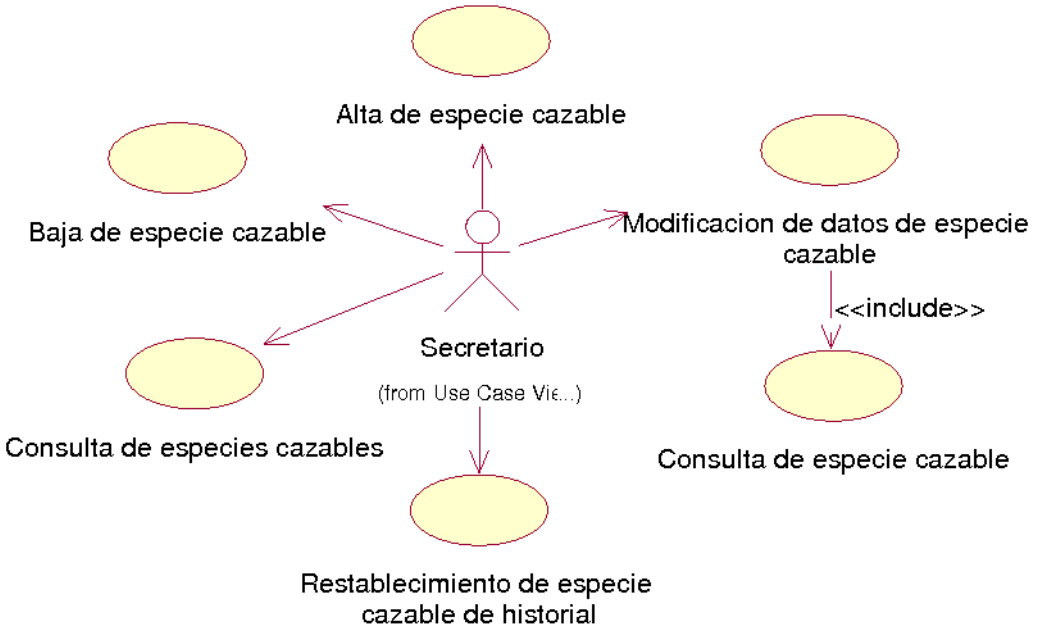

Figura 10. Diagrama gestión de especies cazables.

Diagrama de caso de uso del subsistema gestión de coto de caza

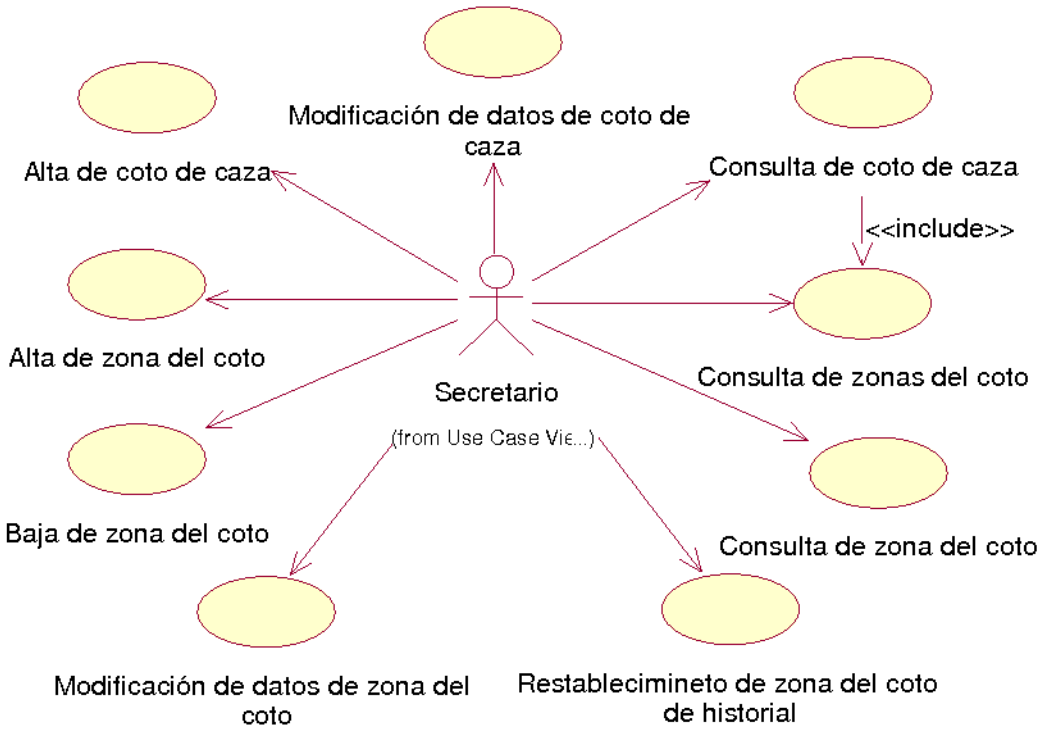

Figura 11. Diagrama gestión de coto de caza.

Diagrama de caso de uso del subsistema gestión de temporadas

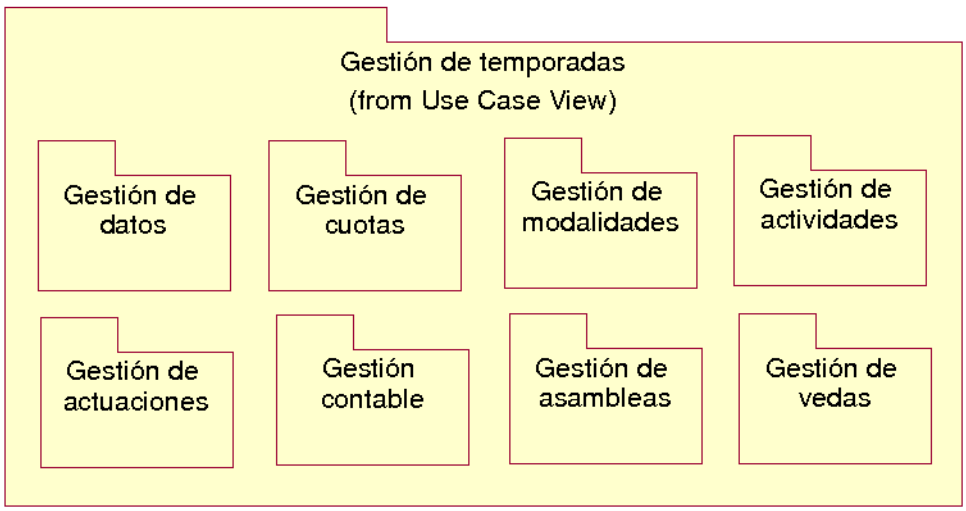

Figura 12. Diagrama gestión de temporadas.

Diagrama de caso de uso del subsistema gestión de datos de temporada

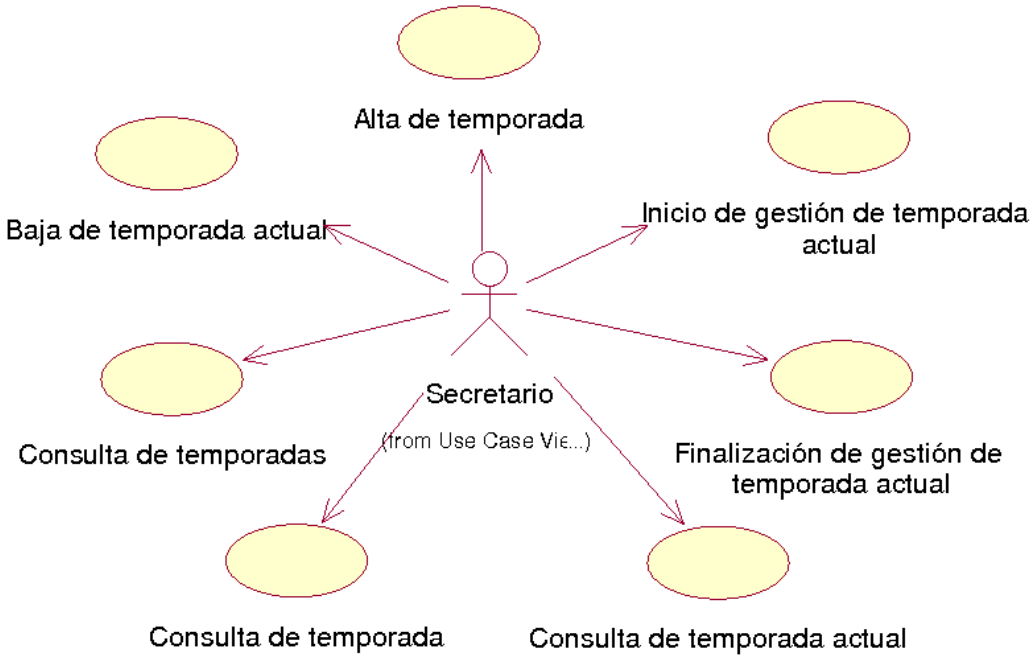

Figura 13. Diagrama gestión de datos temporada.

Diagrama de caso de uso del subsistema gestión de cuotas

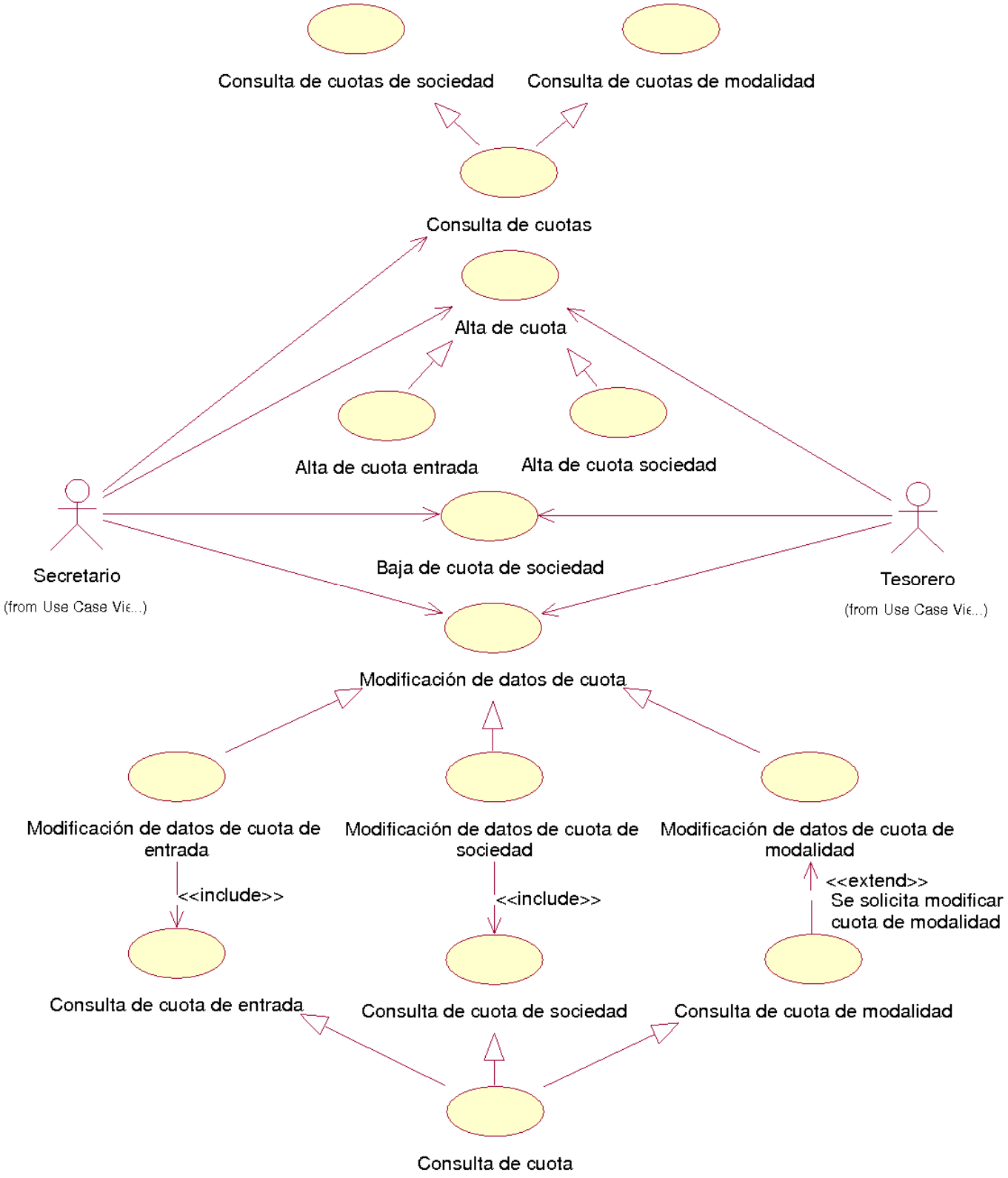

Figura 14. Diagrama gestión de cuotas.

Diagrama de caso de uso del subsistema gestión de modalidades de temporada

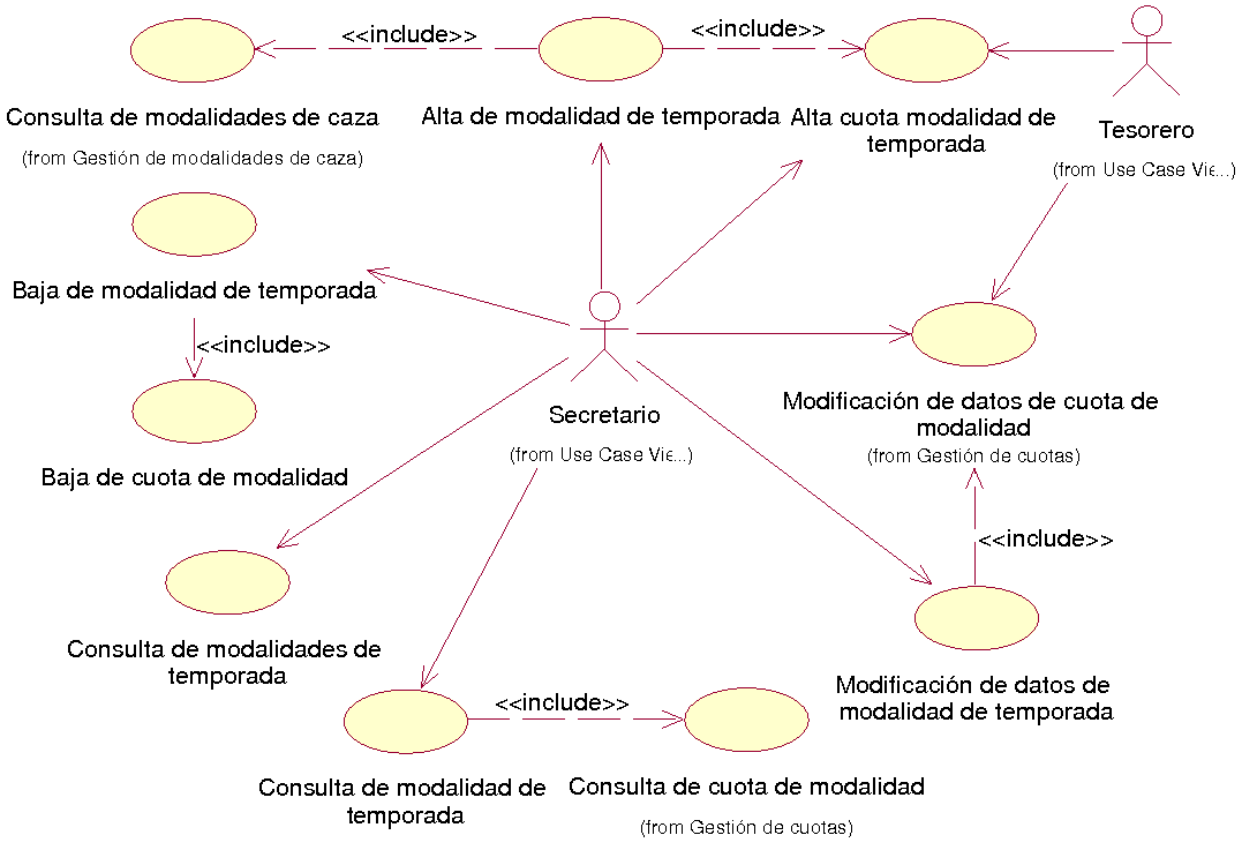

Figura 15. Diagrama gestión de modalidades de temporada.

Diagrama de caso de uso del subsistema gestión de actividades

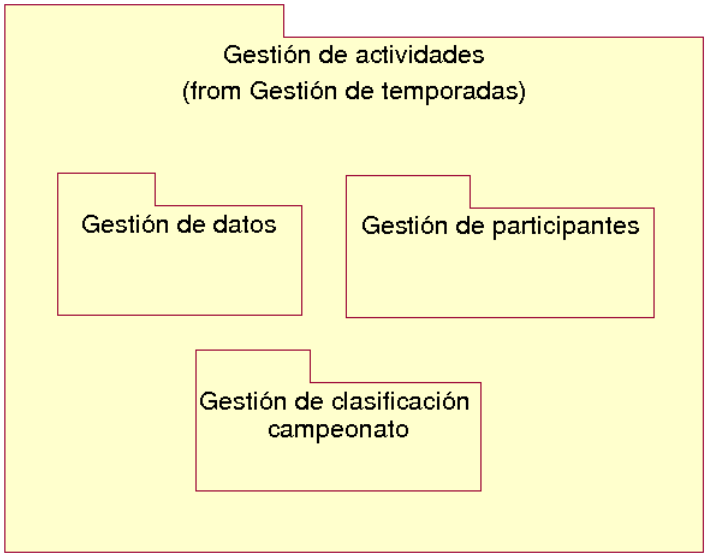

Figura 16. Diagrama gestión de actividades.

Diagrama de caso de uso del subsistema gestión de datos de actividades

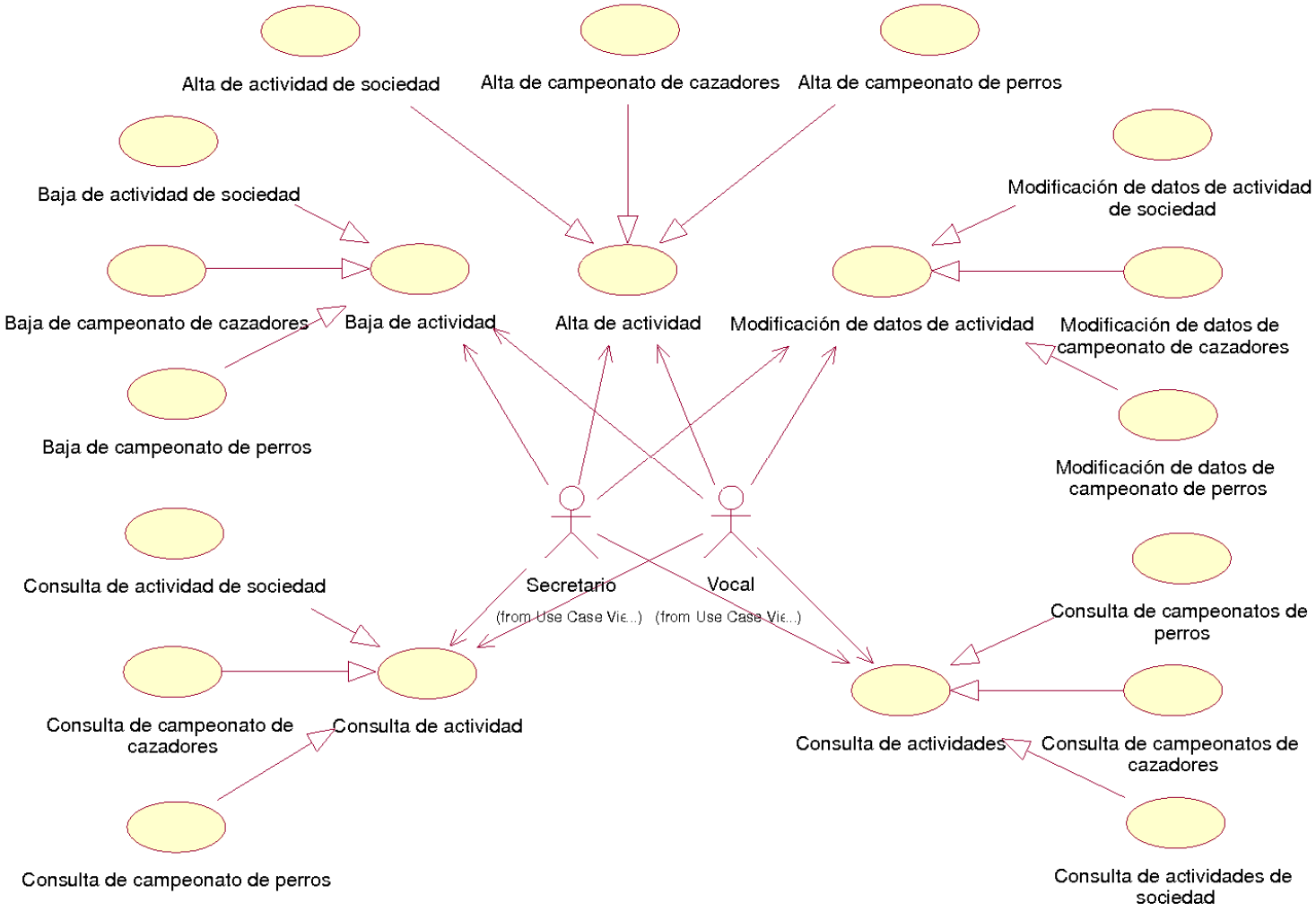

Figura 17. Diagrama gestión de datos actividades.

Diagrama de caso de uso del subsistema gestión de clasificación de campeonatos

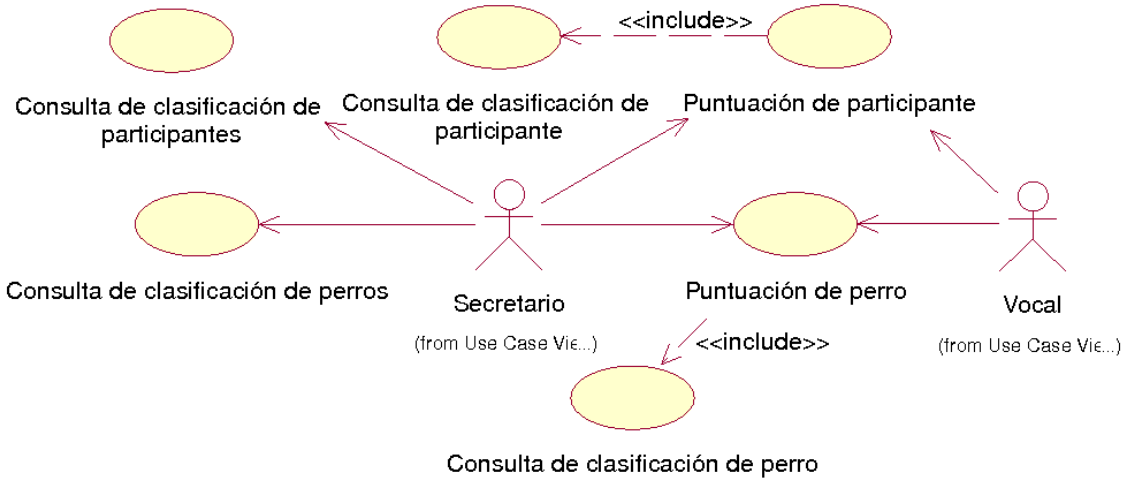

Figura 18. Diagrama gestión de clasificación de campeonatos.

Diagrama de caso de uso del subsistema gestión de participantes

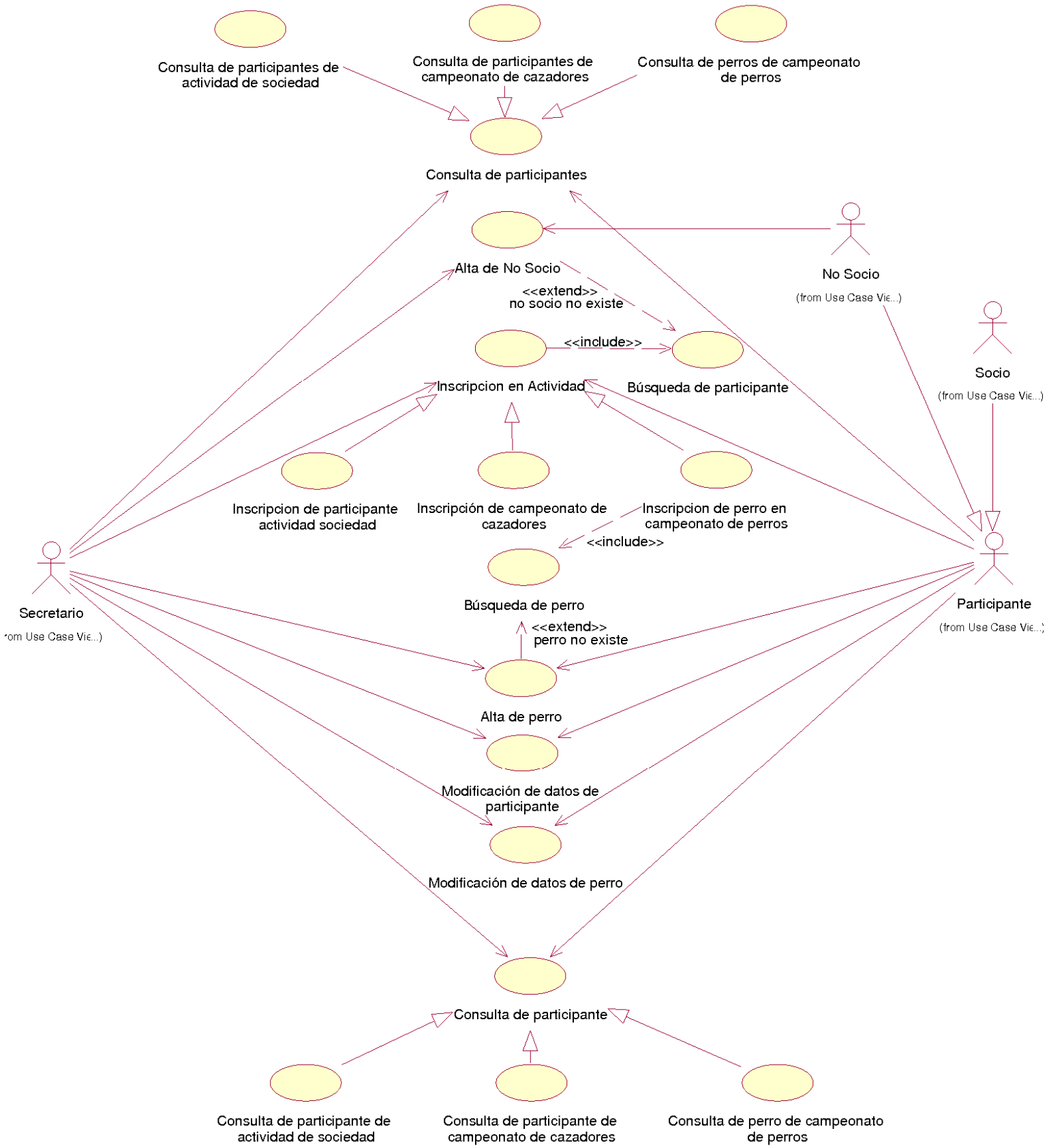

Figura 19. Diagrama gestión de participantes.

Diagrama de caso de uso del subsistema gestión de actuaciones

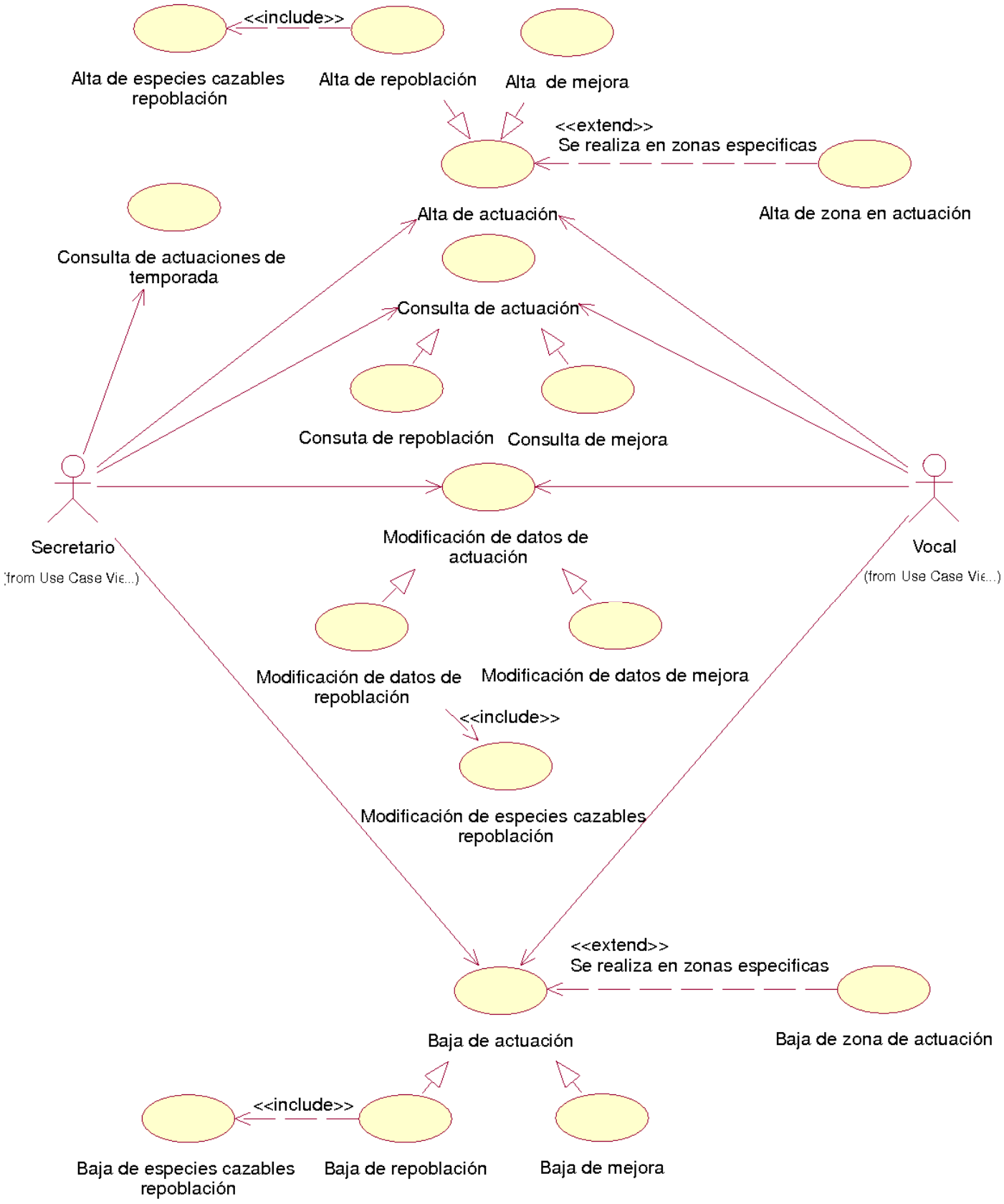

Figura 20. Diagrama gestión de actuaciones.

Diagrama de caso de uso del subsistema gestión contable

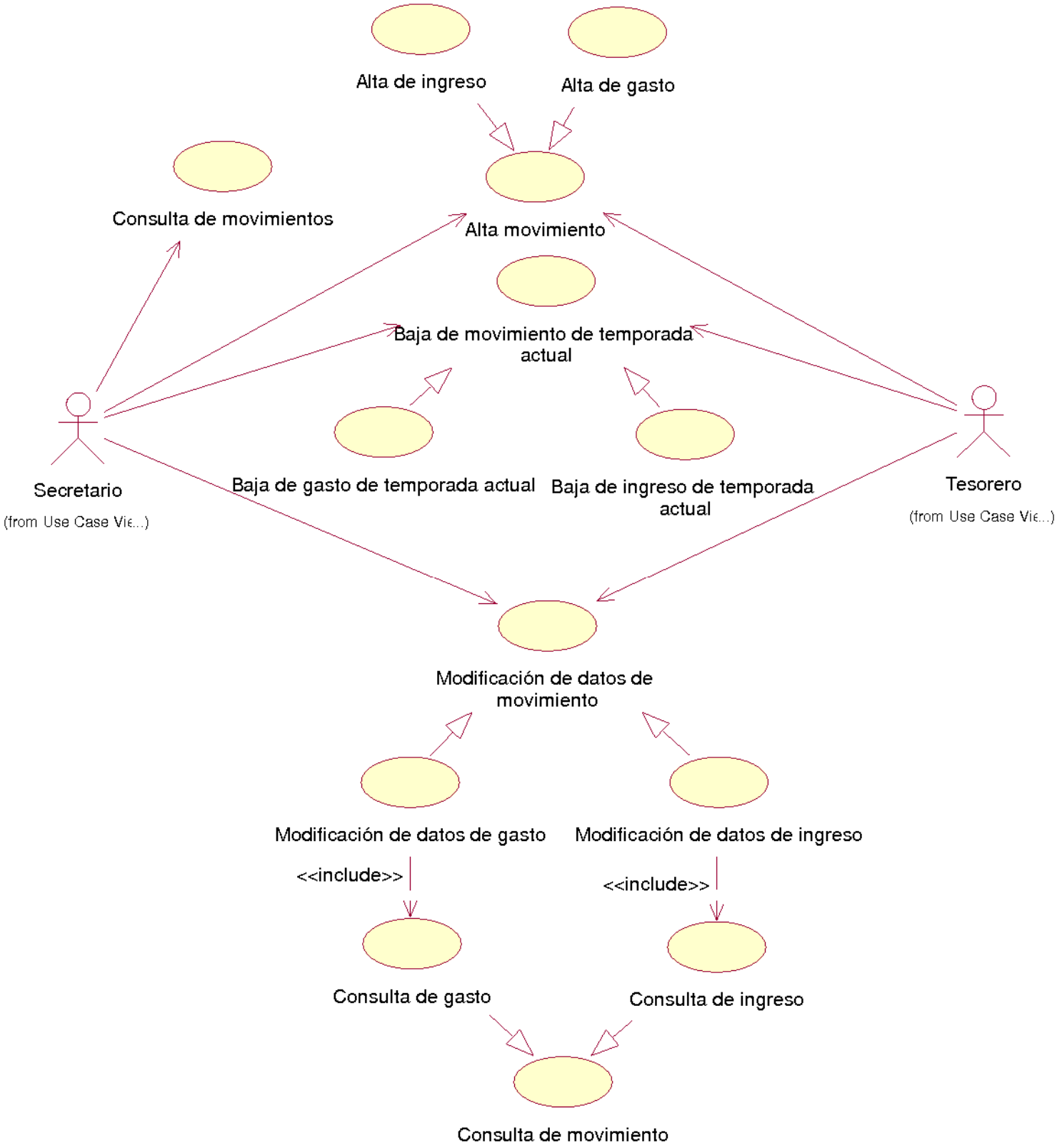

Figura 21. Diagrama gestión contable.

Diagrama de caso de uso del subsistema gestión de vedas

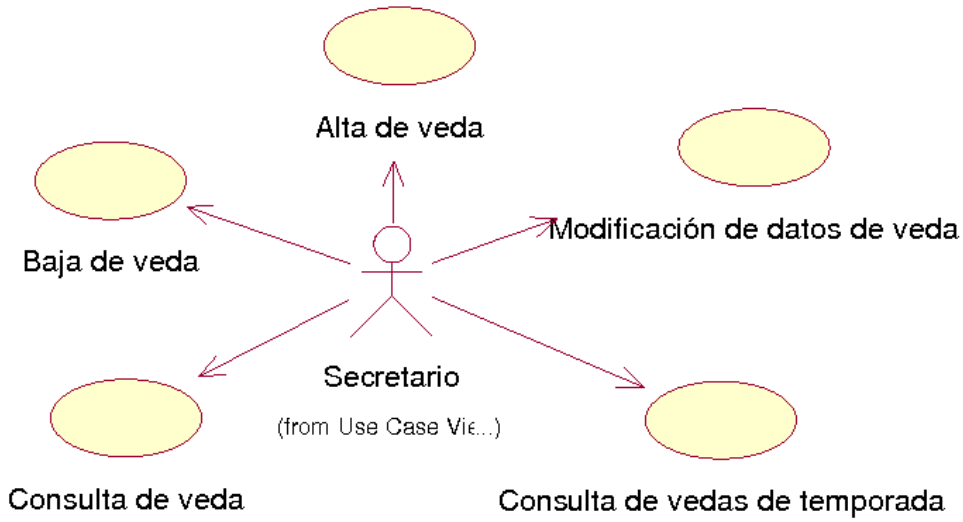

Figura 22. Diagrama gestión de vedas.

Diagrama de caso de uso del subsistema gestión de asambleas

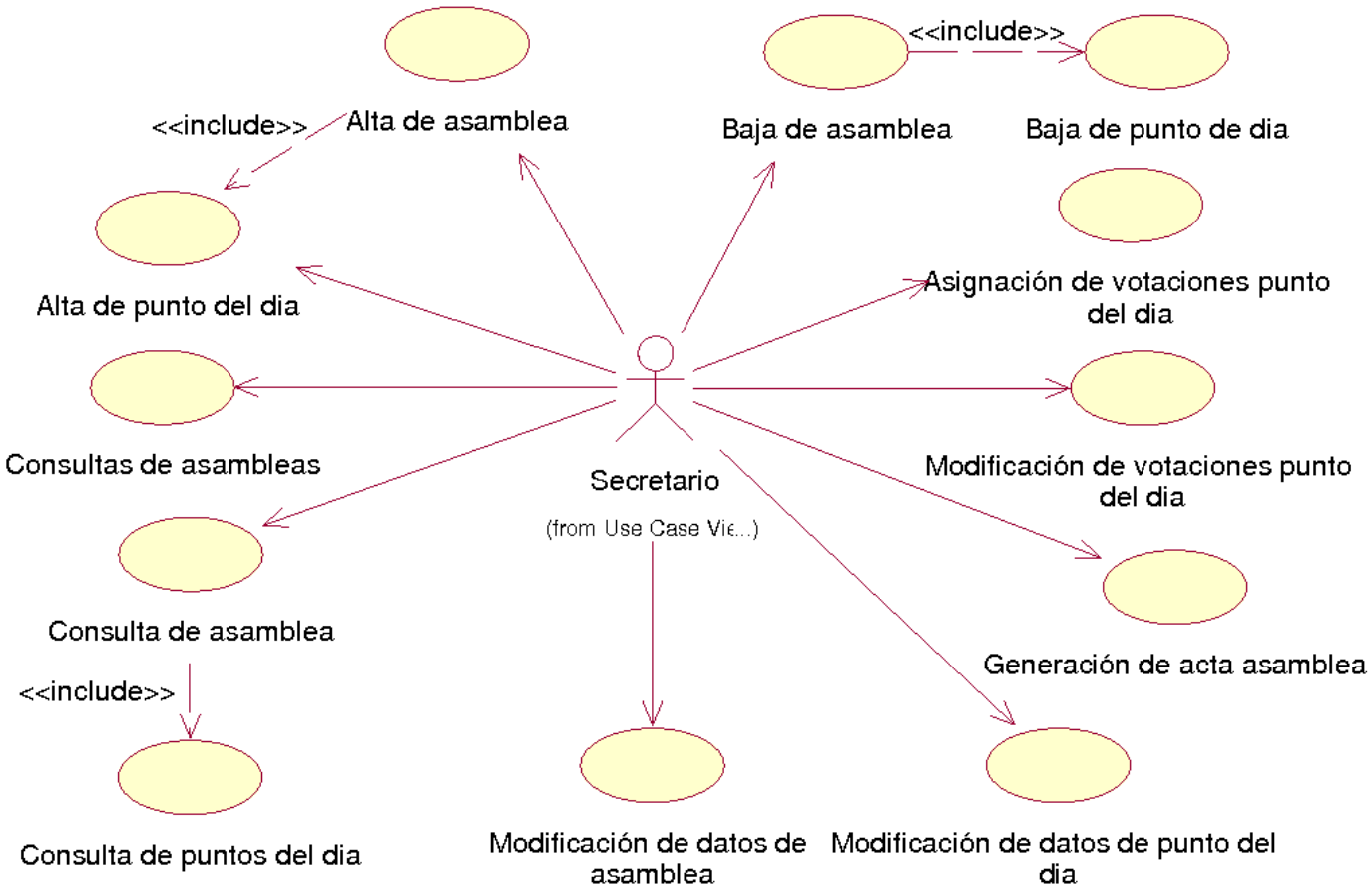

Figura 23. Diagrama gestión de asambleas.

Diagrama de caso de uso gestión de socios

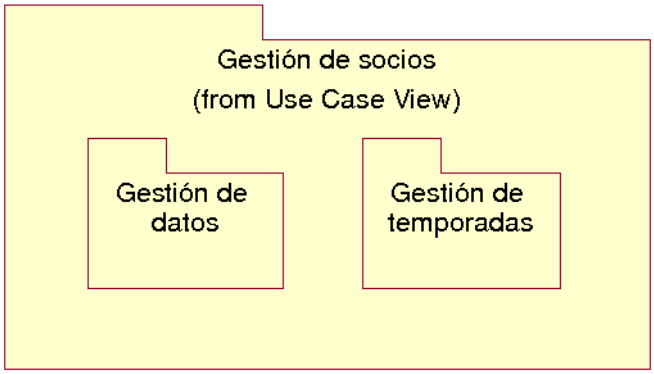

Figura 24. Diagrama gestión de socios.

Diagrama de caso de uso del subsistema gestión de datos de socios

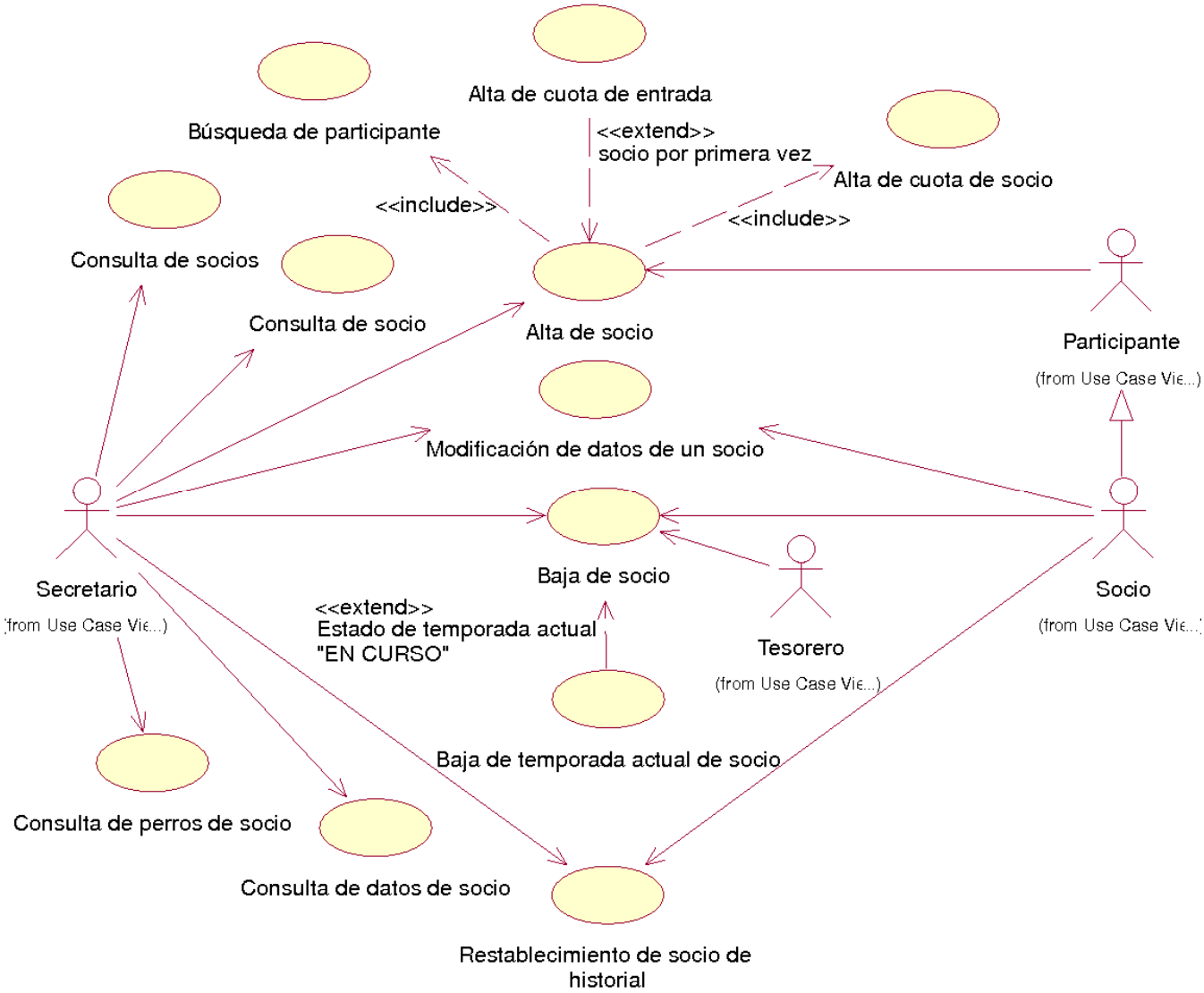

Figura 25. Diagrama gestión de datos de socios.

Caso de uso	Alta de socio.
Actores	Secretario, Cazador.
Propósito	Crear un nuevo socio.
Resumen	El secretario solicita al nuevo socio los datos que se quieren conocer del mismo y lo guarda en el sistema.
Precondiciones	
Postcondiciones	
Incluye	Alta de cuota de socio.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando un cazador desea hacerse socio de la sociedad.	
2. El secretario solicita dar de alta un socio.	
3. El secretario introduce el DNI que el cazador le indica.	4. Comprueba que no existe un socio o no socio con el DNI introducido.
5. Si no existe socio pero si no socio.	
	6. Introduce los datos que ya se conocen del nuevo socio.
7. El secretario introduce los datos de identificación que el cazador le indica.	8. Valida los datos introducidos.
	9. Registra el alta del nuevo socio.
10. Se realiza el caso de uso Alta de cuota de socio.	
	11. Muestra un mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 el DNI del cazador introducido es de un socio antiguo, con estado “HISTORIAL”.	
	1.1. Muestra mensaje informando que fue socio anteriormente.
2. Si en 4 el DNI del cazador introducido es de un socio actual, con estado “ACTUAL”.	
	2.1. Muestra mensaje informando que es socio actual.
3. Si en 6 los datos introducidos son incorrectos.	
	3.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Diagrama de caso de uso del subsistema de gestión de temporadas de socios

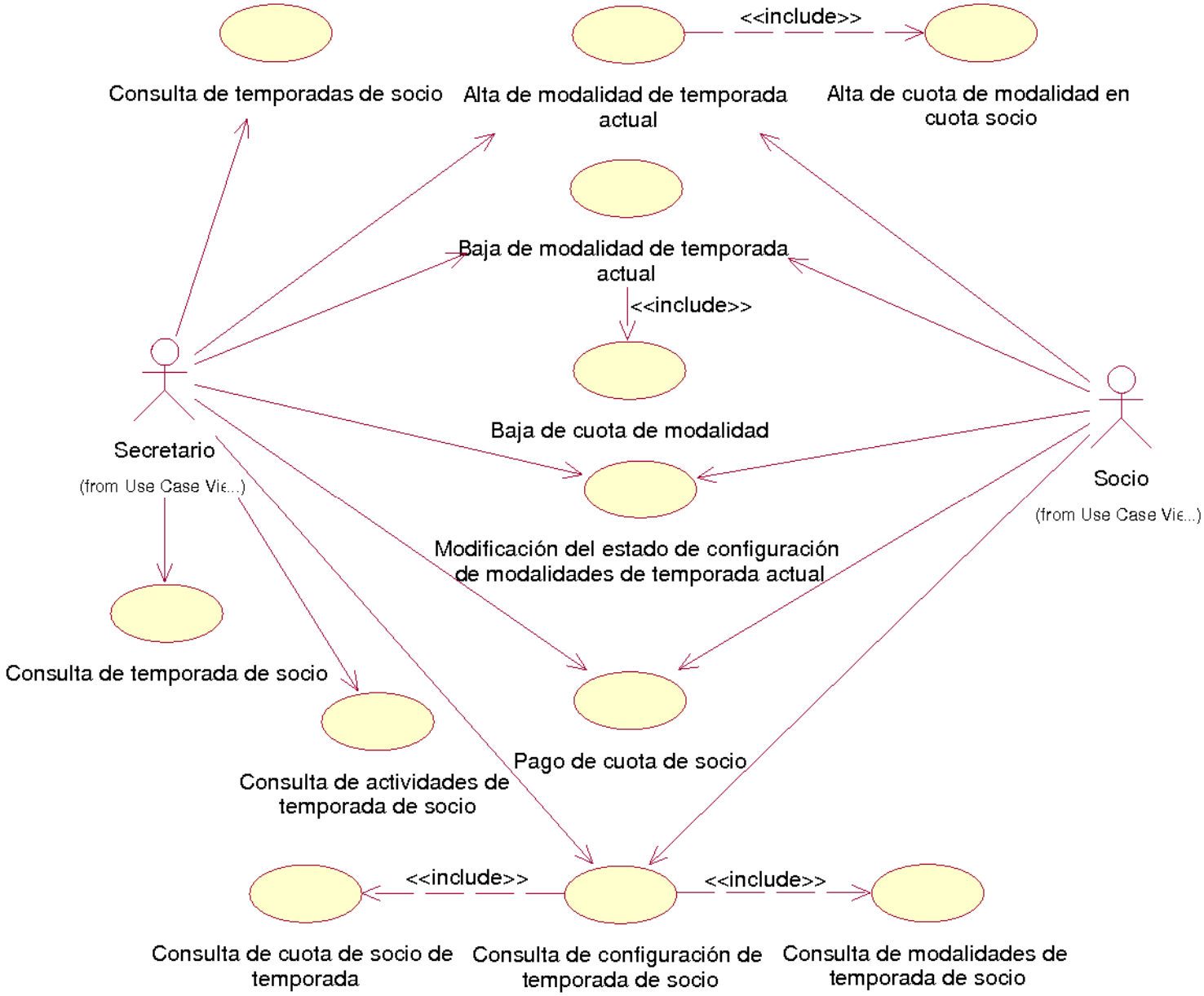

Figura 26. Diagrama gestión de temporadas de socios.

Diagrama de caso de uso subsistema gestión de juntas directivas

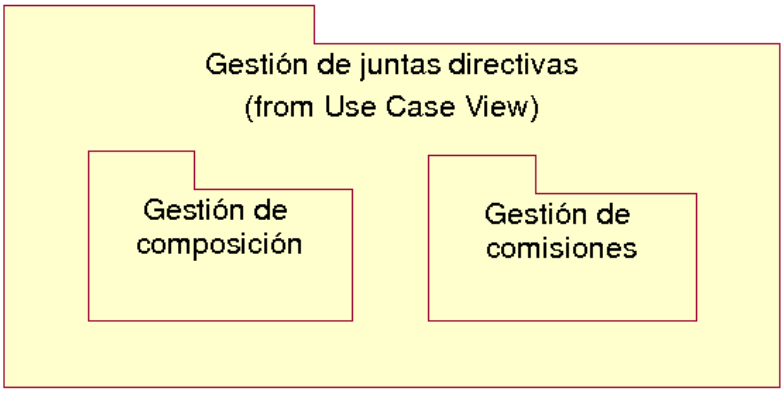

Figura 27. Diagrama gestión de juntas directivas.

Diagrama de caso de uso subsistema gestión de composición de juntas directivas

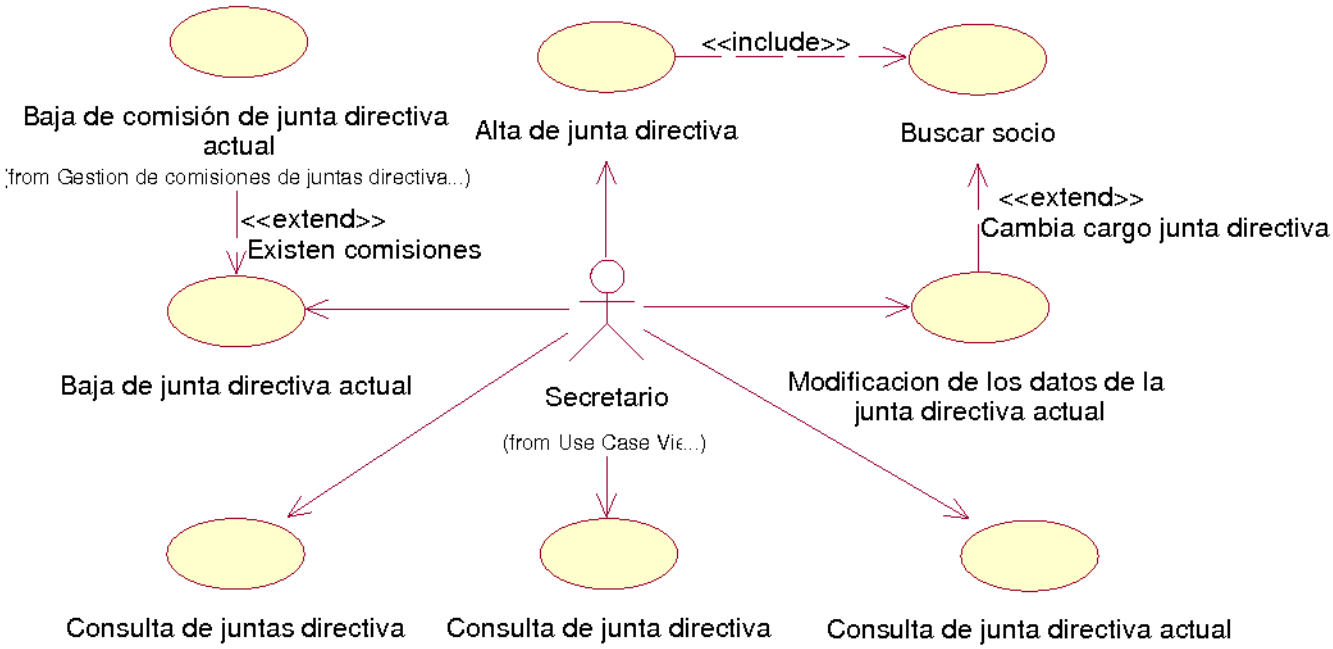

Figura 28. Diagrama gestión de composición de juntas directivas.

Diagrama de caso de uso subsistema gestión de comisiones de juntas directivas

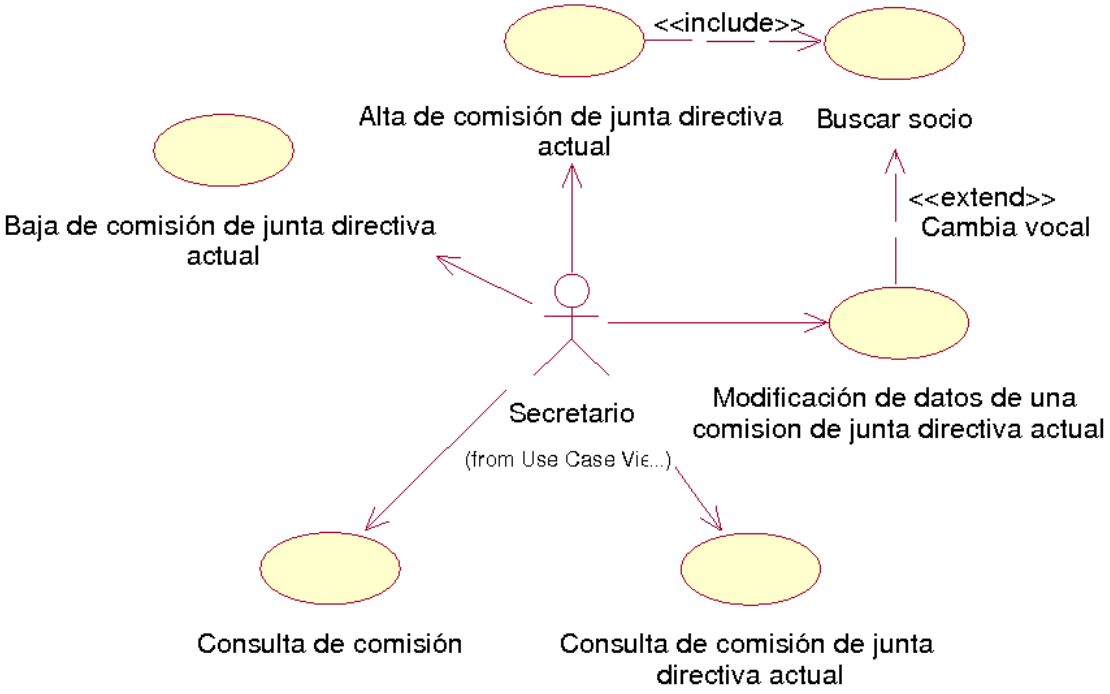

Figura 29. Diagrama gestión de comisiones juntas directivas.

Diagrama de caso de uso subsistema gestión de perros

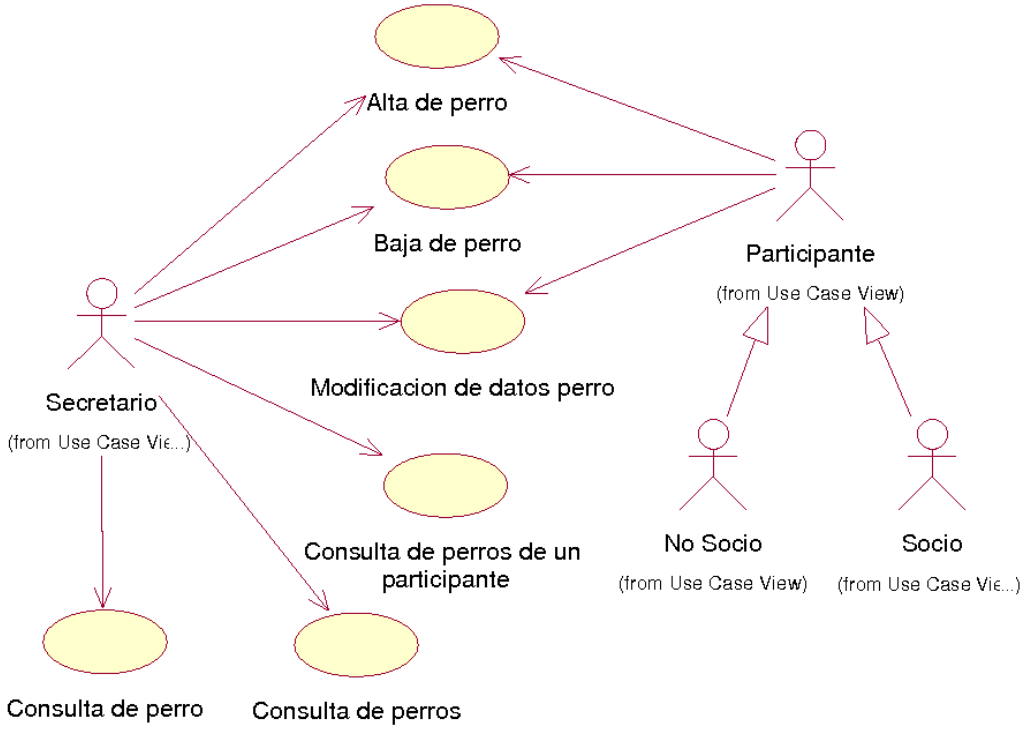

Figura 30. Diagrama gestión de perros.

Para describir cada uno de los casos de uso mostrados, en el anexo C aparecen sus plantillas, mediante las cuales se entenderá la forma de interactuar entre los actores y el sistema, permitiéndose que se entienda mucho mejor el sistema.

3.2. Diagrama de clases

Es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

A continuación se muestra el diagrama de clases del presente caso de estudio. En él no aparecen los atributos de las diferentes clases para que se pueda mostrar el diagrama completo.

Tras el diagrama de clases se muestra a modo de ejemplo una de las clases para poder ver en detalle cómo se definen los atributos.

Para finalizar se muestra un fragmento del diagrama de clases mediante el cual se podrá ver con mayor claridad la manera de definir las clases y las relaciones entre ellas.

Diagrama de clases sin especificación de atributos

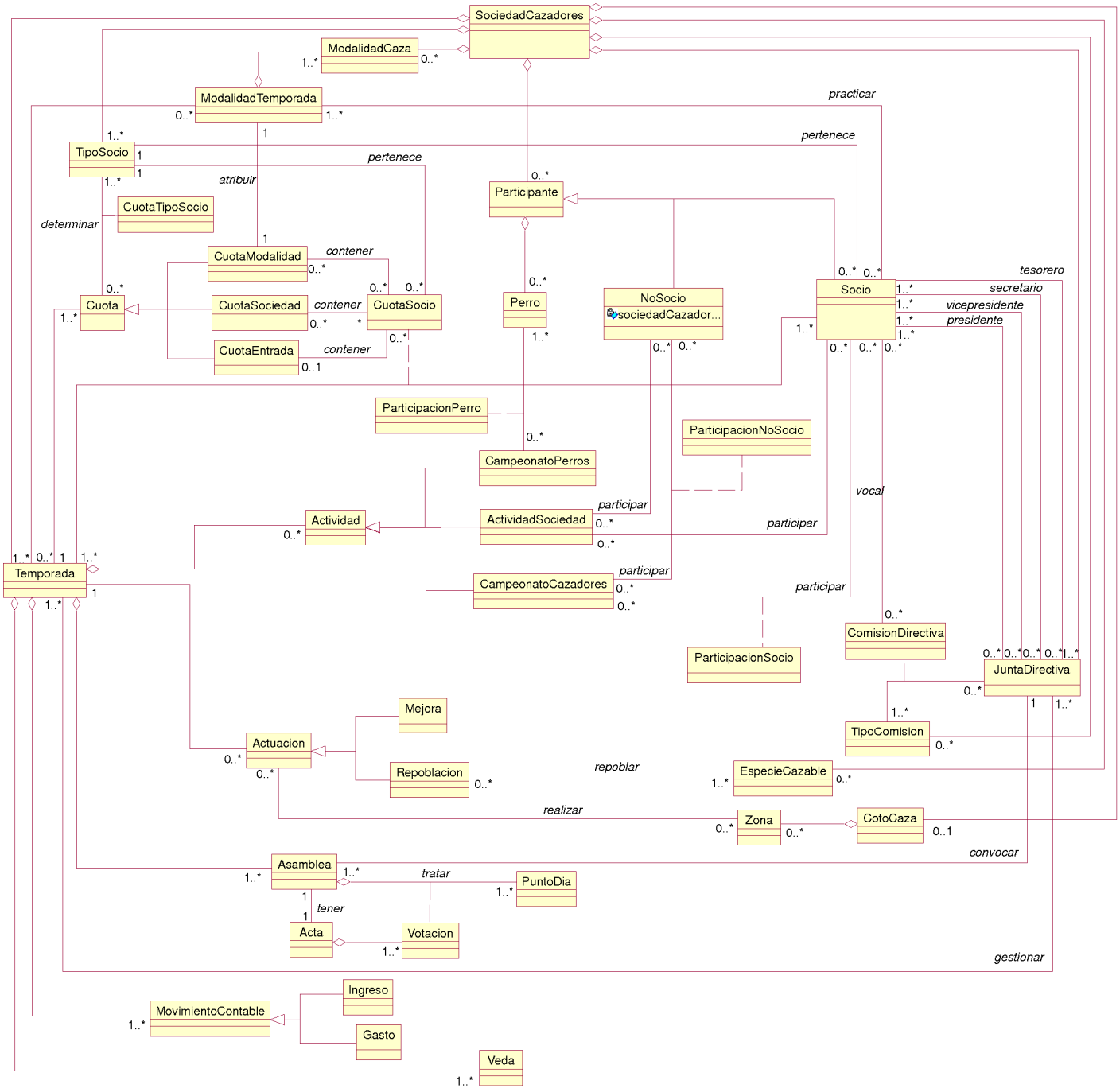

Figura 31. Diagrama de clases.

Clase Participante y Socio en detalle

Se visualizan los atributos que definen la clase Participante y Socio.

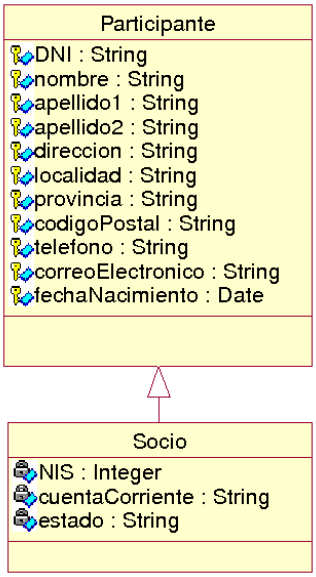

Figura 32. Clase Participante y Socio en detalle.

Fragmento del diagrama de clases

En particular se muestra la parte del sistema la que gestiona las cuotas de los socios.

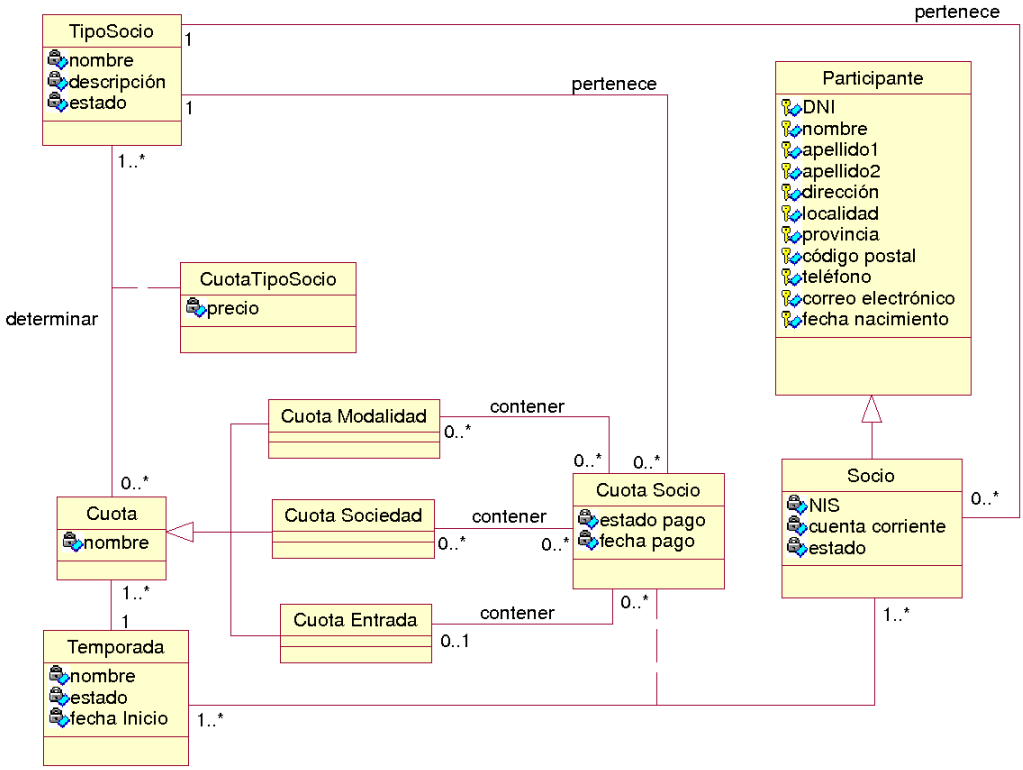

Figura 33. Fragmento de diagrama de clases.

3.3. Diagramas de secuencia

Un diagrama de secuencia describe un escenario, que es una secuencia de sucesos que se produce durante una ejecución concreta de un sistema, mostrando la interacción de un conjunto de objetos a través del tiempo. Mientras que el diagrama de casos de uso permite el modelado de una vista *business* del escenario, el diagrama de secuencia contiene detalles de implementación del escenario.

Muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.

El punto de partida para modelar un diagrama de secuencia es el caso de uso al que hace referencia, y en particular su plantilla de descripción que es donde se describe la comunicación entre los actores y el sistema. Mediante la secuencia de pasos que define se descubren los objetos que intervienen en el escenario así como los eventos dirigidos hacia el sistema y desde el sistema a los actores.

A continuación se muestran 4 escenarios del presente caso de estudio.

Escenario baja de modalidad de caza

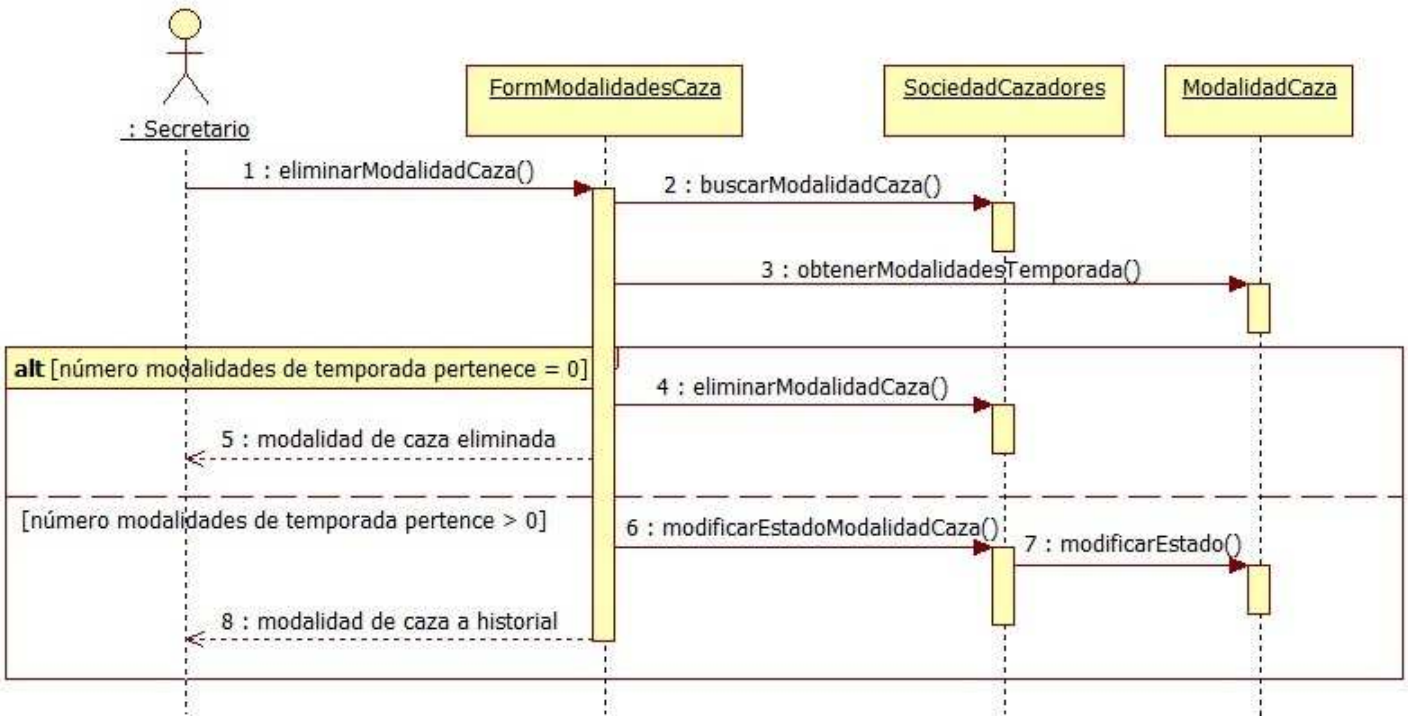

Figura 34. Diagrama de secuencia baja de modalidad de caza.

Escenario alta de socio

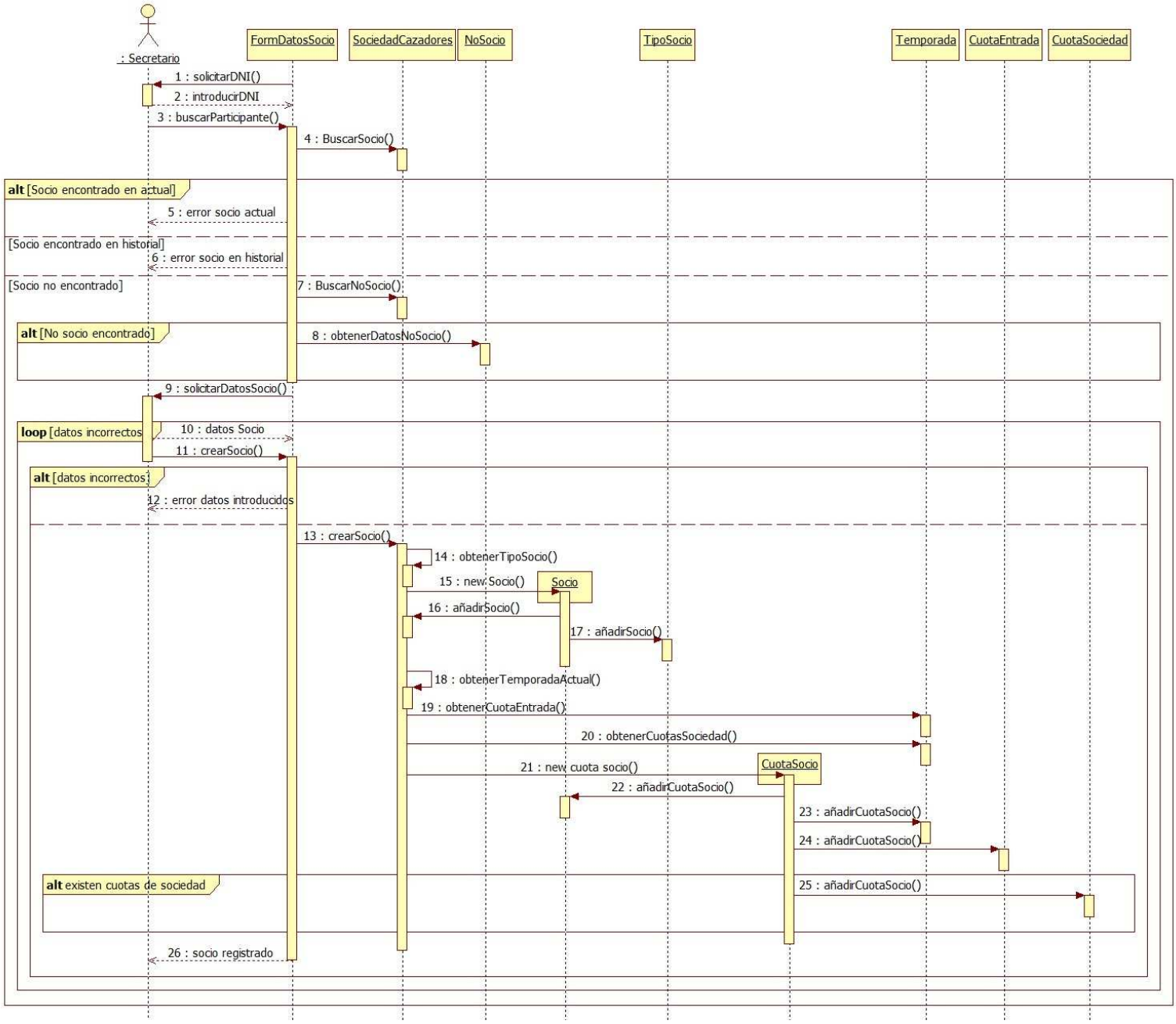

Figura 35. Diagrama de secuencia alta de socio.

Escenario modificación de modalidad de temporada

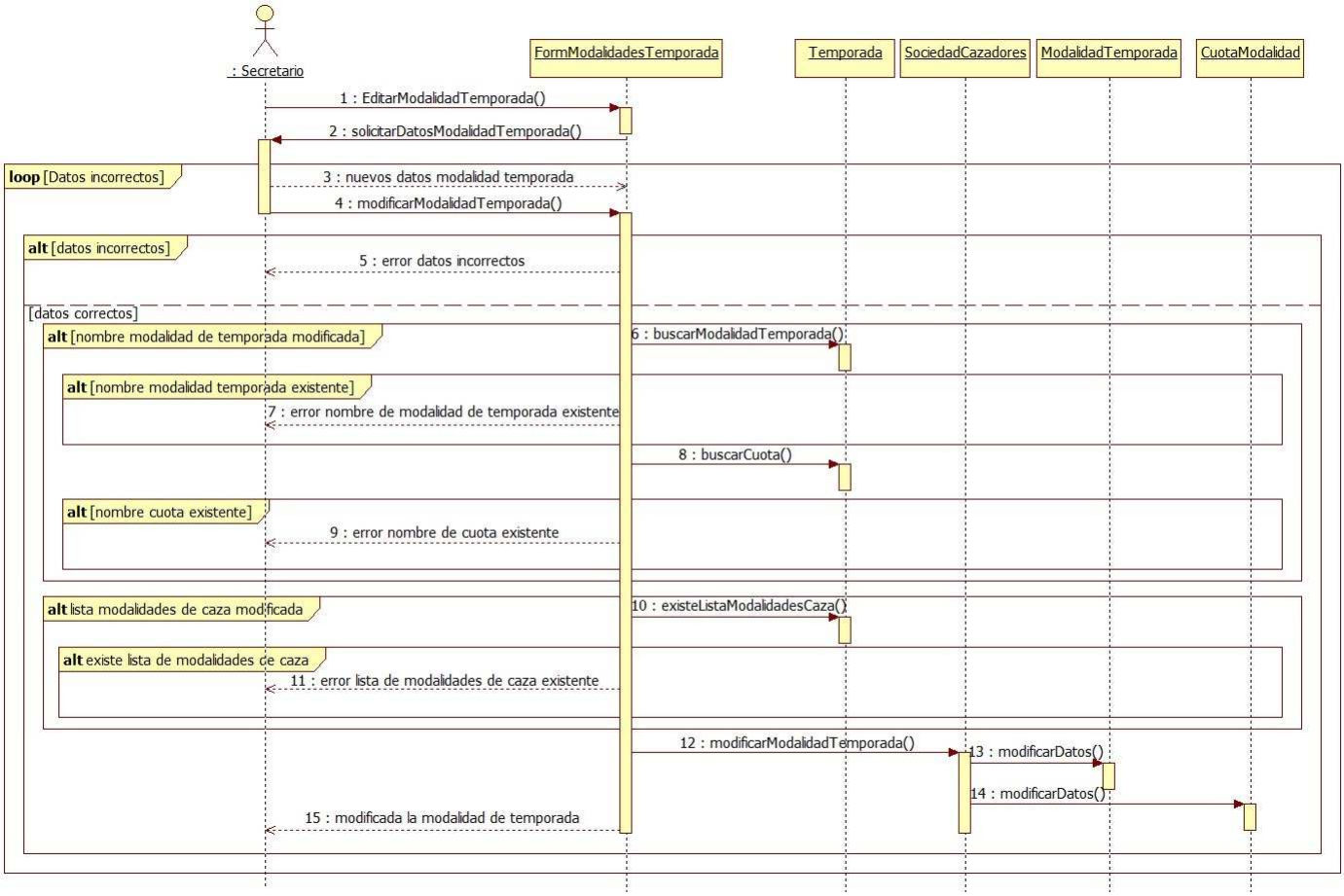

Figura 36. Diagrama de secuencia modificación de modalidad de temporada.

Escenario consulta clasificación de campeonato de cazadores

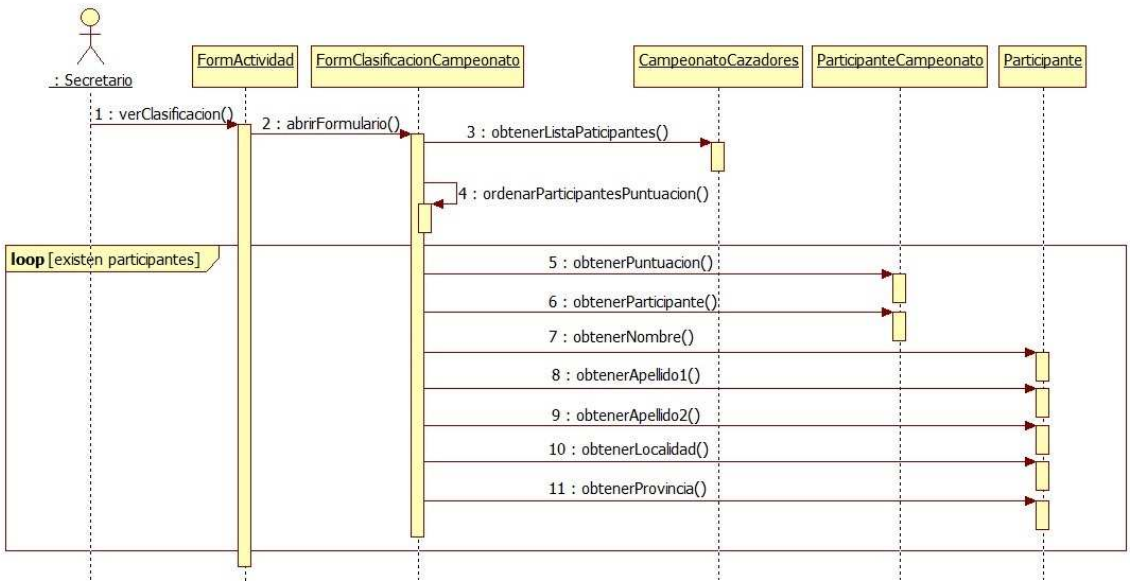

Figura 37. Diagrama de secuencia consulta de clasificación de campeonato de cazadores.

Capítulo 4:

Arquitectura del sistema

Este término tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad (Kruchten, November 1995).

La aplicación del presente caso de estudio se ha desarrollado bajo la **arquitectura cliente/servidor de tres capas**.

Que una **arquitectura** es **cliente/servidor** significa que se modela como un conjunto de servicios que son proporcionados por los servidores y un conjunto de clientes que usan dichos servicios, siendo clientes y servidores procesos lógicos y distintos.

Que es una **arquitectura n-capas**, en este caso tres en capas, quiere decir que el sistema es un conjunto ordenado de subsistemas, cada uno de los cuales está construido en términos de los que tiene por debajo, y proporciona la base de la implementación de aquellos que estén por encima de él.

Es recomendable que los objetos de cada capa sean independientes, aunque es habitual que existan dependencias entre objetos de distintas capas ya que existe una relación cliente/servidor entre las capas inferiores que proporcionan servicios y las capas superiores que son consumidores de estos servicios.

Las arquitecturas basadas en capas, pueden ser abiertas o cerradas según la dependencia que exista entre las capas:

- Abierta**:** Una capa puede utilizar características de cualquier capa a cualquier nivel.

- Cerrada: Una capa sólo utiliza características de la capa inmediatamente inferior.

Normalmente es recomendable trabajar con arquitecturas cerradas pues así se reducen las dependencias entre niveles y esto permite que los cambios en la implementación de una capa no afecten al resto de capas.

La **arquitectura de 3 capas** es la más básica de las arquitecturas de n-capas y en ella podemos distinguir los niveles de:

- **Presentación**: Proporciona la interfaz visual que los clientes utilizarán para ver la información y los datos. Los componentes son responsables de solicitar y recibir servicios de otros componentes del mismo nivel o del nivel de negocio.

- **Negocio o Lógica**: Como los servicios de usuario no pueden contactar directamente con el nivel de servicios de datos, es responsabilidad de los servicios de negocio hacer de puente entre estos. Los objetos de negocio proporcionan servicios que completan las tareas de negocio tales como verificar los datos enviados por el cliente. Antes de llevar a cabo una transacción en la B.D.

- **Persistencia o Datos**: Se encarga de las típicas tareas que realizamos con los datos: Inserción, modificación, consulta y borrado. La clave de este nivel es que los papeles de negocio no son implementados aquí. Aunque un componente de servicio de datos es responsable de la gestión de las peticiones realizadas por un objeto de negocio.

El esquema genérico de la arquitectura cliente/servidor de 3 capas es el siguiente:

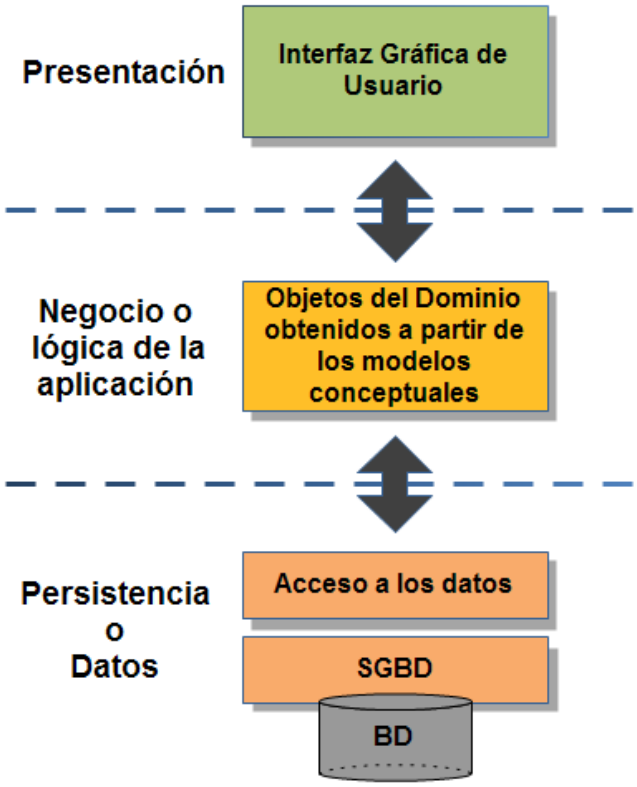

Figura 38. Arquitectura genérica de tres capas (extraída de (ISG, Ingeniería del software de gestión, 2008/09)).

La adaptación de dicha arquitectura a nuestro caso de estudio tendrá la estructura y los componentes que se muestran en la figura 39:

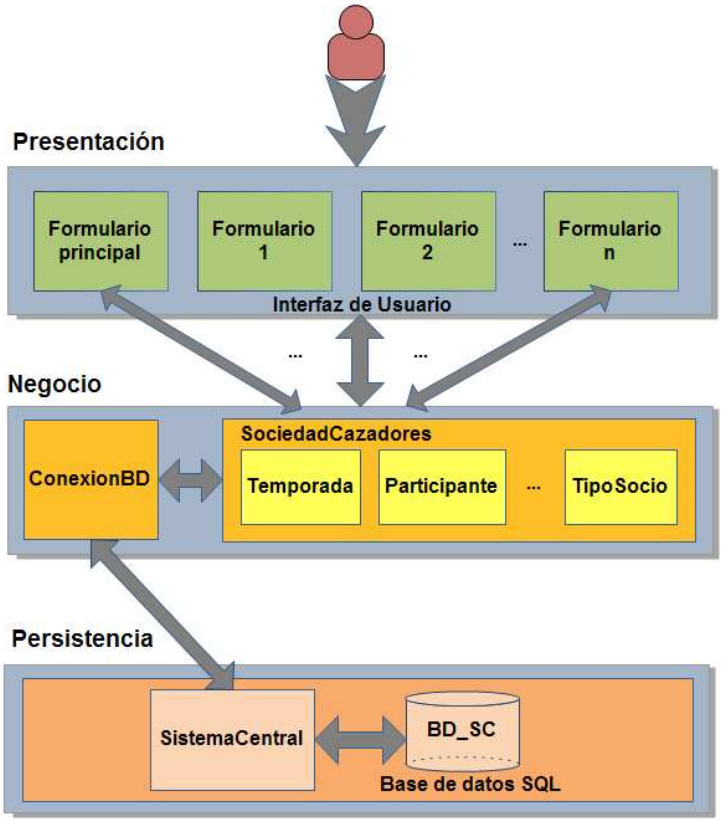

Figura 39. Arquitectura de tres capas del caso de estudio (adaptada de (ISG, Ingeniería de software de gestión, 2008/09).

- En la capa de **presentación** entre todas las clases/formularios que existen la más importante es el formulario *Formulario principal*, una instancia de este, es el punto de entrada a la aplicación, y por ello es responsabilidad de dicho formulario inicializar el sistema (escenario típicamente conocido como carga del sistema) creando los objetos necesarios para la correcta ejecución del programa.

- En la capa de **negocio** hay que destacar la clase *SociedadCazadores* y la clase *ConexionBD*.

  La clase *SociedadCazadores* es la clase a través de la cual se accede al resto de las clases de la capa negocio. Mediante ella se realiza la **comunicación entre** la capa de **presentación** y la capa de **negocio**, en cada una de las instancias que se creen de los diferentes formularios se asignara el atributo que corresponde al objeto de la clase *SociedadCazadores*.

  La otra clase importante es *ConexionBD* ya que es la clase a través de la que se realiza la **comunicación entre** la capa de **negocio y** la capa de **persistencia**, comunicando el objeto creado de la clase *SociedadCazadores* y el objeto creado de la clase *SistemaCentral*.

- En la capa de **persistencia** la única clase, que no por ello menos importante, es la clase *SistemaCentral* que es la clase mediante la cual se accede a la base de datos.

Capítulo 5:

Diseño

En este capítulo se muestran los diseños asociados a las tres capas de la arquitectura del sistema:

- Diseño de IGU (capa de presentación).
- Diseño de clases (capa de negocio o lógica de la aplicación).
- Diseño de base de datos (capa de persistencia).

5.1. Diseño de IGU

El diseño de la interfaz gráfica (IGU) de usuario es un factor muy importante en la realización de una aplicación debido a que es el medio de comunicación mediante el cual el usuario interactúa con la aplicación.

Por ello se ha intentado que nuestra aplicación haya logrado un diseño de interfaces eficaces que cumpla con los principios generales de diseño de las interfaces de usuario (Sommerville, 2002) que se definen a continuación:

- **Familiaridad del usuario**. La interfaz debe utilizar términos y conceptos que se toman de la experiencia de las personas que más utilizan el sistema.

- **Consistencia.** La interfaz debe ser consistente en el sentido de que operaciones similares se realizan o activan de la misma forma.

- **Mínima Sorpresa.** El comportamiento del sistema no debe provocar sorpresa a los usuarios.

- **Recuperabilidad.** La interfaz debe incluir mecanismos que permitan a los usuarios recuperarse de los errores.

- **Guía al usuario** La interfaz debe proveer retroalimentación significativa y características de ayuda sensible al contexto.

- **Diversidad de Usuarios.** La interfaz debe proveer características de interacción apropiada para los diferentes tipos de usuarios del sistema.

Las interfaces, formularios, en nuestra aplicación se dividen en tres tipos:

- **Formulario padre**. Es un formulario contenedor que se divide en dos paneles verticales, en el de la izquierda es donde se mostraran los formularios hijos.

- **Formulario hijo tipo A**. Es un formulario dividido en dos paneles horizontales, en el superior se gestionan y se listan las instancias de un concepto, en el inferior se gestiona toda la información de una instancia concreta.

- **Formulario hijo tipo B**. Es un formulario en el que se gestiona toda la información de una instancia concreta, cuando el volumen de información no cabe en el panel inferior de un formulario hijo tipo B.

La organización de dichos formularios se muestra en la figura 40. Para una mejor compresión de dicha figura se aclara que **gestión de concepto** es la forma de abstraer a cualquiera de las gestiones que se identifican en el Diagrama inicial de los casos de uso que aparece al comienzo del apartado 3.1.

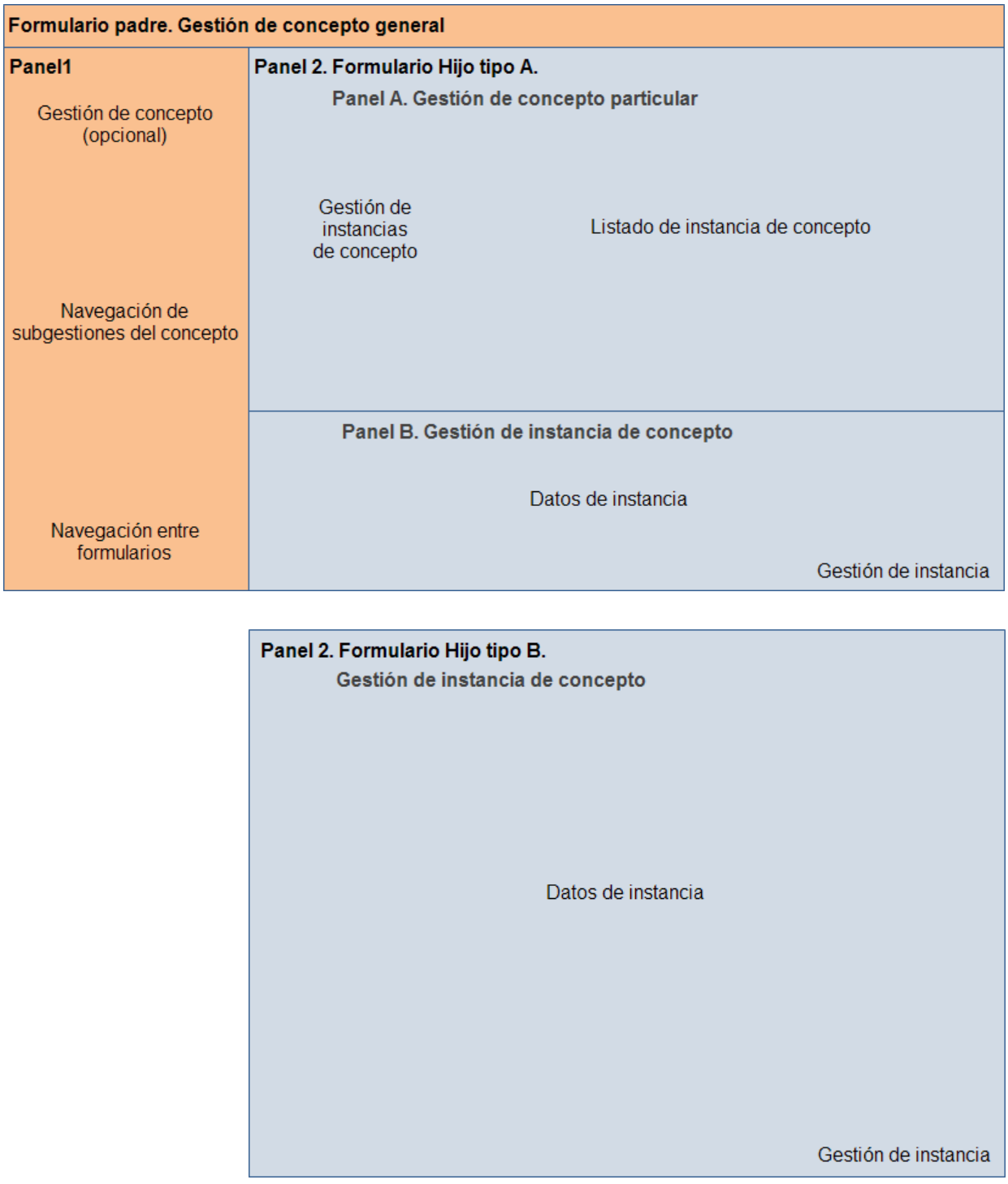

Figura 40. Esquema de formulario.

Para ver diseños concretos de los formularios de la aplicación y comprender como es la interacción con el usuario a se muestran los formularios asociados a los diagrama de secuencia que se mostraron en el apartado 3.3. Diagramas de secuencia.

Baja de modalidad de caza

En el formulario que se muestra en la figura 41 se realizan todas las operaciones asociadas a la gestión de las modalidades de caza.

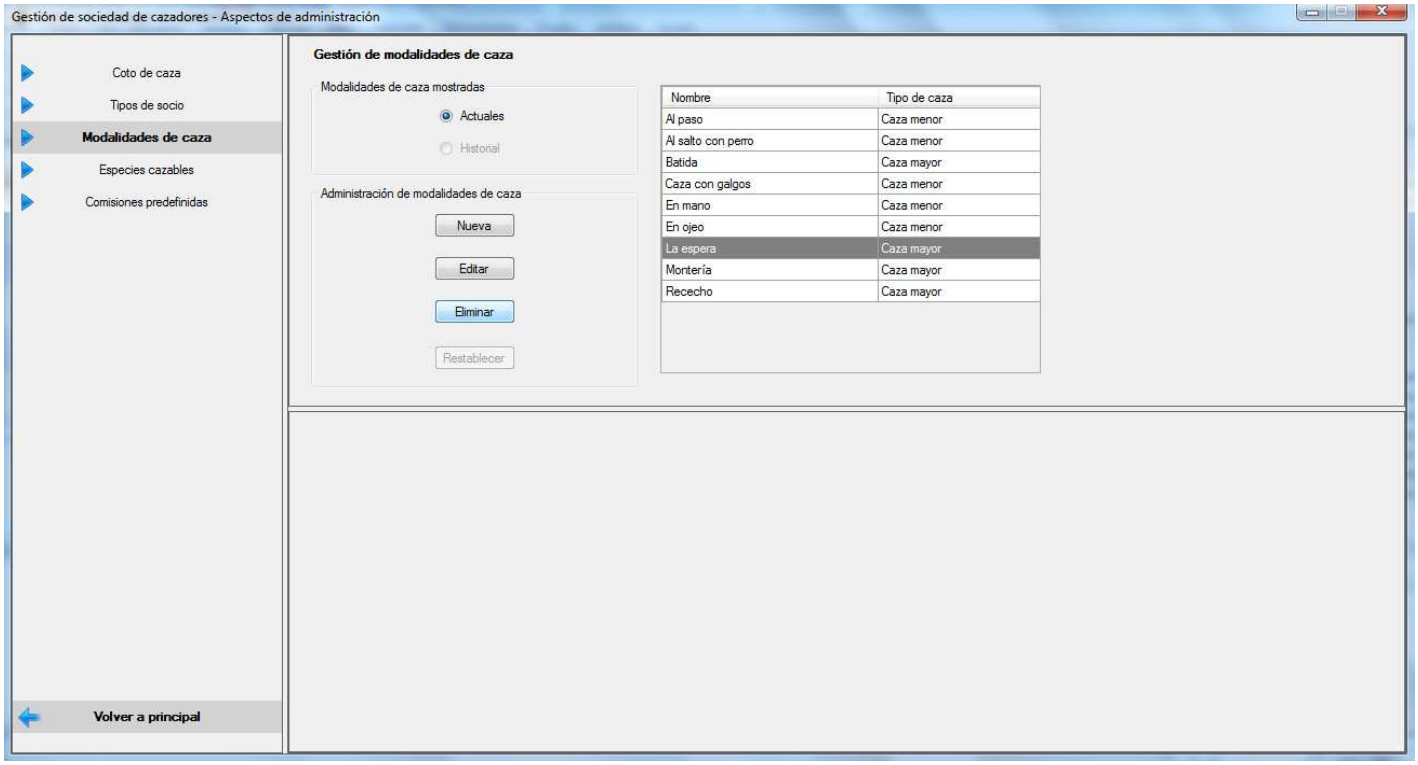

Figura 41. Formulario gestión de modalidades de caza.

A continuación se explica de este formulario como el usuario realiza la operación de dar de baja una modalidad de caza, los pasos a seguir son los siguientes:

1. El usuario hace clic en el panel de la izquierda sobre Modalidades de caza.
2. El sistema muestra el formulario de gestión de modalidades de caza en el panel de la derecha.
3. El usuario selecciona la modalidad de caza que se quiere eliminar.
4. El usuario hace clic en el botón eliminar.
5. El sistema realiza e informa mediante un mensaje de la operación que ha realizado:
   - Dar de baja la modalidad de caza.
   - Establecer la modalidad de caza en el historial de modalidades de caza.
   - Denegar la operación de dar de baja debido a que la modalidad de caza forma parte de una modalidad de temporada actual.

Alta de socio

En el formulario que se muestra en la figura 42 se realizan todas las operaciones asociadas a la gestión de socios.

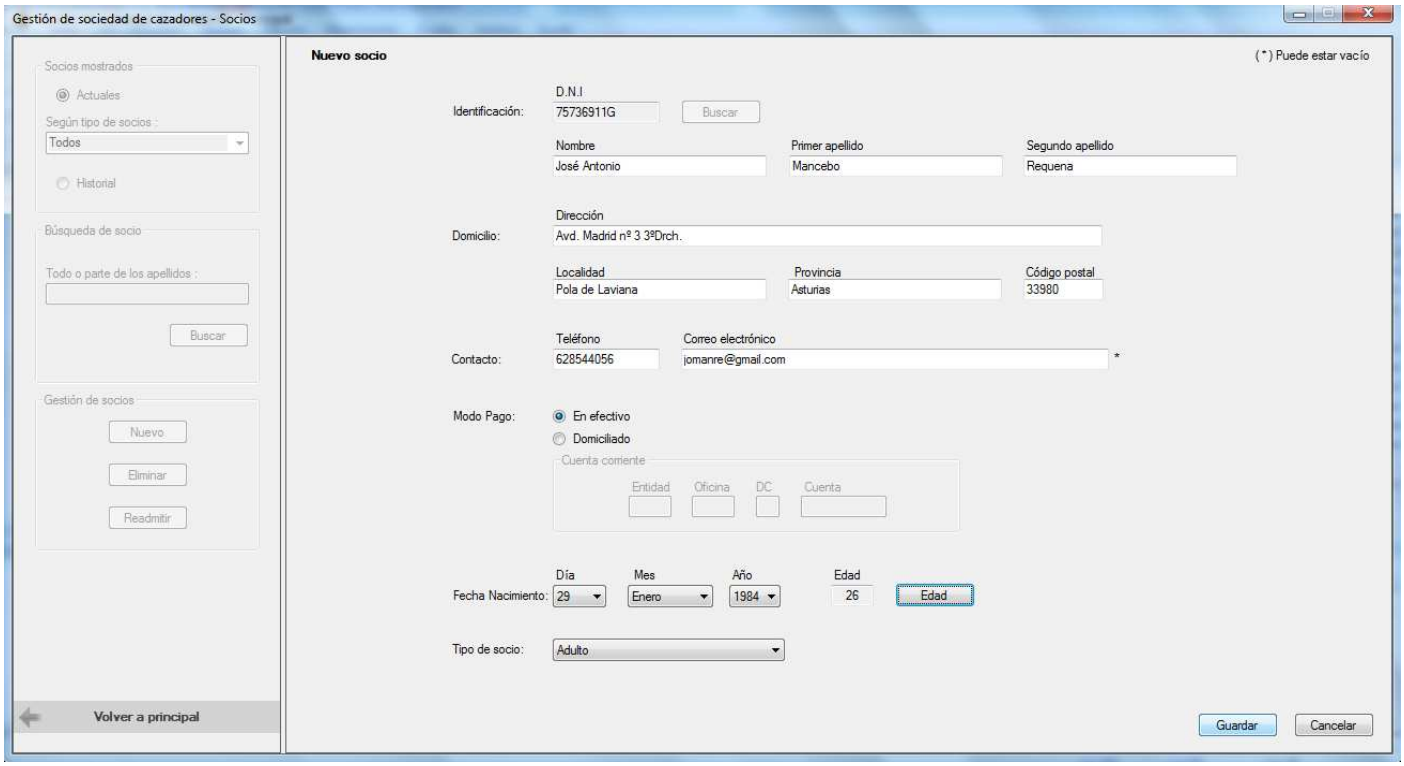

Figura 42. Formulario de gestión de socios.

A continuación se explica de este formulario como el usuario realiza la operación de dar de alta un socio, los pasos a seguir son los siguientes:

1. El usuario hace clic en el panel de la izquierda (deshabilitado en la figura) sobre el botón nuevo.
2. El sistema muestra el formulario de alta de socio en el panel de la derecha.
3. El usuario introduce el DNI del nuevo socio y hace clic en el botón buscar (deshabilitado en la figura).
4. El sistema comprueba si existe el socio.
   - Si la persona no existe, el sistema establece los campos en modo escritura excepto el campo DNI que lo establece en modo lectura.
   - Si la persona no existe como socio pero si como no socio, el sistema recupera los datos del no socio y establece los campos con permiso de escritura excepto el campo DNI que lo establece modo lectura.
   - Si la persona existe como socio actual o como socio de historial muestra un mensaje informando que no se puede realizar la operación.
5. El usuario introduce los datos del nuevo socio.

6. El usuario hace clic en guardar.
7. El sistema comprueba que los datos introducidos son correctos.
   - Si son correctos, el sistema realiza e informa mediante un mensaje que la operación de alta de socio se ha realizado:
   - Si son incorrectos, el sistema muestra mensaje de error.

Modificación de modalidad de temporada

En el formulario que se muestra en la figura 43 se realizan todas las operaciones asociadas a la gestión de modalidad de temporada.

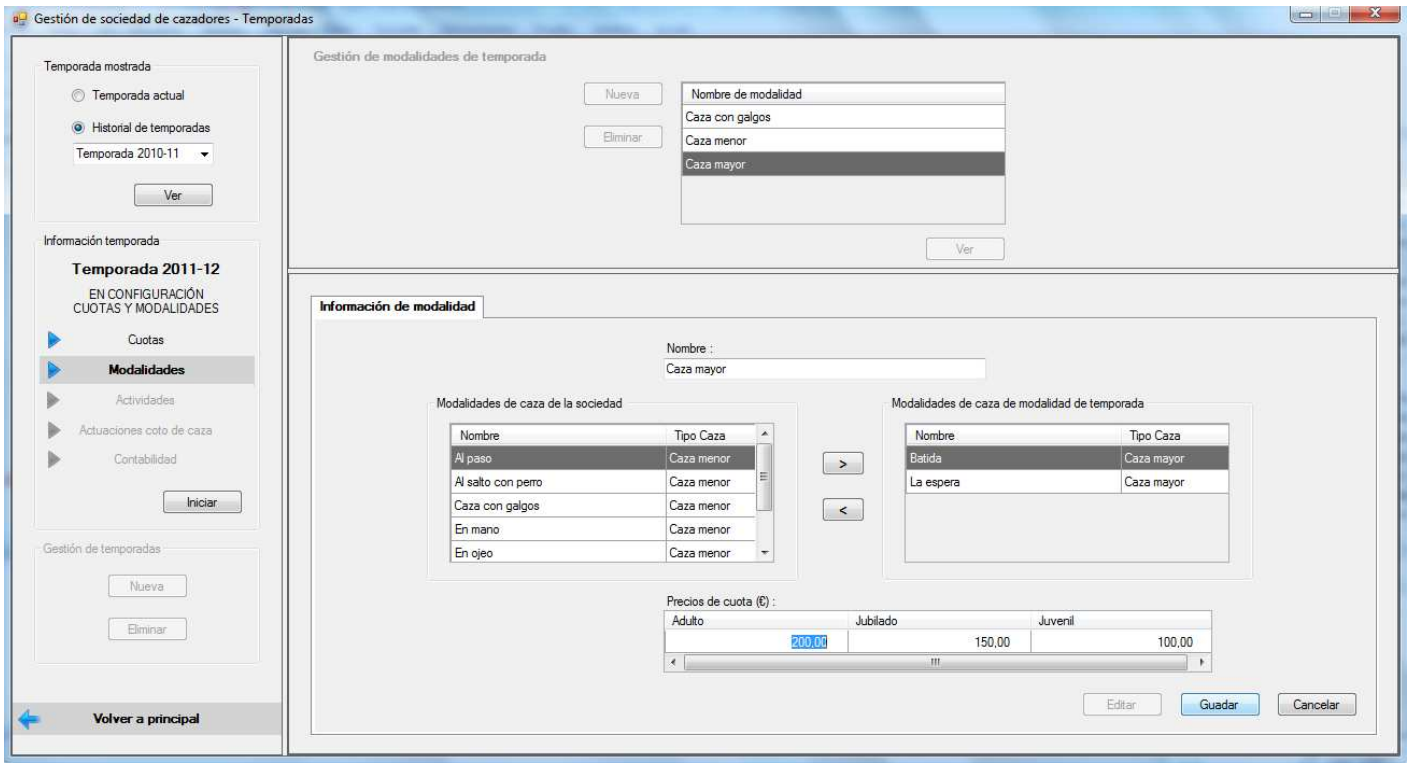

Figura 43. Formulario gestión de modalidades de temporada.

A continuación se explica de este formulario como el usuario realiza la operación de modificar una modalidad de temporada, los pasos a seguir son los siguientes:

1. El usuario hace clic en el panel de la izquierda sobre el botón modalidades.
2. El sistema muestra el formulario de gestión de modalidades de temporada en el panel de la derecha.
3. El usuario selecciona la modalidad de temporada que desea modificar y hace doble clic o hace clic en el botón Ver.
4. El sistema muestra en detalle la modalidad de temporada en el panel de abajo en modo lectura.

5. El usuario hace clic en el botón editar.
6. El sistema muestra la modalidad de temporada en modo escritura.
7. El usuario modifica los datos que desea, el nombre de la modalidad, las modalidades de caza que la componen o las cuotas que pagan diferentes tipos de socio.
8. El usuario hace clic en el botón guardar.
9. El sistema comprueba que los datos son correctos si no muestra mensaje de error.
10. El sistema, si se ha modificado el nombre, comprueba si existe otra modalidad de la temporada con el mismo nombre u otra cuota con el mismo nombre. Si ocurriera una de las dos cosas se muestra mensaje de error.
11. El sistema, si se ha modificado la lista modalidades de caza, comprueba si existe otra modalidad de la temporada con la misma lista de modalidades de caza. Si existe muestra mensaje de error.
12. Si todos los cambios son correctos, el sistema realiza e informa mediante un mensaje de la operación de modificación de modalidad de temporada.

Consulta clasificación de campeonato de cazadores

En el formulario que se muestra en la figura 44 se realizan todas las operaciones asociadas a la gestión de modalidad de temporada.

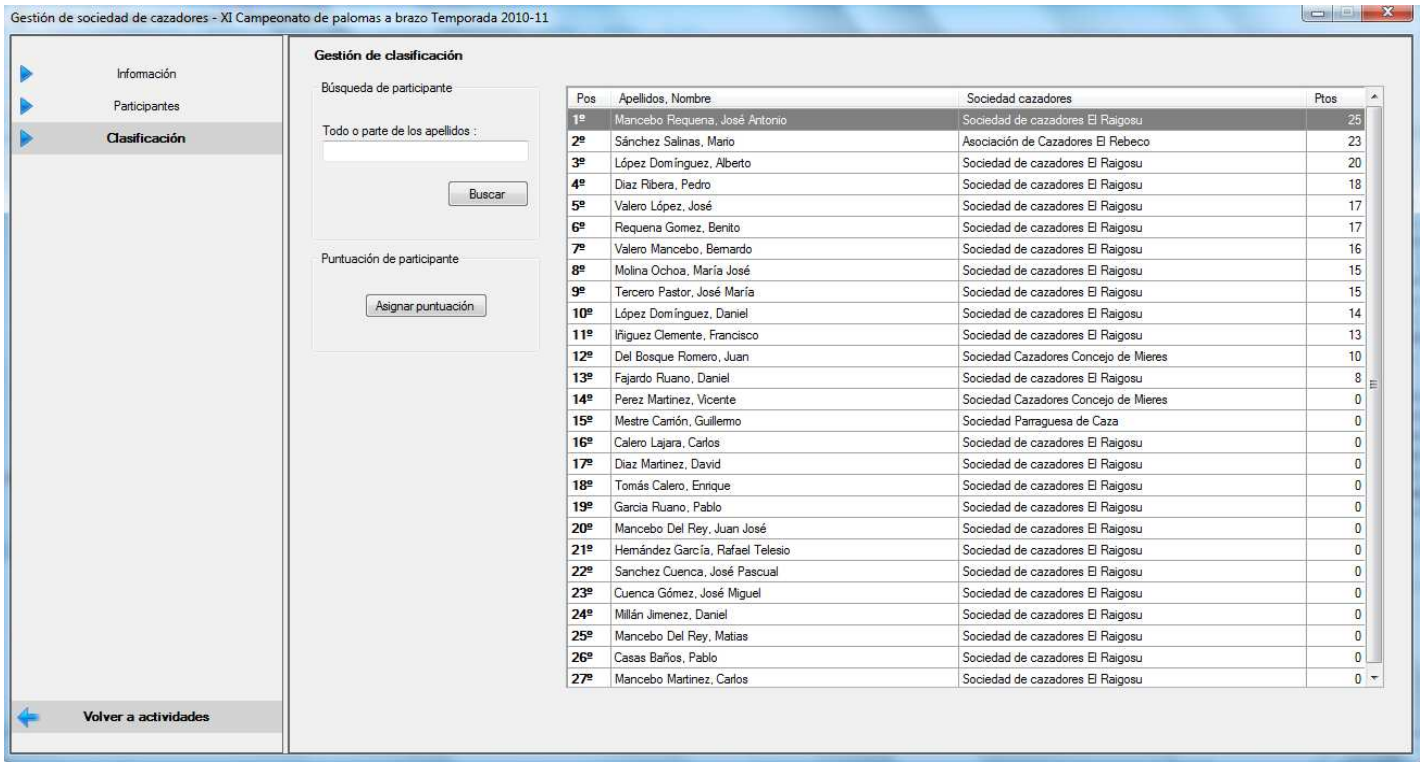

Figura 44. Formulario de gestión de campeonato de cazadores.

A continuación se explica de este formulario como el usuario realiza la operación de consulta de clasificación de campeonato de caza, los pasos a seguir son los siguientes:

1. El usuario hace clic en el panel de la izquierda sobre el botón clasificación.
2. El sistema muestra el formulario de gestión de la clasificación del campeonato de caza en el panel de la derecha.
3. El sistema carga la clasificación tras haber ordenado la lista de los participantes por los puntos que tiene.

5.2. Diseño de clases

Las clases que conforman la capa de negocio del caso de estudio se han diseñado bajo el lenguaje C# (lenguaje que se explica en el punto 6.1.Tecnología) siguiendo la siguiente estructura:

- Declaración de atributos.
- Constructores de la clase.
- Consultores de atributos.
- Modificadores de atributos.
- Métodos de tratamiento de colecciones.

Para mostrar dicha estructura a continuación se muestran las clases *Participante* y *Socio* las cuales fueron detalladas en la figura 32 del apartado 3.2. Diagrama de clases.

Estas clases también permitirán ver como se realiza la **herencia** que es el mecanismo por el cual elementos más específicos incorporan la estructura y el comportamiento definido en elementos más generales. En este caso en particular entre la clase padre, *Participante*, y las clase hija, *Socio*.

```
public class Participante
{
    //Declaración de atributos
    …
    //Constructores
    public Participante(string DNINuevo, string nombreNuevo, string apellido1Nuevo,
                        string apellido2Nuevo,string direccionNueva,string localidadNueva,
                       string provinciaNueva,string codigoPostalNuevo,string telefonoNuevo,
                        string correoElectronicoNuevo,DateTime fechaNacimientoNueva)
    {
        DNI = DNINuevo;
        nombre = nombreNuevo;
        apellido1 = apellido1Nuevo;
        apellido2 = apellido2Nuevo;
        direccion = direccionNueva;
        localidad = localidadNueva;
        provincia = provinciaNueva;
        codigoPostal = codigoPostalNuevo;
        telefono = telefonoNuevo;
        correoElectronico = correoElectronicoNuevo;
        fechaNacimiento = fechaNacimientoNueva;
    }

    //Consultores de atributos
    public string getDNI()
    {
        return DNI;
    }

    public string getNombre()
    {
        return nombre;
    }

    //Igual para el resto de atributos
    …

    //Modificadores de atributos
    public void setDNI(string DNINuevo)
    {
        DNI = DNINuevo;
    }

    public void setNombre(string nombreNuevo)
    {
        nombre = nombreNuevo;
    }

    //Igual para el resto de atributos
    …


    //Tratamiento de colecciones

    //En esta clase no existen colecciones

    //Fin de clase

}
```

```
public class Socio:Participante //Herencia
{
    //Declaración de atributos
    private int NIS;
    private string cuentaCorriente;
    private string estado;
    private TipoSocio tipoSocioPertenece;
    static private int siguienteNIS = 1;
    //Atributos colección
    private List<CuotaSocio> listaCuotasSocio;
    private List<ModalidadTemporada> listaModalidadesTemporadas;
    private SortedList<long, JuntaDirectiva> listaTesoreroJuntasDirectivas;
    private SortedList<long, JuntaDirectiva> listaSecretarioJuntasDirectivas;
    private SortedList<long, JuntaDirectiva> listaVicepresidenteJuntasDirectivas;
    private SortedList<long, JuntaDirectiva> listaPresidenteJuntasDirectivas;
    private List<Comision> listaVocalComisiones;
    private List<ActividadSociedad> listaActividadesSociedad;
    private List<CampeonatoSocio> listaCampeonatosSocio;

    //Constructores
    public Socio(string cuentaCorrienteNueva,TipoSocio tipoSocioNuevo,string DNINuevo,
                 string nombreNuevo, string apellido1Nuevo,string apellido2Nuevo,
                 string direccionNueva,string localidadNueva,string provinciaNueva,
                 string codigoPostalNuevo,string telefonoNuevo,
                 string correoElectronicoNuevo,DateTime fechaNacimientoNueva)
           :base(DNINuevo,nombreNuevo,apellido1Nuevo,apellido2Nuevo,
                 direccionNueva,localidadNueva,provinciaNueva,codigoPostalNuevo,
                 telefonoNuevo,correoElectronicoNuevo,fechaNacimientoNueva)
    {
         NIS = siguienteNIS;
         siguienteNIS++;
         cuentaCorriente = cuentaCorrienteNueva;
         estado = "ACTUAL";
         tipoSocioPertenece = tipoSocioNuevo;
         tipoSocioPertenece.anyadirSocio(this);

         listaCuotasSocio = new List<CuotaSocio>();
         listaModalidadesTemporadas = new List<ModalidadTemporada>();
         listaTesoreroJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
         listaSecretarioJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
         listaVicepresidenteJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
         listaPresidenteJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
         listaVocalComisiones = new List<Comision>();
         listaActividadesSociedad = new List<ActividadSociedad>();
         listaCampeonatosSocio = new List<CampeonatoSocio>();
     }
     …
}
```

```
public class Socio:Participante
{
    //Declaración de atributos
    …
    //Constructores
    public Socio(int NISNuevo,string estadoNuevo,string cuentaCorrienteNueva,
                 TipoSocio tipoSocioNuevo, string DNINuevo, string nombreNuevo,
                 string apellido1Nuevo, string apellido2Nuevo,string direccionNueva,
                 string localidadNueva, string provinciaNueva, string codigoPostalNuevo,
                 string telefonoNuevo, string correoElectronicoNuevo,
                 DateTime fechaNacimientoNueva)
           :base(DNINuevo, nombreNuevo, apellido1Nuevo, apellido2Nuevo,
                 direccionNueva, localidadNueva, provinciaNueva, codigoPostalNuevo,
                 telefonoNuevo, correoElectronicoNuevo, fechaNacimientoNueva)
    {
       NIS = NISNuevo;
       estado = estadoNuevo;
       cuentaCorriente = cuentaCorrienteNueva;
       tipoSocioPertenece = tipoSocioNuevo;
       tipoSocioPertenece.anyadirSocio(this);

       listaCuotasSocio = new List<CuotaSocio>();
       listaModalidadesTemporadas = new List<ModalidadTemporada>();
       listaTesoreroJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
       listaSecretarioJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
       listaVicepresidenteJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
       listaPresidenteJuntasDirectivas = new SortedList<long, JuntaDirectiva>();
       listaVocalComisiones = new List<Comision>();
       listaActividadesSociedad = new List<ActividadSociedad>();
       listaCampeonatosSocio = new List<CampeonatoSocio>();
   }

   //Consultores de atributos
   public string getCuentaCorriente()
   {   return cuentaCorriente;}

   public TipoSocio getTipoSocio()
   {   return tipoSocioPertenece;}

   public List<CuotaSocio> getListaCuotasSocio()
   {   return listaCuotasSocio;}

       //Igual para el resto de atributos
       …

   //Modificadores de atributos
   public void setCuentaCorriente(string cuentaCorrienteNueva)
   {   cuentaCorriente = cuentaCorrienteNueva;}

   public void setTipoSocio(TipoSocio tipoSocioNuevo)
   {   tipoSocioPertenece = tipoSocioNuevo;}

       //Igual para el resto de atributos
       …
}
```

```
public class Socio:Participante
{
   //Declaración de atributos
   …
   //Constructores
   …
   //Consultores
   …
   //Modificadores
   …
   //Tratamiento de colecciones
   //Métodos añadir,borrar y buscar en la colección List listaCuotasSocios
   public void anyadirCuotaSocio(CuotaSocio cuotaSocioNueva)
   {
       bool existe = false;
       foreach(CuotaSocio cuotaSocioActual in listaCuotasSocio)
       {
           if(cuotaSocioActual.getTemporadaCuota().getNombre().Equals
                                       (cuotaSocioNueva.getTemporadaCuota().getNombre()))
           {
               existe = true;
               break;
           }
       }
       if(!existe) listaCuotasSocio.Add(cuotaSocioNueva);
       else throw new ElementoYaExistente("Ya existe la cuota del socio con los datos
                                          especificados.");
   }

   public void borrarCuotaSocio(CuotaSocio cuotaSocioBorrar)
   {
       bool existe = false;
       foreach(CuotaSocio cuotaSocioActual in listaCuotasSocio)
       {
           if(cuotaSocioActual.getTemporadaCuota().getNombre().Equals
                                      (cuotaSocioBorrar.getTemporadaCuota().getNombre()))
           {
               existe = true;
               break;
           }
       }
       if(existe) listaCuotasSocio.Remove(cuotaSocioBorrar);
       else throw new ElementoNoEncontrado("No se ha encontrado la cuota del socio con los
                                          datos especificados.");
   }

   public CuotaSocio buscarCuotaSocio(string nombreTemporada)
   {
       foreach(CuotaSocio cuotaSocioActual in listaCuotasSocio)
       {
           if(cuotaSocioActual.getTemporadaCuota().getNombre().Equals(nombreTemporada))
              return cuotaSocioActual;
       }
       throw new ElementoNoEncontrado("No se ha encontrado la cuota del socio de la
                                      temporada " + nombreTemporada + ".");
   }

   //Igual para el resto de colecciones List
   …
}
```

```
public class Socio:Participante
{
   //Declaración de atributos
   …
   //Constructores
   …
   //Consultores
   …
   //Modificadores
   …
   //Tratamiento de colecciones
   //Métodos añadir,borrar y buscar para la colección
   //SortedList listaTesoreroJuntasDirectivas
   public void anyadirTesoreroJuntaDirectiva(JuntaDirectiva juntaDirectivaNueva)
   {
       if (!listaTesoreroJuntasDirectivas.ContainsKey(juntaDirectivaNueva.getNumero()))
           listaTesoreroJuntasDirectivas.Add(juntaDirectivaNueva.getNumero(),
                                             juntaDirectivaNueva);
       else throw new ElementoYaExistente("Ya existe el socio como tesorero de la junta
                                          directiva actual");
   }

   public void borrarTesoreroJuntaDirectiva(JuntaDirectiva juntaDirectivaBorrar)
   {
       if (listaTesoreroJuntasDirectivas.ContainsKey(juntaDirectivaBorrar.getNumero()))
           listaTesoreroJuntasDirectivas.Remove(juntaDirectivaBorrar.getNumero());
       else throw new ElementoNoEncontrado("No se ha encontrado el socio como tesorero de
                                           la junta directiva actual");
   }

   public JuntaDirectiva buscarTesoreroJuntaDirectiva(long numeroJuntaDirectiva)
   {
       if (listaTesoreroJuntasDirectivas.ContainsKey(numeroJuntaDirectiva))
           return listaTesoreroJuntasDirectivas[numeroJuntaDirectiva];
       else throw new ElementoNoEncontrado("No se ha encontrado el socio como tesorero de
                                           la junta directiva indicada");
   }

   //Igual para el resto de colecciones SortedList
   …
   //Fin de clase
}
```

5.3. Diseño de base de datos

Los diseños orientados a objetos son eficientes, coherentes y menos proclives los problemas de actualización que aquejan a muchas otras técnicas de diseño de bases de datos. Pero, el modelo relacional ha ganado popularidad, por lo que se han incrementado sus ventajas en lo tocante a funcionalidad y flexibilidad. Además, a pesar de que las bases de datos orientadas a objetos tienen un aspecto prometedor, todavía no han alcanzado una amplia aceptación masiva por parte del mercado. El diagrama de clases se puede usar para implementar un diseño de una base de datos relacional, pero para traducir un diagrama de clases a tablas ideales, hay que proporcionar detalles que faltan en el diagrama, como la clave primaria y los candidatos a clave para cada tabla; así como si un atributo puede ser o no ser nulo y asignando un dominio a cada atributo.

A partir del diagrama de clases del presente caso de estudio se obtiene el siguiente diseño lógico relacional de la base de datos en el cual se muestran las tablas surgidas ordenadas alfabéticamente.

Actividad (nombre: tira (60),
fechaInicio: fecha,
id_Temporada: tira (20),
fechaFin: fecha,
lugar: tira(60),
descripcion: tira(MAX),
fechaInicioInscripcion: fecha,
fechaFinInscripcion: fecha,
cuotaInscripcionSocio: decimal,
participaNoSocio: tira(2)(“SI”,”NO”),
cuotaInscripcionNoSocio: decimal,
maximoParticipantes: entero)

Clave primaria: {nombre,
fechaInicio,
id_Temporada}

Clave ajena: {id_Temporada} hace referencia a Temporada

Valores no nulos: {fechaFin,
lugar,
descripcion,
fechaInicioInscripcion,
fechaFinInscripcion,
participaNoSocio,
maximoParticipantes}

ActividadSociedad (id_NombreActividad: tira (60),
id_fechaInicioActividad: fecha,
id_TemporadaActividad: tira (20))

Clave primaria: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad}

Clave ajena: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad} hace referencia a Actividad

ActividadSociedad-NoSocio (id_NombreActividad: tira (60),
id_fechaInicioActividad: fecha,
id_TemporadaActividad: tira (20),
id_NoSocio: tira (9))

Clave primaria: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad}

Clave ajena: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad} hace referencia a ActividadSociedad

Clave ajena: {id_NoSocio} hace referencia a NoSocio

ActividadSociedad-Socio (id_NombreActividad: tira (60),
id_fechaInicioActividad: fecha,
id_TemporadaActividad: fecha,
id_Socio: tira (9))

Clave primaria: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad}

Clave ajena: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad} hace referencia a ActividadSociedad

Clave ajena: {id_Socio} hace referencia a Socio

Actuacion (nombre: tira (40),
id_Temporada: tira (20),
descripcion: tira (MAX))

Clave primaria: {nombre,
id_Temporada}

Clave ajena: {id_Temporada} hace referencia a Temporada

Valor no nulo: {descripcion}

Actuacion-Zona (id_NombreActuacion: tira (40),
id_TemporadaActuacion: tira (20),
id_NombreZona: tira (40),
id_CotoZona: tira (20))

Clave primaria: {id_NombreActuacion,
id_TemporadaActuacion,
id_NombreZona,
id_CotoZona}

Clave ajena: {id_NombreActuacion,
id_TemporadaActuacion} hace referencia a Actuacion

Clave ajena: { id_NombreZona,
id_CotoZona} hacer referencia a Zona

Asamblea (nombre: tira (40),
fecha: fecha,
id_Temporada: tira (20),
lugar: tira (60),
id_JuntaDirectiva: entero)

Clave primaria: {nombre,
fechaInicio,
idTemporada}

Clave ajena: {idTemporada} hace referencia a Temporada

Clave ajena: {idJuntaDirectiva} hace referencia a JuntaDirectiva

Valor no nulo: {lugar, idJuntaDirectiva}

CampeonatoCazadores (id_NombreActividad: tira (60),
id_fechaInicioActividad: fecha,
id_TemporadaActividad: tira (20),
estadoFinalizacion: tira (2)(“SI”,”NO”))

Clave primaria: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad}

Clave ajena: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad} hace referencia a Actividad

Valor no nulo: {estadoFinalizacion}

CampeonatoCazadores-NoSocio (id_NombreActividad: tira (60),
id_fechaInicioActividad: fecha,
id_TemporadaActividad: tira (20),
id_NoSocio: tira (9)).

Clave primaria: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad,
id_Nosocio}.

Clave ajena: {id_NombreActividad,
id_fechaInicioActividad,
id_TemporadaActividad} hace referencia a Actividad

Clave ajena: {id_NoSocio} hace referencia a NoSocio

CampeonatoCazadores-Socio (id_NombreActividad: tira (60),
id_FechaInicioActividad: fecha,
id_TemporadaActividad: tira (20),
id_Socio: tira (9)).

Clave primaria: {id_NombreActividad,
id_FechaInicioActividad,
id_TemporadaActividad,

id_Nosocio}

Clave ajena: {id_NombreActividad,
id_FechaInicioActividad,
id_TemporadaActividad} hace referencia a Actividad

Clave ajena: {id_Socio} hace referencia a Socio

CampeonatoPerros (id_NombreActividad: tira (60),
id_FechaInicioActividad: fecha,
id_TemporadaActividad: tira (20),
estadoFinalizacion: tira(2)("SI","NO"))

Clave primaria: {id_NombreActividad,
id_FechaInicioActividad,
id_TemporadaActividad}

Clave ajena: {id_NombreActividad,
id_FechaInicioActividad,
id_TemporadaActividad} hace referencia a Actividad

Valor no nulo: {estadoFinalizacion}

CampeonatoPerros-PerroSocio (idNombreCampeonato: tira (60),
idFechaInicioCampeonato: fecha,
idTemporadaCampeonato: tira (20),
idPerro: tira (20),
idSocio: tira (9),
puntuacion: entero)

Clave primaria: {idNombreCampeonato,
idFechaInicioCampeonato,
idTemporadaCampeonato,
idPerro,
idSocio}

Clave ajena: {idNombreCampeonato,
idFechaInicioCampeonato,
idTemporadaCampeonato } hace referencia a CampeonatoPerros

Clave ajena: {idPerro,

idSocio} hace referencia a PerroSocio

Valor no nulo: {puntuacion}

CampeonatoPerros-PerroNoSocio (idNombreCampeonato: tira (60),
idFechaInicioCampeonato: fecha,
idTemporadaCampeonato: tira (20),
idPerro: tira (20),
idNoSocio: tira (9),
puntuacion: entero)

Clave primaria: {idNombreCampeonato,
idFechaInicioCampeonato,
idTemporadaCampeonato,
idPerro,
idNoSocio}

Clave ajena: {idNombreCampeonato,
idFechaInicioCampeonato,
idTemporadaCampeonato } hace referencia a CampeonatoPerros

Clave ajena: {idPerro,
idNoSocio } hace referencia a PerroNoSocio

Valor no nulo: {puntuacion}

Comision (id_TipoComision: tira (40),
id_JuntaDirectiva: entero).

Clave primaria: {id_TipoComision,
id_JuntaDirectiva}

Clave ajena: {id_TipoComision} hace referencia a Actividad.

Clave ajena: {id_JuntaDirectiva } hace referencia a JuntaDirectiva.

Comision-Socio (id_TipoComision: tira (40),
id_JuntaDirectiva: entero,
id_Vocal: tira (9))

Clave primaria: {id_TipoComision,
id_JuntaDirectiva,
id_Vocal}

Clave ajena: {id_TipoComision,
id_JuntaDirectiva} hace referencia a Comision.

Clave ajena: {id_Vocal} hace referencia a Socio.

Coto (codigo: tira (20),
nombre: tira (40),
superficie: decimal,
descripcion: tira (MAX)).

Clave primaria: {codigo}

Valor no nulo: {nombre,
superficie,
descripcion}

Cuota (nombre: tira (50),
id_Temporada: tira (20))

Clave primaria: {nombre,
id_Temporada}

Clave ajena: {id_Temporada} hace referencia Temporada.

CuotaEntrada (id_NombreCuota: tira (50),
id_TemporadaCuota: tira (20))

Clave primaria: {id_NombreCuota,
id_TemporadaCuota}

Clave ajena: {id_NombreCuota,
id_TemporadaCuota} hace referencia Cuota.

CuotaModalidad (id_NombreCuota: tira (50),
id_TemporadaCuota: tira (20))

Clave primaria: {id_NombreCuota,

id_TemporadaCuota}.

Clave ajena: {id_NombreCuota,
id_TemporadaCuota} hace referencia Cuota.

CuotaSociedad (id_NombreCuota: tira (50),
id_TemporadaCuota: tira (20))

Clave primaria: {id_NombreCuota,
id_TemporadaCuota}

Clave ajena: {id_NombreCuota,
id_TemporadaCuota} hace referencia Cuota.

CuotaSocio (id_Socio: tira (9),
id_Temporada: tira (20),
estadoConfirmacion: tira (2)(“SI”,”NO”),
fechaPago: fecha,
id_NombreCuotaEntrada: tira (50),
id_TemporadaCuotaEntrada: tira (20),
id_TipoSocio: tira (30))

Clave primaria: {id_Socio,
id_Temporada}

Clave ajena: {id_Socio} hace referencia Socio

Clave ajena: {id_Temporada} hace referencia Temporada

Clave ajena: {id_NombreCuotaEntrada,
id_TemporadaCuotaEntrada} hace referencia CuotaEntrada

Clave ajena: {id_TipoSocio} hace referenciaTipoSocio

Valor no nulo: {estadoConfirmacion, id_TipoSocio}

CuotaSocio-CuotaModalidad (id_SocioCuotaSocio: tira (9),
id_TemporadaCuotaSocio: tira (20),
id_NombreCuotaModalidad: tira (50),
id_TemporadaCuotaModalidad: tira (20))

Clave primaria: {id_SocioCuotaSocio,
id_TemporadaCuotaSocio
id_NombreCuotaModalidad,
id_TemporadaCuotaModalidad}

Clave ajena: {id_SocioCuotaSocio,
id_TemporadaCuotaSocio} hace referencia CuotaSocio

Clave ajena: {id_NombreCuotaModalidad,
id_TemporadaCuotaModalidad} hace referencia CuotaModalidad

EspecieCazable (nombreComun: tira (30),
nombreCientifico: tira (30),
tipoCaza: tira (20),
estado: tira(10)(“ACTUAL”,”HISTORIAL”))

Clave primaria: {nombreComun}

Valor no nulo: {tipoCaza,
estado}

Gasto (idNumeroMovimiento: entero,
idTemporadaMovimiento: tira (20))

Clave primaria: {idNumeroMovimiento,
idTemporadaMovimiento}

Clave ajena: {idNumeroMovimiento,
idTemporadaMovimiento}

Ingreso (idNumeroMovimiento: entero,
idTemporadaMovimiento: tira (20))

Clave primaria:{idNumeroMovimiento,
idTemporadaMovimiento}

Clave ajena: {idNumeroMovimiento,
idTemporadaMovimiento}

JuntaDirectiva (numero: entero,
id_Presidente: tira (9),
id_Vicepresidente: tira (9),
id_Secretario: tira (9),
id_Tesorero: tira (9))

Clave primaria: {numero}

Clave ajena: {id_Presidente} hace referencia a Socio

Clave ajena: {id_Vicepresidente} hace referencia a Socio

Clave ajena: {id_Secretario} hace referencia a Socio

Clave ajena: {id_Tesorero} hace referencia a Socio

Valor no nulo: {id_Presidente,
id_Vicepresidente,
id_Secretario,
id_Tesorero}

Mejora (id_NombreActuacion: tira (40),
id_TemporadaActuacion: tira (20))

Clave primaria: {id_NombreActuacion,
id_TemporadaActuacion}

Clave ajena: {id_NombreActuacion,
id_TemporadaActuacion} hace referencia a Actuacion

ModalidadCaza (nombre: tira (40),
descripcion: tira (MAX),
tipoCaza: tira (20),
estado: tira(10)(“ACTUAL”,”HISTORIAL”))

Clave primaria: {nombre}

Valor no nulo: {descripcion,
tipoCaza,
estado}

ModalidadTemporada (nombre: tira (40),
id_Temporada: tira (20),
id_NombreCuotaModalidad: tira (50),
id_TemporadaCuotaModalidad: tira (20))

Clave primaria: {nombre,
id_Temporada}

Clave ajena: {id_Temporada} hace referencia a Temporada

Clave ajena: {id_NombreCuotaModalidad,
id_TemporadaCuotaModalidad} hace referencia CuotaModalidad

Valor no nulo: {id_NombreCuotaModalidad,
id_TemporadaCuotaModalidad}

ModalidadTemporada-ModalidadCaza (id_NombreModalidad: tira (30),
id_TemporadaModalidad: tira (20),
id_ModalidadCaza: tira (40))

Clave primaria: {id_NombreModalidadTemporada,
id_TemporadaModalidadTemporada,
id_ModalidadCaza}.

Clave ajena: {id_NombreModalidad,
id_TemporadaModalidad} hace referencia a ModalidadTemporada

Clave ajena: {id_ModalidadCaza} hace referencia ModalidadCaza

ModalidadTemporada-Socio (id_NombreModalidad: tira (40),
id_TemporadaModalidad: tira (20),
id_Socio: tira (9))

Clave primaria: {id_NombreModalidad,
id_TemporadaModalidad,
id_Socio}

Clave ajena: {id_NombreModalidad,
id_TemporadaModalidad} hace referencia a ModalidadTemporada

Clave ajena: {id_Socio} hace referencia a Socio

MovimientoContable (numero: entero,
idTemporada: tira (20),
concepto: tira (50),
fecha: fecha,
precio: decimal)

Clave primaria: {numero,
idTemporada}

Clave ajena: {idTemporada} hace referencia a Temporada

Valor no nulo: {concepto,
fecha,
precio}

NoSocio (id_Participante: tira (9),
sociedadCazadores: tira (60))

Clave primaria: {id_Participante}

Clave ajena: {id_Participante} hace referencia a NoSocio

Participante (DNI: tira (9),
nombre: tira (30),
apellido1: tira (30),
apellido2: tira (30),
direccion: tira (100),
codigoPostal: tira (5),
localidad: tira (40),
provincia: tira (40),
telefono: tira(9),
correoElectronico: tira (60),
fechaNacimiento: fecha)

Clave primaria: {DNI}

Valor no nulo: {nombre,
apellido1,
apellido2,

telefono}

Perro (numeroIdentificacion: tira (20),
nombre: tira (30),
raza: tira (30),
sexo: tira(10)(“MACHO”,”HEMBRA”),
fechaNacimiento: fecha)

Clave primaria:{numeroIdentificacion}

Valor no nulo: {nombre,
raza,
sexo,
fechaNacimiento}

PerroSocio (idPerro: tira (20),
idSocio: tira (9),
estado: tira(10)(“ACTUAL”,”HISTORIAL”))

Clave primaria: {idPerro,
idNoSocio}

Clave ajena: {idPerro} hace referencia a Perro

Clave ajena: {idSocio} hace referencia a Socio

Valor no nulo: {estado}

PerroNoSocio (idPerro: tira (20),
idNoSocio: tira (9))

Clave primaria: {idPerro,
idNoSocio}

Clave ajena: {idPerro} hace refencia a Perro

Clave ajena: {idNoSocio} hace referencia a NoSocio

PuntoDia (numero: entero,
idNombreAsamblea: tira (40),
idFechaAsamblea: fecha,

idTemporadaAsamblea: tira (20),
asunto: tira (MAX),
votosFavor: entero,
votosContra: entero)

Clave primaria: {numero,
idNombreAsamblea,
idFechaAsamblea,
idTemporadaAsamblea}

Clave ajena: {idNombreAsamblea,
idFechaAsamblea,
idTemporadaAsamblea} hace referencia a Asamblea

Valor no nulo: {asunto}

Repoblacion (id_NombreActuacion: tira (40),
id_TemporadaActuacion: tira (20))

Clave primaria: {id_NombreActuacion,
id_TemporadaActuacion}

Clave ajena: {id_NombreActuacion,
id_TemporadaActuacion} hace referencia a Actuacion.

Repoblacion-EspecieCazable (id_NombreActuacion: tira (40),
id_TemporadaActuacion: tira (20),
id_EspecieCazable: tira (30),
numeroEjemplares: entero)

Clave primaria: {id_NombreRepoblacion,
id_TemporadaRepoblacion,
id_EspecieCazable}

Clave ajena: {id_NombreRepoblacion,
id_TemporadaRepoblacion} hace referencia a Repoblacion

Clave ajena: {id_EspecieCazable} hace referencia a EspecieCazable

Valor no nulo: {numeroEjemplares}

SociedadCazadores (CIF: tira (9),
nombre: tira (60),
logotipo: imagen,
direccion: tira (100),
codigoPostal: tira (5),
localidad: tira (40),
provincia: tira (40),
teléfono: tira (15),
correoElectronico: tira (60),
paginaWeb: tira (60),
cuentaCorriente: tira (23)
(X:1..9, "XXXX XXXX XX XXXXXXXXXX"))

Clave primaria: {CIF}

Valor no nulo: {nombre,
direccion,
codigoPostal,
localidad,
provincia,
telefono}

Socio (id_Participante: tira (9),
NIS: entero,
cuentaCorriente: tira (23) (X:1..9, "XXXX XXXX XX XXXXXXXXXX"),
id_TipoSocio: tira (30),
estado: tira (10)("ACTUAL","HISTORIAL"))

Clave primaria: {id_Participante}

Clave ajena: {id_Participante} hace referencia a Socio

Clave ajena: {id_TipoSocio} hace referencia a TipoSocio

Único: {NIS}

Valor no nulo: {NIS,
id_TipoSocio,
estado}

Temporada (nombre: tira(20),

añoInicio: entero,
estado: tira (20)(“EN CONFIGURACIÓN”,”EN CURSO”,”FINALIZADA”))

Clave primaria: {nombre}

Valor no nulo: {año,
estado}

Temporada-JuntaDirectiva (id_Temporada: tira (20),
id_JuntaDirectiva: entero)

Clave primaria: {id_Temporada,
id_JuntaDirectiva}

Clave ajena: {id_Temporada} hace referencia Temporada

Clave ajena: {id_JuntaDirectiva} hace referencia a JuntaDirectiva

TipoComision (nombre: tira (40),
descripcion: tira (MAX),
estado: tira (10)(“ACTUAL”,”HISTORIAL”)

Clave primaria: {nombre}

Valor no nulo: {descripcion,
estado}

TipoSocio (nombre: tira (30),
descripcion: tira (MAX),
estado: tira (10)(“ACTUAL”,”HISTORIAL”))

Clave primaria: {nombre}

Valor no nulo: {descripcion,
estado}

Veda (nombre: tira (40),
idTemporada: tira(20),
fechaInicio: fecha,
fechaFin: fecha,
descripcion: tira(MAX))

Clave primaria: {nombre, idTemporada}

Clave ajena: {idTemporada} hace referencia aTemporada

Valor no nulo: {fechaInicio,
fechaFin,
descripcion}

Zona (nombre: tira (40),
id_Coto: tira (20),
superficie: decimal,
descripcion: tira(MAX),
estado: tira (10)(“ACTUAL”,”HISTORIAL”))

Clave primaria: {nombre, id_Coto}

Clave ajena: {id_Coto} hace referencia a Coto

Valor no nulo: {superficie,
descripcion,
estado}

Atreves de este diseño lógico relacional se ha desarrollado la base de datos BD_SC mediante la aplicación SQL Server Management Studio Express.

Capítulo 6:

Implementación

6.1. Tecnología

La aplicación del proyecto se desarrolla bajo la tecnología .NET por medio de Microsoft Visual Studio 2008 (Microsoft Visual Studio) y atreves del lenguaje C# (Lenguaje C#).

La tecnología **.NET** es una plataforma de desarrollo de software desarrollada por Microsoft que hace énfasis en la transparencia de redes, orientado completamente a objetos, que es capaz de ejecutarse bajo cualquier plataforma y que permita un rápido desarrollo de aplicaciones, económico, a la vez que seguro y robusto.

.NET es la plataforma soportada por **Visual Studio** en esta aplicación Visual Studio 2008, entorno de desarrollo integrado para sistemas operativos Windows que soporta varios lenguajes de programación tales como visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic.NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

El lenguaje que se ha elegido para implementar la aplicación ha sido **C#**, cuyos principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

C# (pronunciado en inglés "C Sharp" y en español "C almohadilla") es el único lenguaje diseñado específicamente para ser utilizado en .NET, por lo que es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET, y de hecho gran parte de la librería de clases base de .NET ha sido escrito en este lenguaje.

C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado por Microsoft con el ambicioso objetivo de recoger las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, y combinarlas en uno sólo en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++.

Quizás el más directo competidor de C# es Java, lenguaje con el que guarda un enorme parecido en su sintaxis y características. En este aspecto, es importante señalar que C# incorpora muchos elementos de los que Java carece (sistema de tipos homogéneo, propiedades, indexadores, tablas multidimensionales, operadores redefinibles, etc.).

Ahora se expondrán las principales características de C#:

- Dispone de todas las características propias de cualquier lenguaje orientado a objetos: **encapsulación**, **herencia** y **polimorfismo**.

- Ofrece un **modelo de programación orientada a objetos homogéneos**, en el que todo el código se escribe dentro de clases y todos los tipos de datos, incluso los básicos, son clases que heredan de **System.Object**.

- Permite definir **estructuras**, que son clases un tanto especiales: sus objetos se almacenan en pila, por lo que se trabaja con ellos directamente y no referencias al montículo, lo que permite accederlos más rápido. Sin embargo, esta mayor eficiencia en sus accesos tiene también sus inconvenientes, fundamentalmente que el tiempo necesario para pasarlas como parámetros a métodos es mayor (hay que copiar su valor completo y

no sólo una referencia) y no admiten herencia (aunque sí implementación de interfaces).

- Es un **lenguaje fuertemente tipado**, lo que significa se controla que todas las conversiones entre tipos se realicen de forma compatible, lo que asegura que nunca se acceda fuera del espacio de memoria ocupado por un objeto. Así se evitan frecuentes errores de programación y se consigue que los programas no puedan poner en peligro la integridad de otras aplicaciones.

- Tiene a su disposición un **recolector de basura** que libera al programador de la tarea de tener que eliminar las referencias a objetos que dejen de ser útiles, encargándose de ello éste y evitándose así que se agote la memoria porque al programador olvide liberar objetos inútiles o que se produzcan errores porque el programador libere áreas de memoria ya liberadas y reasignadas.

- Incluye soporte nativo para **eventos** y **delegados**. Los delegados son similares a los punteros a funciones de otros lenguajes como C++ aunque más cercanos a la orientación a objetos, y los eventos son mecanismos mediante los cuales los objetos pueden notificar de la ocurrencia de sucesos. Los eventos suelen usarse en combinación con los delegados para el diseño de interfaces gráficas de usuario, con lo que se proporciona al programador un mecanismo cómodo para escribir códigos de respuesta a los diferentes eventos que puedan surgir a lo largo de la ejecución de la aplicación. (pulsación de un botón, modificación de un texto, etc.)

- Incorpora **propiedades**, que son un mecanismo que permite el acceso controlado a miembros de una clase tal y como si de campos públicos se tratasen. Gracias a ellas se evita la pérdida de legibilidad que en otros lenguajes causa la utilización de métodos *Set()* y *Get()* pero se mantienen todas las ventajas de un acceso controlado por estos proporcionada.

- Permite la **definición del significado de los operadores básicos** del lenguaje (+, -, *, &, ==, etc.) para nuestros propios tipos de datos, lo que facilita enormemente tanto la legibilidad de las aplicaciones como el esfuerzo necesario para escribirlas. Es más, se puede incluso definir el significado del operador ***[]*** en cualquier clase, lo que permite acceder a sus objetos tal y como si fuesen tablas. A la definición de éste último operador se le denomina **indizador**, y es especialmente útil a la hora de escribir o trabajar con colecciones de objetos.

- Admite unos elementos llamados **atributos** que no son miembros de las clases sino información sobre éstas que podemos incluir en su declaración. Por ejemplo, indican si un miembro de una clase ha de aparecer en la

ventana de propiedades de Visual Studio.NET, cuáles son los valores admitidos para cada miembro en ésta, etc.

¿Por qué esta tecnología? Porque nuestra aplicación se va a ejecutar bajo un entorno Windows y porque es una tecnología muy extendida y fácil de usar y además porque yo autor de este proyecto ya he trabajado con ella en otros trabajos.

6.2. Acceso a Datos: Escenario desconectado

Para detallar cómo se ha implementado la aplicación hay que explicar que la implementación por un lado ha dependido de cómo se ha realizado el acceso a la base de datos relacional.

El acceso a la base de datos se ha realizado mediante el escenario desconectado que proporciona ADO.NET.

ADO.NET es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos. Es una parte de la biblioteca de clases base que están incluidas en el Microsoft .NET Framework. Es comúnmente usado por los programadores para acceder y modificar los datos almacenados en un Sistema Gestor de base de datos Relacionales, aunque también puede ser usado para acceder a datos no relacionales.

La aplicación se ha realizado bajo el **escenario desconectado**, que es el entorno en el que una parte de los datos del servidor central se copia localmente y puede luego ser consultada y actualizada sin contar con una conexión abierta. Luego si se desea puede establecerse una conexión con el servidor de base de datos para sincronizar los cambios efectuados sobre la copia local y actualizar los datos.

El modelo de objetos que se han utilizado para llevar a cabo el escenario desconectado que accede a la base de datos Microsoft SQL Server utilizada por esta aplicación, son los que el proveedor de ADO.NET *Data Provider For SQL Server* nos proporciona a través de las clases que se encuentra en el *namespace System.Data.SqlCient*.

En la figura que se muestra a continuación se definen los objetos utilizados por el escenario desconectado además de cómo interactúan entre ellos.

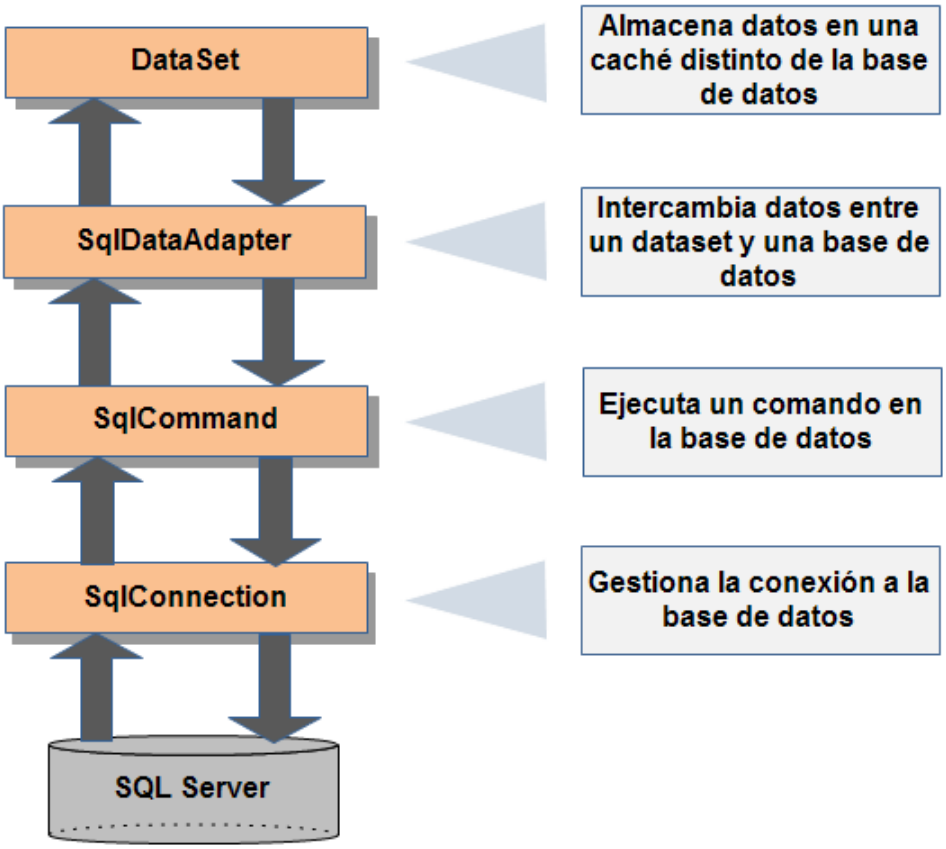

Figura 45. Esquema del escenario desconectado extendida de (DSW, Técnicas Avanzadas para Desarrollo de Software, 2008/09).

El acceso a los datos a través de un escenario desconectado es el siguiente:

1. Abrir Conexión.
2. Llenar *DataSet* mediante *DataAdapter*.
3. Cerrar Conexión.
4. Procesar *DataSet*.
5. Abrir conexión.
6. Actualizar fuente de datos mediante *DataAdapter*.
7. Cerrar Conexión.

Después de explicar el escenario desconectado, se mostrara detalles de la implementación teniendo en cuenta los pasos que sigue.

En momentos puntuales de la implementación, las explicaciones se centrarán en cómo se gestionan las modalidades de caza. Dichos detalles de implementación se pueden extrapolar para determinar cómo se realizan el resto de gestiones que soporta el sistema.

6.3. Detalles de implementación por capas

6.3.1. Implementación del inicio de la aplicación

En C#, el punto de entrada o inicio de una aplicación viene determinado por un método de clase (estático) con nombre *Main*. La plantilla para aplicaciones *Windows Forms* en Microsoft Visual Studio genera de forma automática una clase *Program* que implementa el método de *Main().* Dentro de este método se invoca al método *Run()* de la clase *Application*, pasándole como argumento una instancia del formulario principal, en este caso una instancia de *FormPrincipal*.

```
static class Program
{
      [STAThread]
      static void Main()
      {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new presentacion.FormPrincipal());
      }
}
```

6.3.2. Implementación de la creación de clases de comunicación de capas

Mediante la siguiente implementación se crean las clases que nos permite la comunicación entre capa de presentación y capa de negocio así como la comunicación entre capa de negocio y capa de persistencia.

```
Capa de presentación
public partial class FormPrincipal : FormBase
{
    // atributo herededado de FormBase
    private SociedadCazadores socCazadores socCazadores;

    public FormPrincipal(): base()
    {
        InitializeComponent();
        negocio.ConexionBD conexionBDSC = new negocio.ConexionBD();
        …
    }
    …
}
```

```
Capa de negocio
public class ConexionBD
{
     …
     public ConexionBD()
     {
          sisCentral = new persistencia.SistemaCentral();
     }
```

```
Capa de persistencia
public class SistemaCentral
{
      private SqlConnection conexionBDSC;
      …
      public SistemaCentral()
      {
          string cadenaConexion = @"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\BD_SC.mdf;
                              Integrated Security=True;Connect Timeout=30;User Instance=True";
          conexionBDSC = new SqlConnection(cadenaConexion);
      }
```

6.3.3. Implementación de carga del DataSet

A continuación se realiza la carga de la instacia del *DataSet* de la aplicación, *DataSetBD_SC*, llamada *dsSC*. Para ello el orden que va a seguir la implemnetación es de los tres primeros pasos que establece el escenario desconectado:

1. Abrir conexión.
2. Llenar *DataSet* mediante *DataAdapter*.
3. Cerrar conexión.

```
Capa de presentación
public partial class FormPrincipal : FormBase
{
    // atributo herededado de FormBase
    private SociedadCazadores socCazadores socCazadores;

    public FormPrincipal(): base()
    {
        InitializeComponent();
        negocio.ConexionBD conexionBDSC = new negocio.ConexionBD();
        // establecer la conexión con la base de datos y cargar DataSet
        if(conexionBDSC.cargarBaseDatos(out socCazadores))
        {
           …
        }
        else   // no se ha realizado la conexión con la base de datos
        {
            MessageBox.Show("No se ha abierto la conexión con la base de datos BD_SC",
                            "Conexión errónea", MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.Close();
        }
     }
```

Capa de negocio

```
public class ConexionBD
{
    private persistencia.SistemaCentral sisCentral;
    // establecer la conexión con la base de datos y cargar DataSet
    public bool cargarBaseDatos(out SociedadCazadores socCazadoresActual)
    {
        // 1.Abrir conexión
        if (sisCentral.conectarBaseDatos())
        {   // la conexión se ha realizado con éxito
            // 2.Llenar DataSet mediante DataAdapter
            sisCentral.cargarDataSet();
            // 3.Cerra conexión
            sisCentral.desconectarBaseDatos();
            …
        }
        else
        {// la conexión no se ha realizado éxito
            socCazadoresActual = null;
            return false;
        }
    }
```

Capa de persistencia

```
public class SistemaCentral
{
    private SqlConnection conexionBD;
    private DataSetBD_SC dsSC;
    private SqlDataAdapter daModalidadCaza;
    // 1.Abrir conexión
    public bool conectarBaseDatos()
    {
        try
        {
            conexionBD.Open();
            return true;  // la conexión se ha realizado con éxito
        }
        catch
        {
             return false;  // la conexión no se ha realizado
        }
    }
    // 2.Llenar DataSet mediante DataAdapter
    public void cargarDataSet()
    {
        dsSC = new DataSetBD_SC();
        …
        daModalidadCaza = new SqlDataAdapter("select * from ModalidadCaza", conexionBD);
        SqlCommandBuilder cmdbModalidadCaza = new SqlCommandBuilder(daModalidadCaza);
        daModalidadCaza.Fill(dsSC, "ModalidadCaza");
        …
    }
    // 3.Cerra conexión
    public void desconectarBaseDatos()
    {
        if (conexionBD.State == System.Data.ConnectionState.Open)
        {
            conexionBD.Close();
        }
    }
```

6.3.4. Implementación de la carga del sistema

Tras haber llenado el *DataSet*, ya se han obtenido los datos que almacena la base de datos. En este punto la aplicación se situa en el paso 4 que establece el escenario desconectado, Procesar *DataSet*.

Es el turno de ver como se transfieren los datos cargados en el *DataSet* a las diferentes capas de la aplicación. Lo que permitirá que se puedan manipular la información de la sociedad de cazadores.

Capa de presentación

```
public partial class FormPrincipal : FormBase
{
    // atributo herededado de FormBase
    private SociedadCazadores socCazadores socCazadores;

    public FormPrincipal(): base()
    {
        InitializeComponent();
        negocio.ConexionBD conexionBDSC = new negocio.ConexionBD();
        // obtener la información de la sociedad de cazadores
        if(conexionBDSC.cargarBaseDatos(out socCazadores))
        {
           …
        }
        else
        {
            MessageBox.Show("No se ha abierto la conexión con la base de datos BD_SC",
                            "Conexión errónea",MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.Close();
        }
     }
```

Capa de negocio

```
public class ConexionBD
{
    private persistencia.SistemaCentral sisCentral;
    private SociedadCazadores socCazadores;
    …
    public bool cargarBaseDatos(out SociedadCazadores socCazadoresAct)
    {
        if (sisCentral.conectarBaseDatos())
        {
            sisCentral.cargarDataSet();
            sisCentral.desconectarBaseDatos();
            // obtener los datos de la sociedad de cazadores
            cargarSociedadCazadores();
            socCazadoresActual = socCazadores;
            return true;
        }
        else
        {
            socCazadoresActual = null;
            return false;
        }
    }
    // obtener los datos de la sociedad de cazadores
    public void cargarSociedadCazadores()
    {
        socCazadores = null;
        // comprobar si hay datos en la base de datos
        if (sisCentral.existeSociedadCazadores())
        {
            …
            string descripcion;
            string tipoCaza;
            string estado;
            List<string> idsModalidadesCaza;
            // comprobar si se han obtenido las claves primarias de las modalidades de caza
            if (sisCentral.getModalidadesCaza(out idsModalidadesCaza))
            {
               foreach (string nombreModalidadCazaActual in idsModalidadesCaza)
               {
                   ModalidadCaza nuevaModalidadCaza;
                   // comprobar si se han obtenido el resto de datos de una modalidad de caza
                   if(sisCentral.getModalidadCaza(nombreModalidadCazaActual,out descripcion,
                                                  out tipoCaza, out estado))
                   {
                       // crear modalidad de caza
                       nuevaModalidadCaza = new ModalidadCaza(nombreModalidadCazaActual,
                                                              descripcion,tipoCaza,estado);
                       // añadir la modalidad de caza a la sociedad de cazadores
                       socCazadores.anyadirModalidadCaza(nuevaModalidadCaza);
                   }
                   else throw new Exception("Error al leer los datos de la modalidad de
                                            caza " + nombreModalidadCazaActual);
               }
            }
            …
            // comunicar SociedadCazadores con ConexionBDSC
            socCazadores.setConexionBDSC(this);
        }
    }
```

Capa de negocio

```
public class ModalidadCaza
{
    private string nombre;
    private string tipoCaza;
    private string estado;
    // crear modalidad de caza
    public ModalidadCaza(string nombreNuevo,string tipoCazaNueva,string estadoNuevo)
    {
        nombre = nombreNuevo;
        tipoCaza = tipoCazaNueva;
        estado = estadoNuevo;
    }
    …
}

public class SociedadCazadores
{
     private List<ModalidadCaza> listaModalidadesCaza;
     …
     // añadir la modalidad de caza a la sociedad de cazadores
     public void anyadirModalidadCaza(ModalidadCaza modalidadCazaNueva)
     {
         bool existe = false;
         foreach (ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
         {
             if (modalidadCazaActual.getNombre().Equals(modalidadCazaNueva.getNombre()))
             {
                 existe = true;
                 break;
             }
         }
         if (!existe)
            listaModalidadesCaza.Add(modalidadCazaNueva);
         else throw new ElementoYaExistente("Ya existe una modalidad de caza con los datos
                                             especificados.");
      }
```

6.3.5. Implementación de consulta de modalidades

En los siguientes fragmentos de código se muestra como se obtienen el conjunto de modalidades de caza y como se muestran a través de el formulario de Gestión de modalidades de caza.

Capa de presentación

```
public partial class FormModalidadesCaza : FormListaDetalle// formulario heredado de FormBase
{
    …
    public void cargarModalidadesCaza()
    {
        List<negocio.ModalidadCaza> listaModalidadesCaza = new List<negocio.ModalidadCaza>();
        // obtener las modalidades de caza que se quieren mostrar
        if (rbtnActuales.Checked)
            listaModalidadesCaza = socCazadores.getListaModalidadesCazaActuales();
        else listaModalidadesCaza = socCazadores.getListaModalidadesCazaHistorial();
        dgModalidadesCaza.Rows.Clear();
        // mostrar las modalidades de caza obtenidas en un Datagrid(lista en tabla)
        foreach (negocio.ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
        {
            dgModalidadesCaza.Rows.Add(modalidadCazaActual.getNombre(),
                                       modalidadCazaActual.getTipoCaza());
        }
        dgModalidadesCaza.ClearSelection();
     }
```

Capa de negocio

```
public class SociedadCazadores
{
    …
    private List<ModalidadCaza> listaModalidadesCaza;
    …
    //obtener las modalidades de caza actuales,es decir modalidadese de caza cuyo estado
      sea igual a “ACTUAL”
    public List<ModalidadCaza> getListaModalidadesCazaActuales()
    {
        List<ModalidadCaza> listaModalidadesCazaActuales = new List<ModalidadCaza>();
        foreach (ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
        {
            if (modalidadCazaActual.getEstado().Equals("ACTUAL"))
                listaModalidadesCazaActuales.Add(modalidadCazaActual);
        }
        return listaModalidadesCazaActuales;
    }
    //obtener las modalidades de caza en historial,es decir las modalidades de caza cuyo
      estado sea igual a “HISTORIAL”
    public List<ModalidadCaza> getListaModalidadesCazaHistorial()
    {
        List<ModalidadCaza> listaModalidadesCazaHistorial = new List<ModalidadCaza>();
        foreach (ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
        {
            if (modalidadCazaActual.getEstado().Equals("HISTORIAL"))
                listaModalidadesCazaHistorial.Add(modalidadCazaActual);
        }
        return listaModalidadesCazaHistorial;
    }
```

6.3.6. Implementación de alta de modalidad de caza

Tras haber solicitado dar de alta a una nueva modalidad de caza y haber introducido la información que identificará a dicha modalidad de caza la aplicación sigue la implementación que se muestra a continuación.

Capa de presentación

```
public partial class FormModalidadesCaza : FormListaDetalle
{
    …
    // se ha hecho clic en el botón guardar
    private void btnGuardar_Click(object sender, EventArgs e)
    {
        // comprobar que se han introducido correctamente los datos
        if (!camposErroneos())
        {
            if (tabDetalle.TabPages[0].Text.Equals("Nueva Modalidad de Caza"))
                guardarModalidadCaza();
            else modificarModalidadCaza();
        }
    }
    // comprobar que se han introducido correctamente los datos
    private bool camposErroneos()
    {
        bool camposErroneos = false;
        errorCampo.Clear();
        if (txtNombre.Text.Equals(""))
        {
            // Indica que no se ha introducido el nombre de la modalidad de caza
            errorCampo.SetError(txtNombre, "¿Qué nombre tiene?");
            camposErroneos = true;
        }
        return camposErroneos;
    }
```

Capa de presentación

```
public partial class FormModalidadesCaza : FormListaDetalle
{
    private negocio.ModalidadCaza modalidadCazaActual;
    …
    // guardar modalidad de caza
    private void guardarModalidadCaza()
    {
        // comprobar que no existe otra modalidad de caza con el mismo nombre(clave primaria)
        if (!nombreErroneo())
        {
            // no existe modalidad de caza con el mismo nombre
            string nombre = txtNombre.Text.Trim();
            string tipoCaza = "";
            if (rbtnCazaMenor.Checked)
                tipoCaza = "Caza menor";
            else tipoCaza = "Caza mayor";
            // crear modalidad de caza
            socCazadores.crearModalidadCaza(nombre,tipoCaza);
            MessageBox.Show("Guardada la información de la modalidad de caza "+nombre+ ".",
            "Modalidad de Caza guardada", MessageBoxButtons.OK, MessageBoxIcon.Information);
            //actualizar el Datagrid
            dgModalidadesCaza.Rows.Add(nombre,tipoCaza);
            establecerInterfazActuales();
        }
        else// existe modalidad de caza con el mismo nombre
        {
            // comprobar si la modalidad de caza existente es actual o es de historial
            if (modalidadCazaActual.getEstado().Equals("ACTUAL"))
            {
                MessageBox.Show("Existe la modalidad de caza " + txtNombre.Text + " como
                                 modalidad de caza actual.", "Modalidad de Caza existente",
                                 MessageBoxButtons.OK, MessageBoxIcon.Error);
                …
            }
            else
            {
                string nombreModalidadCaza = modalidadCazaActual.getNombre();
                MessageBox.Show("Existe la modalidad de caza " + nombreModalidadCaza +
                               "en el historial.","Modalidad de caza existente",
                                MessageBoxButtons.OK,MessageBoxIcon.Information);
                …
            }
        }
    }
   // comprobar que no existe otra modalidad de caza con el mismo nombre(clave primaria)
    private bool nombreErroneo()
    {
        try
        {
            modalidadCazaActual = socCazadores.buscarModalidadCaza(txtNombre.Text.Trim());
            return true;
        }
        catch
        {
            return false;
        }
    }
```

Capa de negocio

```
public class SociedadCazadores
{
    private ConexionBD conexionBDSC;
    static private persistencia.SistemaCentral sisCentral;
    private List<TipoSocio> listaTiposSocio;
    …
    // crear modalidad de caza
    public void crearModalidadCaza(string nombre,string tipoCaza)
    {
         // crear modalidad de caza
         ModalidadCaza modalidadCazaNueva = new ModalidadCaza(nombre,tipoCaza);
         // añadir la modalidad de caza a la sociedad de cazadores
         anyadirModalidadCaza(modalidadCazaNueva);
         // insertar la nueva fila de modalidad de caza en el DataSet
         sisCentral.insertarModalidadCaza(nombre, descripcion, tipoCaza,"ACTUAL");
    }

    // buscar modalidad de caza
    public ModalidadCaza buscarModalidadCaza(string nombreModalidad)
    {
         foreach (ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
         {
             if (modalidadCazaActual.getNombre().Equals(nombreModalidad))
                    return modalidadCazaActual;
         }
         throw new ElementoNoEncontrado("No se ha encontrado una modalidad de caza con
                                         nombre: " + nombreModalidad + ".");
    }
```

Capa de persistencia

```
public class SistemaCentral
{
    private DataSetBD_SC dsSC;
    …
    // insertar la nueva fila de modalidad de caza en el DataSet
    public void insertarModalidadCaza(string nombre,string tipoCaza,string estado)
    {
        DataSetBD_SC.ModalidadCazaRow filaModalidadCaza =
                                                  dsSC.ModalidadCaza.NewModalidadCazaRow();
        filaModalidadCaza.nombre = nombre;
        filaModalidadCaza.tipoCaza = tipoCaza;
        filaModalidadCaza.estado = estado;
        dsSC.ModalidadCaza.AddModalidadCazaRow(filaModalidadCaza);
     }
```

6.3.7. Implementación de modificación de modalidad de caza

Tras haber solicitado modificar la modalidad de cazada que se haya especificado y haber modificado la información que identificará a dicha modalidad de caza la aplicación sigue el siguiente código.

Capa de presentación

```
public partial class FormModalidadesCaza : FormListaDetalle
{
    // se ha hecho clic en el botón guardar
    private void btnGuardar_Click(object sender, EventArgs e)
    {
        if (!camposErroneos())
        {
            if (tabDetalle.TabPages[0].Text.Equals("Nueva Modalidad de Caza"))
                guardarModalidadCaza();
            else modificarModalidadCaza();
        }
    }
    private void modificarModalidadCaza()
    {
        bool correcto = true;
        // obtener la modalidad de caza a partir de su nombre
        string nombreActual = dgModalidadesCaza.SelectedCells[0].Value.ToString();
        negocio.ModalidadCaza modalidadCazaModificar =
                                              socCazadores.buscarModalidadCaza(nombreActual);
        string nombreNuevo = txtNombre.Text.Trim();
        // comprobar si se ha modificado el nombre de la modalidad de caza
        if(!nombreNuevo.Equals(modalidadCazaModificar.getNombre()))
        // comprobar que no existe otra modalidad de caza con el mismo nombre(clave primaria)
            if (nombreErroneo()) correcto = false;
        if(correcto)
        {
            string tipoCaza = "";
            if (rbtnCazaMenor.Checked)
                tipoCaza = "Caza menor";
            else tipoCaza = "Caza mayor";
            // modificar datos de modalidad de caza
            socCazadores.modificarModalidadCaza(modalidadCazaModificar,nombreNuevo,tipoCaza);
            MessageBox.Show("Modificada la información de la modalidad de caza "+
                            txtNombre.Text+".","Información de Modalidad de caza modificada",
                             MessageBoxButtons.OK, MessageBoxIcon.Information);
            // actualizar información del Datagrid
            dgModalidadesCaza.SelectedCells[0].Value = nombreNuevo;
            dgModalidadesCaza.SelectedCells[2].Value = tipoCaza;
            …
        }
        else // existe otra modalidad de caza con el mismo nombre(clave primaria)
        {   // comprobar si la modalidad de caza existente es actual o es de historial
            if(modalidadCazaActual.getEstado().Equals("ACTUAL"))
               MessageBox.Show("Existe la modalidad de caza+modalidadCazaActual.getNombre()
                            +" como modalidad de caza actual.","Modalidad de Caza existente",
                            MessageBoxButtons.OK, MessageBoxIcon.Error);
            else MessageBox.Show("Existe la modalidad de caza "+
                                  modalidadCazaActual.getNombre()+
                                 " en el historial.", "Modalidad de Caza existente",
                                 MessageBoxButtons.OK, MessageBoxIcon.Information)
        }
```

Capa de negocio

```
public class SociedadCazadores
{
    static private persistencia.SistemaCentral sisCentral;
    …
    public void modificarModalidadCaza(ModalidadCaza modalidadCazaActual,
                                       string nombre,string tipoCaza)
    {
        string nombreActual = modalidadCazaActual.getNombre();
        //comprobar si se ha modificado el nombre
        if (!nombre.Equals(nombreActual))
        {
            //modificar el nombre
            modalidadCazaActual.setNombre(nombre);
            //modificar nombre de la fila de modalidad de caza del DataSet
            sisCentral.setNombreTipoSocio(nombreActual, nombre);
        }
        //comprobar si se ha modificado el tipo caza
        if (!tipoCaza.Equals(modalidadCazaActual.getTipoCaza()))
        {
            //modificar el tipo de caza de la capa de negocio
            modalidadCazaActual.setTipoCaza(tipoCaza);
            //modificar el tipo de caza de la capa de persistencia
            sisCentral.setTipoCazaModalidadCaza(nombre, tipoCaza);
        }
    }
    …
}

public class ModalidadCaza
{
    private string nombre;
    private string tipoCaza;
    …
    //consultores
    public string getNombre()
    {   return nombre;}

    public string getTipoCaza()
    {   return tipoCaza;}

    //modificadores
    public void setNombre(string nombreNuevo)
    {   nombre = nombreNuevo;}

    public void setTipoCaza(string tipoCazaNueva)
    {   tipoCaza = tipoCazaNueva;}
    …
}
```

Capa de persistencia

```
public class SistemaCentral
{
    private DataSetBD_SC dsSC;
    …
    //modificar nombre de la fila de modalidad de caza del DataSet
    public void setNombreModalidadCaza(string nombreActual, string nombreNuevo)
    {
       //encontrar la fila de la modalidad de caza mediante el nombre
       DataSetBD_SC.ModalidadCazaRow filaModalidadCaza =
                                              dsSC.ModalidadCaza.FindBynombre(nombreActual);
        filaModalidadCaza.nombre = nombreNuevo;
    }
    //modificar tipo de caza de la fila de modalidad de caza del DataSet
    public void setTipoCazaModalidadCaza(string nombre, string tipoCaza)
    {
        DataSetBD_SC.ModalidadCazaRow filaModalidadCaza =
                                                 dsSC.ModalidadCaza.FindBynombre(nombre);
        filaModalidadCaza.tipoCaza = tipoCaza;
    }
```

6.3.8. Implementación de baja de modalidad de caza

Tras haber solicitado eliminar la modalidad de caza que se haya especificado la implementación que se realiza para dar de baja una modalidad de caza es la siguiente.

```
Capa de presentación
public partial class FormModalidadesCaza : FormListaDetalle
{
    private negocio.ModalidadCaza modalidadCazaActual;
    …
    //se ha hecho clic en el botón eliminar
    private void btnEliminar_Click(object sender, EventArgs e)
    {
        //comprobar si se ha seleccionado una modalidad de caza del Datagrid
        if (dgModalidadesCaza.SelectedRows.Count != 0)
        {
            //obtener la modalidad de caza a partir de su nombre
            DataGridViewRow filaSeleccionada = dgModalidadesCaza.SelectedRows[0];
            string nombre = filaSeleccionada.Cells[0].Value.ToString();
            modalidadCazaActual = socCazadores.buscarModalidadCaza(nombre);
            //comprobar que no se ha practicado en ninguna temporada
            if (modalidadCazaActual.getListaModalidadesTemporada().Count == 0)
            {
                 //eliminar modalidad de caza;
                socCazadores.eliminarModalidadCaza(modalidadCazaActual);
                MessageBox.Show("Eliminada la modalidad de caza "+nombre,
                                "Modalidad de caza eliminada",
                                MessageBoxButtons.OK, MessageBoxIcon.Information);
                //actualizar Datagrid
                dgModalidadesCaza.Rows.Remove(filaSeleccionada);
                establecerInterfazActuales();
            }
            else //se ha practicado la modalidad en alguna temporada
            {
               //comprobar si se practica en la temporada actual
               if (socCazadores.existeModalidadCazaTemporadaActual(modalidadCazaActual))
               {
                    //se practica en la temporada actual,no se puede eliminar
                    MessageBox.Show("No se puede eliminar una modalidad de caza que se esta
                                     practicando en la temporada actual.",
                                    "Modalidad de Caza no eliminada",
                                    MessageBoxButtons.OK, MessageBoxIcon.Information);
               }
               else //no se practica en la temporada actual
               {
                  //la modalidad de caza pasa a formar parte del historial
                  socCazadores.modificarEstadoModalidadCaza(modalidadCazaActual,"HISTORIAL");
                   MessageBox.Show("Eliminada de las modalidades de caza actuales la
                                    modalidad de caza " + nombre + ".",
                                   "Modalidad de Caza en historial",
                                   MessageBoxButtons.OK, MessageBoxIcon.Information);
                   dgModalidadesCaza.Rows.Remove(filaSeleccionada);
                   establecerInterfazActuales();
               }
            }
        }
        else MessageBox.Show("Haz clic en la modalidad de caza que desea eliminar.",
                             "Modalidad de Caza no seleccionada",
    }                        MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Capa de negocio

```
public class SociedadCazadores
{
    private ConexionBD conexionBDSC;
    static private persistencia.SistemaCentral sisCentral;
    private List<TipoSocio> ListaModalidadesCaza;
    …
    //eliminar la modalidad de caza del sistema
    public void eliminarModalidadCaza(ModalidadCaza modalidadCazaEliminar)
    {
        //eliminar la modalidad de caza de la capa de negocio
        borrarModalidadCaza(modalidadCazaEliminar);
        //eliminar la fila de modalidad de caza de la capa de persistencia
        sisCentral.eliminarModalidadCaza(modalidadCazaEliminar.getNombre());
    }
    //modificar estado de la modalidad de caza del sistema
    public void modificarEstadoModalidadCaza(ModalidadCaza modalidadCazaActual,string estado)
    {
        //modificar estado de la modalidad de caza de la capa de negocio
        modalidadCazaActual.setEstado(estado);
        //modificar estado de la modalidad de caza de la capa de persistencia
        sisCentral.setEstadoModalidadCaza(modalidadCazaActual.getEstado(), estado);
    }
    //eliminar la modalidad de caza de la capa de negocio
    public void borrarModalidadCaza(ModalidadCaza modalidadCazaBorrar)
    {
        bool existe = false;
        foreach (ModalidadCaza modalidadCazaActual in listaModalidadesCaza)
        {
            if (modalidadCazaActual.getNombre().Equals(modalidadCazaBorrar.getNombre()))
            {
                existe = true;
                break;
            }
        }
        if (existe)
            listaModalidadesCaza.Remove(modalidadCazaBorrar);
        else throw new ElementoNoEncontrado("No se ha encontrado una modalidad de caza con
                                            los datos epecificados.");
     }
```

Capa de persistencia

```
public class SistemaCentral
{
    private DataSetBD_SC dsSC;
    …
    //eliminar la fila de modalidad de caza del DataSet
    public void eliminarModalidadCaza(string nombre)
    {
        //encontrar la fila de la modalidad de caza mediante el nombre
        DataSetBD_SC.ModalidadCazaRow filaModalidadCaza =
                                                dsSC.ModalidadCaza.FindBynombre(nombre);
        filaModalidadCaza.Delete();
    }
    //modificar el estado de la fila de modalidad de caza del DataSet
    public void setEstadoModalidadCaza(string nombre, string estado)
    {
        DataSetBD_SC.ModalidadCazaRow filaModalidadCaza =
                                                dsSC.ModalidadCaza.FindBynombre(nombre);
        filaModalidadCaza.estado = estado;
    }
```

6.3.9. Implementación de actualizar la información de la base de datos

Tras haber manipulado con los datos de la aplicación y despues de solicitar el cierre de aplicación se actualiza los datos de la base de datos. Se realizan los ultimos tres pasos que especifica el escenario desconectado

1. Abrir conexión.
2. Actualizar fuente de datos mediante *DataAdapter*.
3. Cerrar conexión.

Capa de presentación

```
//Para cualquier formulario que cierre la aplicación
private void cerrarAplicacion()
{
     //establecer conexión y actualizar la base de datos
     if (socCazadores.getConexionBDSC().actualizarBaseDatos())
         this.Close();
     else MessageBox.Show("No se ha podido realizar la conexión con la base de datos
                          BD_SC", "Conexión errónea",
                           MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Capa de negocio

```
public class ConexionBD
{
    private persistencia.SistemaCentral sisCentral;
    private SociedadCazadores socCazadores;
    …
    //establecer conexión y actualizar la base de datos
    public bool actualizarBaseDatos()
    {
        //5.Abrir conexión
        if (sisCentral.conectarBaseDatos())
        {
            //la conexión se ha realizado con éxito
            //6.Actualizar DataSet mediante DataAdapter
            sisCentral.actualizarBaseDatos();
            //7.Cerrar conexión
            sisCentral.desconectarBaseDatos();
            return true;
        }
        else return false;
    }
```

Capa de persistencia

```
public class SistemaCentral
{
    private SqlConnection conexionBD;
    private DataSetBD_SC dsSC;
    private SqlDataAdapter daModalidadCaza;
    …
    //6.Actualizar DataSet mediante DataAdapter
    public void actualizarBaseDatos()
    {
        daModalidadCaza.Update(dsSC,"ModalidadCaza");
        daModalidadCaza.Fill(dsSC);
    }
```

Capítulo 7:

Conclusiones

En este apartado se verán las conclusiones extraídas a lo largo del desarrollo del proyecto. Se expondrá el trabajo que se ha realizado, la valoración personal del mismo así como las posibles ampliaciones del sistema de información desarrollado.

7.1. Trabajo realizado

El proyecto realizado responde a las expectativas y requerimientos que se pusieron sobre la mesa en el punto de partida del mismo. Realizar mediante la metodología de desarrollo RUP una aplicación implementada bajo el lenguaje C# en la plata y con arquitectura de 3 capas.

Los artefactos RUP desarrollados para lograr la consecución de dicha aplicación son los principales de la metodología: casos de uso, diagrama de clase y diagramas de secuencia.

La aplicación realizada ofrece al usuario una interfaz intuitiva y de fácil manejo en la que se ha logrado cubrir las funcionalidades que le permiten a una sociedad de cazadores realizar la completa gestión de los socios y de las juntas directivas así como de la mayoría de gestiones que se realizan a lo largo de una temporada como son la gestión de cuotas, de modalidades, de actividades, de actuaciones en el coto de caza y de la contabilidad.

Para que la información gestionada en la aplicación tenga persistencia se ha construido una base de datos SQL Server 2005 realiza mediante el programa SQL Server Management Studio Express.

7.2. Valoración personal

Voy a comenzar este apartado exponiendo las principales dificultades encontradas durante la realización del proyecto, que una vez superadas, han producido en mi mayor satisfacción.

La primera dificultad que me encontré fue a la hora de realizar el análisis de requisitos, es decir, al enfrentarme a acotar el problema al que me iba a enfrentar. Conocía que existían sociedades de cazadores, que los cazadores madrugaban los fines de semana para irse a cazar y poco más. Esta falta de conocimiento supuso que tuviera que dedicar más tiempo del que a priori había estimado para documentarme a través de cazadores conocidos y de páginas web de cómo era el funcionamiento de una sociedad de cazadores.

Otra dificultad a la que me enfrente fue la de encontrar la forma de acceder a la base de datos teniendo en cuenta que estaba realizando una arquitectura de tres capas. Yo había realizado acceso a base de datos en otros proyectos pero en todos ellos comunicaba directamente la capa de presentación con la capa de persistencia. En cambio en este proyecto he tenido que resolver la comunicación entre las capas de presentación y lógica, y entre las capas de lógica y de persistencia.

Y por último quiero destacar la dificultad para realizar el diseño de la interfaz, el propósito buscado era que fuera interfaz cómoda de usar, en la que no se necesitara abrir demasiados formularios para realizar una gestión. Ello ha hecho necesario el desarrollado de formularios contenedores de otros. Lo que ha añadido la dificultad de comunicación entre los formularios contenedores y los formularios que contiene.

Por la superación de estas y otras dificultades y por la consecución de un proyecto de principio a fin, considero que he hecho un buen trabajo. Trabajo que me

ha llevado más tiempo del que esperaba, pero en el que he adquirido muchos conocimientos. Me ha ayudado a conocerme más a mí mismo, haciéndome ver que puedo ser muy cabezón y que a veces me ofusco demasiado en problemas que tienen poco peso en el proyecto.

En conclusión, me ha resultado una experiencia muy positiva con la que estoy muy satisfecho.

7.3. Ampliaciones futuras

Este trabajo realizado se puede continuar completando las funcionalidades que al final la aplicación no ha soportado. Dichas funcionalidades son los que hacen referencias a la gestión de los perros que participan en la sociedad de cazadores, la gestión de las asambleas que convoca las juntas directivas y las vedas que establecen las instituciones para cada temporada.

Por otro lado una posible y necesaria ampliación sería la de realizar una página web de la sociedad de cazadores. En dicha página se mostraría dos tipos de información:

- Información abierta a todos los usuario de internet que quiere conocer aspectos de la sociedad de cazadores como puede ser: donde se encuentra, que campeonatos organiza o como se puede ser socio de la misma.

- Información restringida para los socios que accederían a ella tras haberse registrado. Mediante esta zona de información se abriría al socio un nuevo canal de comunicación con la sociedad de cazadores. El socio podría ver el estado de la cuota que ha contraído, las modalidades que va a practicar o si es correcta la información personal que la sociedad de cazadores contempla de él, etc. Además de poder discutir con otros socios de asuntos de la sociedad de cazadores mediante el foro interno.

Bibliografía

Booch, G. J. (2006). *UML. El lenguaje Unificado de Modelado.UML 2.0* (2ª ed.). Addison-Wesley.

Booch, G. (1994). *Object-Oriented Analysis and Design With Applications* (2ª ed.). Addison-Wesley Object Technology Series. Addison-Wesley, 2nd edition. .

Definición de UML. (s.f.). *www.wikipedia.org.* Obtenido de http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado

DSW, T. A. (2008/09). Practica 3. Acceso a base de datos .NET.

DSW, Técnicas Avanzadas para Desarrollo de Software. (2008/09). Practica 3. Acceso a base de datos .NET. 3º Curso Ingeniería técnica de informatica de gestión.UPV.

Extreme Programing. *www.extremeprogramming.org.* Obtenido de http://www.extremeprogramming.org

IBM Rational Rose. *www.ibm.com.* Obtenido de http://www-01.ibm.com/software/rational/

ISG, Ingeniería de software de gestión. (2008/09). Práctica 3. Diseño de persistencia. 3º Curso Ingeniería técnica de informatica de gestión.UPV.

ISG, Ingeniería del software de gestión. (2008/09). Práctica 3. Diseño de negocio. 3º Curso Ingeniería técnica de informatica de gestión.UPV.

Kruchten, P. (November 1995). *"Architectural Blueprints--The 4+1 View Model of Software Architecture".* IEEE Software, Institute of Electrical and Electronics Engineers.

Lenguaje C#. *http://msdn.microsoft.com/es-es/default.aspx.* Obtenido de http://msdn.microsoft.com/es-es/vcsharp/aa336809.aspx

Microsoft Visual Studio. *http://msdn.microsoft.com/es-es/default.aspx.* Obtenido de http://msdn.microsoft.com/es-es/vstudio/default.aspx

Rational Unified Process. (s.f.). *www.ibm.com.* Obtenido de http://www-01.ibm.com/software/awdtools/rup/

Sociedad de cazadores Ecija. (s.f.). *www.cazaecija.com.* Obtenido de http://www.cazaecija.com/

Sociedad de cazadores El Raigosu. (s.f.). *www.elraigosu.es.* Obtenido de http://www.elraigosu.es/

Sociedad de cazadores San Saturio. (s.f.). *www.sociedadsansaturio.com.* Obtenido de http://www.sociedadsansaturio.com/

Sommerville, I. (2002). *Ingeniería del Software.* England: Addison Weley.

Anexo A:

Plantillas de casos de usos

En este apartado se muestran un alto número de las plantillas de los casos de uso del proyecto que se mostraron en el apartado 3.1.

Gestión de especies cazables

Caso de uso	Alta de especie cazable.
Actores	Secretario.
Propósito	Crear una nueva especie cazable.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva especie cazable y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea incluir una nueva especie cazable.	
2. El secretario introduce los datos de identificación de la nueva especie cazable.	3. Valida los datos introducidos.
	4. Comprueba que no existe una especie cazable con el mismo nombre común introducido.
	5. Registra el alta de la nueva especie cazable.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 4 el nombre común introducido coincide con el de otra especie cazable.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de especie cazable.
Actores	Secretario.
Propósito	Dar de baja una especie cazable.
Resumen	El secretario da de baja una especie cazable.
Precondiciones	Se ha realizado el caso de uso Consulta de especies cazables mostrando las que son actuales.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja una especie cazable.	
2. El secretario selecciona la especie cazable a dar de baja.	
3. El secretario solicita dar de baja la especie cazable seleccionada.	4. Busca la especie cazable seleccionada.
	5. Comprueba que la especie cazable no ha tenido ninguna repoblación.
	6. Si no ha tenido ninguna repoblación, elimina la especie cazable del sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 5 la especie cazable ha tenido alguna repoblación.	
	1.1. Modifica el estado de la especie cazable a “HISTORIAL” y lo registra en el sistema.
	1.2. Muestra mensaje de información.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de especie cazable.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de una especie cazable.
Resumen	El secretario modifica los datos de una especie cazable por nuevos datos.
Precondiciones	Se ha realizado el caso de uso Consulta de especies cazables mostrando las que son actuales.
Poscondiciones	
Incluye	Consulta de especie cazable.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación de una especie cazable.	
2. El secretario selecciona la modalidad de caza a modificar.	
3. El secretario solicita realizar la modificación de la especie cazable.	
4. Se realiza el caso de uso Consulta de especie cazable.	
5. El secretario modifica los datos de identificación de la especie cazable.	6. Valida los datos introducidos.
	7. Comprueba que no existe una especie cazable con el mismo nombre común introducido.
	8. Registra las modificaciones de los datos de la especie cazable en el sistema.
	9. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 7 el nombre común introducido coincide con el de otra especie cazable.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Restablecimiento de especie cazable de historial.
Actores	Secretario.
Propósito	Restablecer una especie cazable que está en el historial.
Resumen	El secretario restablece una especie cazable.
Precondiciones	Se ha realizado el caso de uso Consulta de especies cazables mostrando las que están en historial.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea restablecer una especie cazable.	
2. El secretario selecciona la especie cazable a restablecer.	
3. El secretario solicita restablecer la especie cazable seleccionada.	4. Busca la especie cazable seleccionada.
	5. Modifica el estado de la especie cazable a “ACTUAL” y lo registra en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de especie cazable.
Actores	
Propósito	Visualizar una especie cazable en detalle.
Resumen	El sistema muestra toda la información de una especie cazable.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca la especie cazable seleccionada.
	2. Muestra todos los datos de identificación de la especie cazable seleccionada con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de especies cazables.
Actores	Secretario.
Propósito	Visualizar las especies cazables.
Resumen	El secretario muestra las especies cazables.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de especies cazables.	
2. El secretario introduce una o varios campos de búsqueda.	3. Muestra la relación de especies cazables encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de modalidades de caza

Caso de uso	Alta de modalidad de caza.
Actores	Secretario.
Propósito	Crear una nueva modalidad de caza.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva modalidad de caza y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea incluir una nueva modalidad de caza.	
2. El secretario introduce los datos de identificación de la nueva modalidad de caza.	3. Valida los datos introducidos.
	4. Comprueba que no existe una modalidad de caza con el mismo nombre introducido.
	5. Registra el alta de la nueva modalidad de caza en el sistema.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	2.1. Muestra mensaje de error.
2. Si en 4 el nombre introducido coincide con el de otra modalidad de caza.	
	2.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de modalidad de caza.
Actores	Secretario.
Propósito	Dar de baja una modalidad de caza.
Resumen	El secretario da de baja una modalidad de caza.
Precondiciones	Se ha realizado el caso de uso Consulta de modalidades de caza mostrando las que son actuales.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja una modalidad de caza.	
2. El secretario selecciona la modalidad de caza a dar de baja.	
3. El secretario solicita dar de baja la modalidad de caza seleccionada.	4. Busca la modalidad de caza seleccionada.
	5. Comprueba que la modalidad de caza no ha pertenecido a ninguna modalidad de temporada.
	6. Si la modalidad de caza no ha pertenecido a ninguna modalidad de temporada, elimina la modalidad de caza del sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 5 la modalidad de caza ha pertenecido alguna modalidad de temporada.	
	1.1. Modifica el estado de la modalidad de caza a “HISTORIAL” y lo registra en el sistema.
	1.2. Muestra mensaje de información.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de modalidad de caza.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de una modalidad de caza.
Resumen	El secretario modifica los datos de una modalidad de caza por nuevos datos.
Precondiciones	Se ha realizado el caso de uso Consulta de modalidad de caza mostrando las que son actuales.
Poscondiciones	
Incluye	Consulta de modalidad de caza.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación de una modalidad de caza.	
2. El secretario selecciona la modalidad de caza a modificar.	
3. El secretario solicita realizar la modificación de la modalidad de caza.	
4. Se realiza el caso de uso Consulta de modalidad de caza.	
5. El secretario modifica los datos de identificación de la modalidad de caza.	5. Valida los datos introducidos.
	6. Comprueba que no existe una modalidad de caza con el mismo nombre introducido.
	7. Registra las modificaciones de los datos de la modalidad de caza en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 7 el nombre introducido coincide con el de otra modalidad de caza.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Restablecimiento de modalidad de caza de historial.
Actores	Secretario.
Propósito	Restablecer una modalidad de caza que está en el historial.
Resumen	El secretario restablece una modalidad de caza.
Precondiciones	Se ha realizado el caso de uso Consulta de modalidades de caza mostrando las que están en historial.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea restablecer una modalidad de caza.	
2. El secretario selecciona la modalidad de caza a restablecer.	
3. El secretario solicita restablecer la modalidad de caza seleccionada.	4. Busca la modalidad de caza seleccionada.
	5. Modifica el estado de la modalidad de caza a “ACTUAL” y lo registra en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de modalidad de caza.
Actores	
Propósito	Visualizar una modalidad de caza en detalle.
Resumen	El sistema muestra toda la información de una modalidad de caza.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca la modalidad de caza seleccionada.
	2. Muestra todos los datos de identificación de la modalidad de caza seleccionada con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de modalidades de caza.
Actores	Secretario.
Propósito	Visualizar las modalidades de caza.
Resumen	El secretario muestra las modalidades de caza.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las modalidades de caza.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de modalidades de caza encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de coto de caza

Caso de uso	Alta de coto de caza.
Actores	Secretario.
Propósito	Crear el coto de caza de la sociedad.
Resumen	El secretario introduce los datos que se quieren conocer del coto de caza y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea crear el coto de caza de la sociedad.	
2. El secretario introduce los datos de identificación del coto de caza.	3. Valida los datos introducidos.
	4. Registra el alta del coto de caza en el sistema.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
Extensiones asíncronas	

Caso de uso	Modificación de datos del coto de caza.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer del coto de caza.
Resumen	El secretario modifica los datos del coto de caza por nuevos datos.
Precondiciones	
Poscondiciones	
Incluye	Consulta de coto de caza.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación del coto de caza.	
2. El secretario solicita realizar la modificación del coto de caza.	3. Muestra todos los datos de identificación del coto de caza con permiso de escritura.
4. El secretario modifica los datos de identificación de la modalidad de caza.	5. Valida los datos introducidos.
	6. Registra las modificaciones de los datos del coto de caza en el sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
Extensiones asíncronas	

Caso de uso	Consulta del coto de caza
Actores	Secretario.
Propósito	Visualizar el coto de caza.
Resumen	El secretario muestra la información del coto de caza.
Precondiciones	
Poscondiciones	
Incluye	Consulta de zonas del coto
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso uso se inicia cuando se desea realizar la consulta del coto de caza.	
2. El secretario solicita realizar la consulta del coto de caza seleccionada.	3. El sistema obtiene el coto de caza y muestra todos sus datos de identificación.
4. Se realiza el caso de uso Consulta de zonas de caza.	
Extensiones síncronas	
Extensiones asíncronas	

Caso de uso	Alta de zona del coto.
Actores	Secretario.
Propósito	Crear una nueva zona del coto.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva zona y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea incluir una nueva zona del coto.	
2. El secretario introduce los datos de identificación de la nueva zona del coto.	3. Valida los datos introducidos.
	4. Comprueba que no existe una zona del coto con el mismo nombre introducido.
	5. Registra el alta de la nueva zona del coto.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	2.1. Muestra mensaje de error.
1. Si en 4 el nombre introducido coincide con el de otra zona del coto.	
	1.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de zona del coto.
Actores	Secretario.
Propósito	Dar de baja una zona del coto.
Resumen	El secretario da de baja una zona del coto.
Precondiciones	Se ha realizado el caso de uso Consulta de zonas del coto mostrando las que son actuales.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja una zona del coto.	
2. El secretario selecciona la zona del coto.	
3. El secretario solicita dar de baja la zona de coto seleccionada.	4. Busca la zona del coto seleccionada.
	5. Comprueba que en la zona del coto se ha realizado alguna actuación de mejora.
	6. Si en la zona del coto no se ha realizado ninguna actuación de mejora, mensaje preguntando si desea eliminar la zona del coto del sistema o pasarla a historial.
7. El secretario solicita eliminar la zona de caza.	8. Elimina la zona de coto del sistema o la pasa a.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 en la zona del coto se ha realizado alguna actuación de mejora.	
	1.1. Modifica el estado de la zona del coto a “HISTORIAL” y lo registra en el sistema.
	1.2. Muestra mensaje de información.
2. Si en 7 el secretario solicita que la zona del coto pase a formar parte del hsitorial	
	2.1. Modifica el estado de la zona del coto a “HISTORIAL” y lo registra en el sistema.
	2.2. Muestra mensaje de información.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de zona del coto.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de una zona del coto.
Resumen	El secretario modifica los datos de una zona del coto por nuevos datos.
Precondiciones	Se ha realizado el caso de uso Consulta de zona del coto actual.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación de una zona del coto actual.	
2. El secretario solicita realizar la modificación de la zona del coto.	3. Muestra los datos de identificación de la zona del coto con permiso de escritura.
4. El secretario modifica los datos de identificación de la zona del coto seleccionada.	5. Valida los datos introducidos.
	6. Comprueba que no existe una zona del coto con el mismo nombre introducido.
	7. Registra las modificaciones de los datos de la zona de caza en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 5 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
2. Si en 6 existe una zona del coto con el mismo nombre.	
	2.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Restablecimiento de zona del coto de historial.
Actores	Secretario.
Propósito	Restablecer una zona del coto que está en el historial.
Resumen	El secretario restablece una zona del coto.
Precondiciones	Se ha realizado el caso de uso Consulta de zonas del coto mostrando las que están en historial.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea restablecer una zona del coto.	
2. El secretario selecciona la zona del coto a restablecer.	
3. El secretario solicita restablecer la zona del coto seleccionada.	4. Busca la zona del coto seleccionada.
	5. Modifica el estado de la zona del coto a “ACTUAL” y lo registra en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de zona del coto.
Actores	Secretario.
Propósito	Visualizar una zona del coto en detalle.
Resumen	El secretario muestra toda la información de una zona del coto.
Precondiciones	Se ha realizado el caso de uso Consulta de zonas del coto.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso uso se inicia cuando se desea realizar la consulta de una zona del coto.	
2. El secretario selecciona la zona del coto a consultar.	
3. El secretario solicita realizar la consulta de la zona del coto seleccionada.	4. Busca la zona del coto seleccionada.
	5. Muestra todos los datos de identificación de la zona del coto seleccionada.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de zonas del coto.
Actores	Secretario.
Propósito	Visualizar las zonas del coto.
Resumen	El secretario muestra las zonas de coto.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las zonas del coto.	
2. El secretario introduce una o varios campos de búsqueda.	3. Muestra la relación de zonas del coto encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de tipos de socio

Caso de uso	Alta de tipo de socio.
Actores	Secretario.
Propósito	Crear una nuevo tipo de socio.
Resumen	El secretario introduce los datos que se quieren conocer del nuevo tipo de socio y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea incluir un nuevo tipo de socio.	
2. El secretario introduce los datos de identificación del nuevo tipo de socio.	3. Valida los datos introducidos.
	4. Comprueba que no existe una tipo de socio con el mismo nombre introducido.
	5. Registra el alta del tipo de socio.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 4 el nombre introducido coincide con el de otro tipo de socio.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de tipo de socio.
Actores	Secretario.
Propósito	Dar de baja un tipo de socio.
Resumen	El secretario da de baja un tipo de socio.
Precondiciones	Se ha realizado el caso de uso Consulta de tipos de socio mostrando los actuales.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja un tipo de socio.	
2. El secretario selecciona el tipo de socio a dar de baja.	
3. El secretario solicita dar de baja el tipo de socio seleccionado.	4. Busca el tipo de socio seleccionado.
	5. Comprueba que el tipo de socio no aparece en ninguna cuota de tipo de socio.
	6. Si no ha aparecido en ninguna cuota de tipo de socio, elimina el tipo de socio del sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 5 el tipo de socio ha tenido alguna cuota de tipo de socio.	
	1.1. Modifica el estado del tipo de socio a “HISTORIAL” y lo registra en el sistema.
	1.2. Muestra mensaje de información.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de tipo de socio.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de un tipo de socio.
Resumen	El secretario modifica los datos de un tipo de socio por nuevos datos.
Precondiciones	Se ha realizado el caso de uso Consulta de tipos de socio mostrando los actuales.
Poscondiciones	
Incluye	Consulta de tipo de socio.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación de un tipo de socio.	
2. El secretario selecciona el tipo de socio a modificar.	
3. El secretario solicita realizar la modificación del tipo de socio.	
4. Se realiza el caso de uso Consulta de tipo de socio.	
5. El secretario modifica los datos de identificación del tipo de socio.	6. Valida los datos introducidos.
	7. Comprueba que no existe un tipo de socio con el mismo nombre introducido.
	8. Registra las modificaciones de los datos del tipo de socio en el sistema.
	9. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 7 el nombre introducido coincide con el de otro tipo de socio.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de tipo de socio.
Actores	
Propósito	Visualizar un tipo de socio en detalle.
Resumen	El sistema muestra toda la información de un tipo de socio.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca el tipo de socio seleccionado.
	2. Muestra todos los datos de identificación del tipo de seleccionado con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de tipos de socio.
Actores	Secretario.
Propósito	Visualizar los tipos de socio.
Resumen	El secretario muestra los tipos de socio.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de los tipos de socio.	
2. El secretario introduce una o varios campos de búsqueda.	3. Muestra la relación de tipos de socio encontrados.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de datos de temporada

Caso de uso	Alta de temporada.
Actores	Secretario.
Propósito	Crear una nueva de temporada.
Resumen	El secretario introduce los datos de identificación de la nueva temporada y la guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita crear una nueva temporada.	2. Registra el alta de la nueva temporada.
	3. Muestra un mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de temporada actual.
Actores	Secretario.
Propósito	Dar de baja la temporada actual.
Resumen	El secretario da de baja la temporada actual.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea dar de baja la temporada actual.	
2. El secretario solicita dar de baja la temporada actual.	3. Elimina toda la información referente a la temporada: datos, cuotas, cuotas de socio, modalidades de temporada, actuaciones, actividades, movimientos y asambleas.
	4. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Inicio de gestión de temporada actual.
Actores	Secretario.
Propósito	Iniciar la gestión de la temporada tras haber configurado sus cuotas y modalidades.
Resumen	El secretario solicita el inicio de la gestión de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de temporada actual.
Poscondiciones	
Incluye	Alta de cuota de socio.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita iniciar la temporada actual.	
2. El secretario solicita iniciar la temporada.	3. Modifica el estado de la temporada actual a “EN CURSO” y lo registra en el sistema.
4. Para cada socio actual con pago de cuotas al corriente.	
	4.1. Registra el alta de la nueva cuota de socio.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Finalización de gestión de temporada actual.
Actores	Secretario.
Propósito	Finalizar la gestión de la temporada.
Resumen	El secretario solicita el fin de la gestión de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de temporada actual.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita iniciar la temporada actual.	
2. El secretario solicita iniciar la temporada.	3. Modifica el estado de la temporada actual a “EN CURSO” y lo registra en el sistema.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de temporada actual.
Actores	
Propósito	Visualizar la temporada actual en detalle.
Resumen	El sistema muestra toda la información de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de temporadas.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Obtiene la temporada actual.
	2. Muestra toda la información de la temporada actual.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de temporada.
Actores	
Propósito	Visualizar una temporada en detalle.
Resumen	El sistema muestra toda la información de una temporada.
Precondiciones	Se ha realizado el caso de uso Consulta de temporadas.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	Obligaciones del sistema
	1. Busca la temporada seleccionada.
	2. Muestra toda la información de la temporada seleccionada.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	
Caso de uso	Consulta de temporadas.
Actores	Secretario.
Propósito	Visualizar las temporadas.
Resumen	El secretario muestra las temporadas.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las temporadas.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de temporadas encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de cuotas

Caso de uso	Alta de cuota.
Actores	Secretario, Tesorero.
Propósito	Crear una nueva cuota de la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva cuota y la guarda en el sistema.
Precondiciones	El estado de la temporada está “EN CONFIGURACIÓN”.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones síncronas	
Extensiones asíncronas	

Caso de uso	Alta de cuota de entrada.
Actores	Secretario, Tesorero
Propósito	Crear la nueva cuota de entrada de la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva cuota de entrada aportados por el tesorero y la guarda en el sistema.
Precondiciones	El estado de la temporada está "EN CONFIGURACIÓN" y no se ha determinado ninguna cuota.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Alta de cuota
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario que establezca una nueva cuota de sociedad para la temporada actual.	
2. El secretario solicita dar de alta la nueva cuota de entrada.	
3. El secretario introduce los datos de identificación de la nueva cuota de entrada que le proporciona el tesorero.	4. Valida los datos introducidos.
	5. Registra el alta de la nueva cuota de sociedad en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Alta de cuota de sociedad.
Actores	Secretario, Tesorero
Propósito	Crear una nueva cuota de sociedad de la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva cuota de sociedad aportados por el tesorero y la guarda en el sistema.
Precondiciones	El estado de la temporada está "EN CONFIGURACIÓN" y se ha determinado la cuota de entrada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Alta de cuota
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario que establezca una nueva cuota de sociedad para la temporada actual.	
2. El secretario solicita dar de alta una nueva cuota de sociedad.	
3. El secretario introduce los datos de identificación de la nueva cuota de sociedad que le proporciona el tesorero.	4. Valida los datos introducidos.
	5. Comprueba que no existe una cuota con el mismo nombre.
	6. Registra el alta de la nueva cuota de sociedad en el sistema.
	7. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
2. Si en 5 el nombre de la cuota de sociedad introducida coincide con el de otra cuota.	
	2.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Alta de cuota de modalidad.
Actores	Secretario, Tesorero
Propósito	Crear una nueva cuota de modalidad de la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva cuota de modalidad aportados por el tesorero y la guarda en el sistema.
Precondiciones	El estado de la temporada está "EN CONFIGURACIÓN" y se ha determinado la cuota de entrada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Alta de cuota
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando se da de alta una modalidad de temporada.	
2. El secretario introduce los datos de identificación de la nueva cuota de modalidad que le proporciona el tesorero.	3. Valida los datos introducidos.
	4. Comprueba que no existe una cuota con el mismo nombre.
	5. Registra el alta de la nueva cuota de modalidad en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra mensaje de error.
2. Si en 4 el nombre de la cuota de sociedad introducida coincide con el de otra cuota.	
	2.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de cuota de sociedad.
Actores	Secretario, Tesorero
Propósito	Dar de baja una cuota de sociedad establecida para la temporada actual.
Resumen	El secretario elimina la cuota de sociedad que el tesorero le indica.
Precondiciones	El estado de la temporada actual es “EN CONFIGURACIÓN” y se ha realizado el caso de uso Consulta de cuotas de temporada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario dar de baja una cuota de sociedad.	
2. El secretario selecciona la cuota de sociedad que el tesorero le indica.	
3. El secretario solicita dar de baja la cuota de sociedad seleccionada.	4. Busca la cuota de sociedad seleccionada.
	5. Elimina la cuota de sociedad del sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de cuota de modalidad.
Actores	
Propósito	Dar de baja una cuota de modalidad establecida para la temporada actual.
Resumen	El sistema elimina la cuota de modalidad.
Precondiciones	El estado de la temporada actual es “EN CONFIGURACIÓN” y se ha solicitado dar de baja la modalidad de la temporada actual a la que pertenece.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca la cuota de modalidad de la modalidad de la temporada actual seleccionada para ser dada de baja.
	2. Elimina la cuota de modalidad del sistema.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de cuota.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de una cuota de la temporada actual.
Resumen	El secretario modifica los datos de una cuota de temporada por los nuevos datos aportados por el tesorero.
Precondiciones	El estado de la temporada actual es “EN CONFIGURACIÓN” y se ha realizado el caso de uso Consulta de cuotas de temporada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de cuota de entrada.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de la cuota de entrada de la temporada actual.
Resumen	El secretario modifica los datos de la cuota de entrada por los nuevos datos aportados por el tesorero.
Precondiciones	El estado de la temporada actual es “EN CONFIGURACIÓN” y se ha realizado el caso de uso Consulta de cuotas de temporada.
Poscondiciones	
Incluye	Consulta de cuota de entrada.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario modificar los datos de la cuota de entrada.	
2. El secretario selecciona la cuota de entrada.	
3. El secretario solicita modificar la cuota de entrada.	
4. Se realiza el caso de uso Consulta de cuota de entrada.	
5. El secretario modifica los datos de identificación de la cuota de entrada.	6. Valida los datos introducidos.
	7. Registra las modificaciones de los datos de la cuota de entrada en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de cuota de sociedad.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de una cuota de sociedad de la temporada actual.
Resumen	El secretario modifica los datos de una cuota de sociedad por los nuevos datos aportados por el tesorero.
Precondiciones	El estado de la temporada actual es "EN CONFIGURACIÓN" y se ha realizado el caso de uso Consulta de cuotas de temporada.
Poscondiciones	
Incluye	Consulta de cuota de sociedad.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario modificar los datos de una cuota de sociedad.	
2. El secretario selecciona la cuota de sociedad que el tesorero le indica.	
3. El secretario solicita modificar la cuota de sociedad seleccionada.	
4. Se realiza el caso de uso Consulta de cuota de sociedad.	
5. El secretario modifica los datos de identificación de la cuota de sociedad.	6. Valida los datos introducidos.
	7. Comprueba que no existe una cuota con el mismo nombre introducido.
	8. Registra las modificaciones de los datos de la cuota de sociedad en el sistema.
	9. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
2. Si en 7 el nombre introducido coincide con el de otra cuota.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de cuota de modalidad.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de una cuota de modalidad de la temporada actual.
Resumen	El secretario modifica los datos de una cuota de sociedad por los nuevos datos aportados por el tesorero.
Precondiciones	El estado de la temporada actual es "EN CONFIGURACIÓN" y se ha realizado el caso de uso Consulta de cuotas de temporada o el caso se uso Modificación de datos de modalidad de temporada.
Poscondiciones	
Incluye	
Extiende	Consulta de cuota de modalidad.
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario modificar los datos de la cuota de modalidad o se ha seleccionado modificar la modalidad de temporada a la que pertenece dicha cuota.	
2. El secretario selecciona la cuota de modalidad que el tesorero le indica.	
3. El secretario solicita modificar la cuota de modalidad seleccionada.	
4. Se realiza el caso de uso Consulta de cuota de modalidad.	
5. El secretario modifica los datos de identificación de la cuota de modalidad.	6. Valida los datos introducidos.
	7. Registra las modificaciones de los datos de la cuota de modalidad en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 2 se ha solicitado modificar la modalidad de temporada a la que pertenece la cuota.	
	1.1. Saltar a la instrucción 5, realizando 6 y 7.
2. Si en 6 los datos introducidos son incorrectos.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de cuota.
Actores	
Propósito	Visualizar una cuota en detalle.
Resumen	El sistema muestra toda la información de una cuota.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de cuota de entrada.
Actores	
Propósito	Visualizar la cuota de entrada en detalle.
Resumen	El sistema muestra toda la información de la cuota de entrada.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Obtiene la cuota de entrada.
	2. Muestra todos los datos de identificación de la cuota de entrada con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de cuota de sociedad.
Actores	
Propósito	Visualizar una cuota de sociedad en detalle.
Resumen	El sistema muestra toda la información de una cuota de sociedad.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca la cuota de sociedad seleccionada.
	2. Muestra todos los datos de identificación de la cuota de sociedad seleccionada con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de cuota de modalidad.
Actores	
Propósito	Visualizar una cuota de modalidad en detalle.
Resumen	El sistema muestra toda la información de una cuota de modalidad.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca la cuota de sociedad seleccionada.
	2. Muestra todos los datos de identificación de la cuota de modalidad seleccionada con permiso de escritura.
Extensiones sincronas	
1. Si se realizó el caso de uso Consulta de modalidad de temporada a la que pertenece a la cuota.	
	1.1. Muestra todos los datos de identificación de la cuota de modalidad seleccionada sin permiso de escritura.
Extensiones asíncronas	

Caso de uso	Consulta de cuotas de temporada.
Actores	Secretario.
Propósito	Visualizar las cuotas de establecidas para una temporada.
Resumen	El secretario muestra información de todas las cuotas establecidas en una temporada.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las cuotas de temporada.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de cuotas de temporada encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de modalidades de temporada

Caso de uso	Alta de modalidad de temporada.
Actores	Secretario, Tesorero.
Propósito	Crear una nueva modalidad de temporada actual.
Resumen	El secretario con ayuda del tesorero crea una nueva modalidad de la temporada actual.
Precondiciones	
Poscondiciones	
Incluye	Consulta de modalidades de caza.
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita crear una nueva modalidad temporada.	
2. Se realiza el caso uso Consulta de modalidades de caza mostrando las modalidades de caza actuales.	
3. El secretario selecciona las modalidades de caza que componen la modalidad de temporada.	4. Comprueba que no existe otra modalidad de la temporada actual con el mismo nombre.
	5. Comprueba que no existe otra modalidad de la temporada actual con las mismas modalidades de caza.
6. Si no existe otra modalidad de temporada con las mismas modalidades de caza se realiza el caso uso Alta de cuota de modalidad.	
	7. Registra el alta de la nueva modalidad de temporada en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 el nombre introducido coincide con el de otra modalidad de la temporada actual.	
	1.1. Muestra mensaje de error.
2. Si en 5 existe otra modalidad de temporada con el mismo conjunto de modalidades de caza.	
	2.1. Muestra mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de modalidad de temporada.
Actores	Secretario.
Propósito	Dar de baja una modalidad de la temporada actual.
Resumen	El secretario elimina una de las modalidades de la temporada actual.
Precondiciones	El estado de la temporada actual es “EN CONFIGURACIÓN” y se ha realizado el caso de uso Consulta de modalidades de temporada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario quiere dar de baja una modalidad de la temporada actual.	
2. El secretario selecciona la modalidad de temporada que quiere dar de baja.	
3. El secretario solicita dar de baja la modalidad de temporada seleccionada.	4. Busca la modalidad de temporada seleccionada.
Por cada modalidad de caza que pertenece a la modalidad de temporada.	
	5. Elimina la relación de la modalidad de caza con la modalidad de temporada del sistema.
6. Se realiza el caso de uso Baja de cuota de modalidad correspondiente	
	7. Elimina la modalidad de temporada del sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de modalidad de temporada.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de una modalidad de la temporada actual.
Resumen	El secretario modifica los datos de una modalidad de temporada por nuevos datos.
Precondiciones	El estado de la temporada actual es "EN CONFIGURACIÓN" y se ha realizado el caso de uso Consulta de modalidad de temporada.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario desea modificar los datos de identificación de una modalidad de temporada.	
2. El secretario solicita realizar la modificación de la modalidad de temporada.	3. Establece la información de la modalidad de temporada con permiso de escritura.
5. El secretario modifica los datos de identificación de la modalidad de temporada.	6. Comprueba que no existe otra modalidad de la temporada actual con el mismo nombre.
	7. Comprueba que no existe otra modalidad de la temporada actual con las mismas modalidades de caza.
8. Se realiza el caso de uso Modificación de datos de cuota de modalidad.	
	9. Registra la modificación en el nombre de la modalidad de temporada así como las relaciones entre ella y sus modalidades de caza en el sistema.
	10. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 el nombre de la modalidad de temporada coincide con el de otra modalidad de la temporada actual.	
	1.1. Muestra un mensaje de error.
2. Si en 7 existe otra modalidad de temporada con el mismo conjunto de modalidades de caza.	
	2.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de modalidad de temporada.
Actores	Secretario.
Propósito	Visualizar una modalidad de temporada en detalle.
Resumen	El secretario muestra toda la información de una modalidad de temporada.
Precondiciones	
Poscondiciones	
Incluye	Consulta de cuota de modalidad
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario quiere ver en detalle una modalidad de temporada.	
2. El secretario selecciona la modalidad de temporada que quiere ver.	
3. El secretario solicita consultar la modalidad de temporada seleccionada.	4. Busca la modalidad de temporada seleccionada.
	5. Muestra todo los datos de identificación de la modalidad de temporada seleccionada.
6. Realiza el caso de uso Consulta de cuota de modalidad.	
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de modalidades de temporada
Actores	Secretario.
Propósito	Visualizar las modalidades de la temporada seleccionada.
Resumen	El secretario muestra todas las modalidades de la temporada seleccionada.
Precondiciones	.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las modalidades de temporada.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de modalidades de temporada encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión contable

Caso de uso	Alta de movimiento.
Actores	Secretario, Tesorero.
Propósito	Crear un nuevo movimiento contable en la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer del nuevo movimiento contable y lo guarda en el sistema.
Precondiciones	El estado de la temporada actual es “EN CURSO”.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Alta de ingreso.
Actores	Secretario, Tesorero.
Propósito	Crear un nuevo ingreso en la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer del nuevo ingreso y lo guarda en el sistema.
Precondiciones	El estado de la temporada actual es “EN CURSO”.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Alta de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario añadir un nuevo ingreso a la temporada actual.	
2. El secretario solicita añadir un nuevo ingreso.	
3. El secretario introduce los datos de identificación del nuevo ingreso.	4. Valida los datos introducidos.
	5. Registra el alta del nuevo ingreso en el sistema.
	6. Muestra un mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 los datos introducidos son incorrectos se	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Alta de gasto.
Actores	Secretario, Tesorero.
Propósito	Crear un nuevo gasto en la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer del nuevo gasto y lo guarda en el sistema.
Precondiciones	El estado de la temporada actual es “EN CURSO”.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Alta de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario añadir un nuevo gasto a la temporada actual.	
2. El secretario solicita añadir un nuevo gasto.	
3. El secretario introduce los datos de identificación del nuevo gasto.	4. Valida los datos introducidos.
	5. Registra el alta del nuevo gasto en el sistema.
	6. Muestra un mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 los datos introducidos son incorrectos se	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de movimiento.
Actores	Secretario, Tesorero.
Propósito	Dar de baja un movimiento de la temporada actual.
Resumen	El secretario elimina un movimiento de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de movimientos.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de ingreso.
Actores	Secretario, Tesorero.
Propósito	Dar de baja un ingreso de la temporada actual.
Resumen	El secretario elimina un ingreso de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de movimientos.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Baja de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario dar de baja a un ingreso.	
2. El secretario selecciona el ingreso que el tesorero le indica.	
3. El secretario solicita dar de baja el ingreso seleccionado.	4. Busca el ingreso seleccionado.
	5. Elimina el ingreso del sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de gasto.
Actores	Secretario, Tesorero.
Propósito	Dar de baja un gasto de la temporada actual.
Resumen	El secretario elimina un gasto de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de movimientos.
Poscondiciones	
Incluye	
Extiende	
Hereda de	Baja de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario dar de baja a un gasto.	
2. El secretario selecciona el gasto que el tesorero le indica.	
3. El secretario solicita dar de baja el gasto seleccionado.	4. Busca el gasto seleccionado.
	5. Elimina el gasto del sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	
Caso de uso	Modificación de datos de movimiento.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de un movimiento contable de la temporada actual.
Resumen	El secretario modifica los datos de un movimiento contable por los nuevos datos que le da el tesorero.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de ingreso.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de un ingreso de la temporada actual.
Resumen	El secretario modifica los datos de un ingreso por los nuevos datos que le da el tesorero.
Precondiciones	Se ha realizado el caso de uso Consulta de movimientos.
Poscondiciones	
Incluye	Consulta de ingreso.
Extiende	
Hereda de	Modificación de datos de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario modificar los datos de un ingreso.	
2. El secretario selecciona el ingreso que el tesorero le indica.	
3. El secretario solicita modificar el ingreso seleccionado.	
4. Se realiza el caso de uso Consulta de ingreso.	
5. El secretario modifica los datos de identificación del ingreso seleccionado.	6. Valida los datos introducidos.
	7. Registra las modificaciones de los datos del ingreso en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de gasto.
Actores	Secretario, Tesorero.
Propósito	Cambiar los datos que se quieren conocer de un gasto de la temporada actual.
Resumen	El secretario modifica los datos de un gasto por los nuevos datos que le da el tesorero.
Precondiciones	Se ha realizado el caso de uso Consulta de movimientos.
Poscondiciones	
Incluye	Consulta de gasto.
Extiende	
Hereda de	Modificación de datos de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el tesorero solicita al secretario modificar los datos de un gasto.	
2. El secretario selecciona el gasto que el tesorero le indica.	
3. El secretario solicita modificar el gasto seleccionado.	
4. Se realiza el caso de uso Consulta de gasto.	
5. El secretario modifica los datos de identificación del gasto seleccionado.	6. Valida los datos introducidos.
	7. Registra las modificaciones de los datos del gasto en el sistema.
	8. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 6 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de movimiento.
Actores	
Propósito	Visualizar un movimiento contable de la temporada actual en detalle.
Resumen	El sistema muestra toda la información de un movimiento contable de la temporada actual.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de ingreso.
Actores	
Propósito	Visualizar un ingreso de la temporada actual contable en detalle.
Resumen	El sistema muestra toda la información de un ingreso de la temporada actual.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	Consulta de movimiento.
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca el ingreso seleccionado.
	2. Muestra todos los datos de identificación del ingreso seleccionado con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de gasto.
Actores	
Propósito	Visualizar un gasto de la temporada actual en detalle.
Resumen	El sistema muestra toda la información de un gasto de la temporada actual.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
	1. Busca el gasto seleccionado.
	2. Muestra todos los datos de identificación del gasto seleccionado con permiso de escritura.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de movimientos
Actores	Secretario.
Propósito	Visualizar los movimientos contables de una temporada.
Resumen	El secretario muestra todos los movimientos contables de una temporada.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de los movimientos de una temporada.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de movimientos encontrados.
	4. Calcula y muestra el balance de los movimientos mostrados.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Gestión de veda

Caso de uso	Alta de veda.
Actores	Secretario, Tesorero.
Propósito	Dar de alta una nueva veda en la temporada actual.
Resumen	El secretario introduce los datos que se quieren conocer de la nueva veda y lo guarda en el sistema.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita añadir una nueva veda a la temporada actual.	
2. El secretario introduce los datos de identificación de la nueva veda.	3. Valida los datos introducidos.
	4. Registra el alta de la nueva veda en el sistema.
	5. Muestra un mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 3 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Baja de veda.
Actores	Secretario, Tesorero.
Propósito	Dar de baja una veda de la temporada actual.
Resumen	El secretario elimina una veda de la temporada actual.
Precondiciones	Se ha realizado el caso de uso Consulta de vedas.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario quiere dar de baja una veda de la temporada actual.	
2. El secretario selecciona la veda a dar de baja.	
3. El secretario solicita eliminar la veda seleccionada.	4. Busca la veda seleccionada.
	5. Elimina la veda del sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Modificación de datos de una veda.
Actores	Secretario.
Propósito	Cambiar los datos que se quieren conocer de una veda de la temporada actual.
Resumen	El secretario modifica los datos de una veda por nuevos datos.
Precondiciones	Se ha realizado el caso de uso Consulta de veda.
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita modificar los datos de una veda.	2. Muestra los datos de identificación de la veda con permiso de escritura excepto para el nombre.
3. El secretario modifica los datos de identificación de la veda.	4. Valida los datos introducidos.
	5. Registra las modificaciones de los datos de la veda en el sistema.
	6. Muestra mensaje informando que la operación ha sido satisfactoria.
Extensiones síncronas	
1. Si en 4 los datos introducidos son incorrectos.	
	1.1. Muestra un mensaje de error.
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de vedas
Actores	Secretario.
Propósito	Visualizar las vedas de una temporada.
Resumen	El secretario muestra todas las vedas de una temporada.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario solicita hacer la consulta de las vedas de una temporada.	
2. El secretario introduce uno o varios campos de búsqueda.	3. Muestra la relación de vedas encontradas.
Extensiones síncronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Caso de uso	Consulta de veda
Actores	Secretario.
Propósito	Visualizar una veda en detalle.
Resumen	El secretario muestra todos los datos de especificación de una veda.
Precondiciones	
Poscondiciones	
Incluye	
Extiende	
Hereda de	
Intenciones de usuario	**Obligaciones del sistema**
1. El caso de uso se inicia cuando el secretario quiere ver en detalle una veda.	
2. El secretario selecciona la veda que se quiere ver.	
3. El secretario solicita consultar la veda seleccionada.	4. Busca la veda seleccionada.
	5. Muestra todo los datos de identificación de la veda seleccionada.
Extensiones sincronas	
Ninguna.	
Extensiones asíncronas	
Ninguna.	

Anexo B:

Manual de instalación

El proceso de instalación de la aplicación es muy sencillo. Si en el ordenador donde se está realizando la operación no están instalados los requisitos mínimos indispensables, el instalador de la aplicación te lo ira indicando y te dirá lo que debes hacer.

Los requisitos previos necesarios que deberán estar instalados para que funcione la aplicación son los siguientes. De cada uno de ellos se muestra la página web de Microsoft donde se puede descargar:

- Windows Installer 3.1.
  http://www.microsoft.com/downloads/details.aspx?displaylang=es&familyid=889482fc-5f56-4a38-b838-de776fd4138c

- .NET framework 3.5.

http://www.microsoft.com/downloads/details.aspx?familyid=333325fd-ae52-4e35-b531-508d977d32a6&displaylang=es

- SQL Server 2005 Express.
  http://www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyID=220549b5-0b07-4448-8848-dcc397514b41

Una vez realizada la instalación de los requisitos previos necesarios a continuación se muestra los pasos a seguir para la correcta instalación de la aplicación.

1. Dentro de la carpeta de la aplicación situarse en ./Septup1/Debug y hacer clic sobre Septup para instalarla.

2. Dentro de la carpeta de la aplicación copiar los archivos de la base de datos: BD_SC y BD_SC_log y pegar en Disco Local (C:).

3. Comprobar si es correcta la configuración de superficie de SQL Server para ello:

   3.1. Ir a Inicio>Microsoft SQL Server 2005> Herramienta de configuración> Configuración de superficie de SQL Server 2005.

   3.2. Seleccionar la opción "Configuración de superficie para servicios y conexiones".

   3.3. En la parte izquierda en las opciones de SQL Server que aparecen son "Servicio" y "Conexiones remotas".

   3.4. Selecciona en "Conexiones remotas". En la parte derecha hacer clic en “Conexiones locales y remotas” y "Usar TCP/IP y canalizaciones con nombre".

   3.5. Seleccionar "Servicio". En la parte de la derecha poner tipo de inicio "Automático". Si el estado del servicio está “Detenido” hacer clic en iniciar. El estado de servicio será “En ejecución” que es el necesario para tener acceso a la base de datos de la aplicación.

   3.6. Después de establecer la configuración hacer clic en el botón aplicar.

Tras realizar los pasos indicados ya se podrá ejecutar la aplicación con éxito.

Anexo C:

Manual de usuario

En el presente anexo se especifica el manual de usuario. Mediante el cual se explicara el funcionamiento del programa para un buen manejo del usuario final.

Para comprender mejor cómo funciona la aplicación a continuación se detalla la correspondencia entre las operaciones mediante las que se gestiona cada uno de los conceptos y los botones a los que se asocian.

- Dar de alta una instancia → Botón Nuevo
- Modificar un instancia → Botón Editar
- Eliminar una instancia → Botón Eliminar
- Restablecer una instancia de historial → Botón Restablecer/Readmitir
- Guardar una instancia → Botón Guardar
- Ver una instancia → Botón Ver
- Anular una operación que no se desea realizar → Botón Cancelar

Una vez hecho este inciso ya se pude empezar a ver el funcionamiento del programa, para ello se explicará en cada una de las gestiones que soporta el sistema alguna de las operaciones antes expuestas.

C.1. Gestión de ventana principal

Cuando se ejecuta la aplicación la primera ventana que se muestra es la ventana principal, figura 45, la cual se encarga de la administración de la información referente a la sociedad de cazadores (panel de la derecha) así como del acceso al resto de gestiones que realiza el sistema (panel de la izquierda):

- Aspectos de administración.
- Socios.
- Temporadas.
- Juntas directivas.

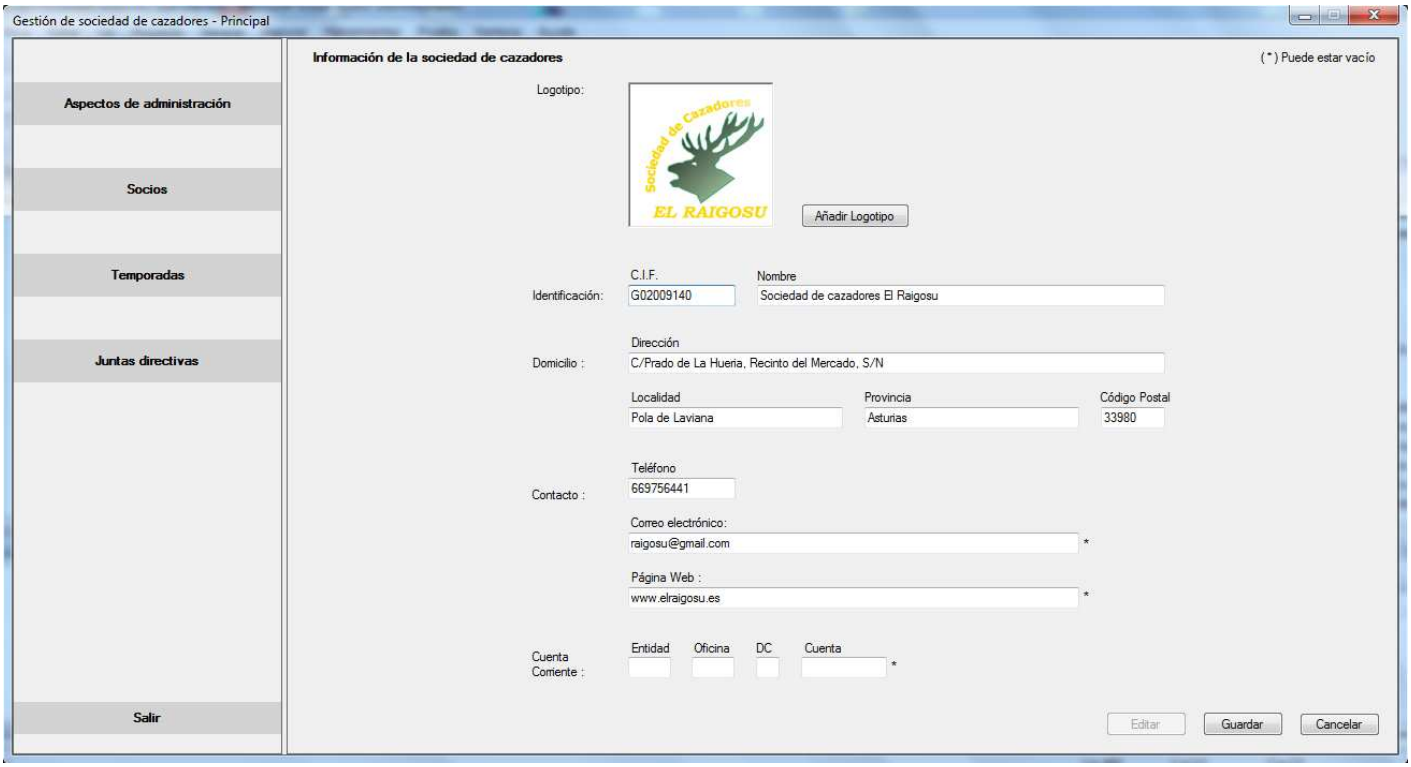

Figura 45. Formulario principal.

C.1.1. Gestión de sociedad de cazadores

Alta de sociedad de cazadores

La primera vez que se accede a la aplicación lo único que permite es realizar la operación alta de sociedad de cazadores para ello se realizan los siguientes pasos:

1. Introducir los datos solicitados
2. Hacer clic sobre el botón *Guardar*.

3. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello y se establece los datos con solo permiso de lectura. Si en cambio los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.

Modificación de información de sociedad de cazadores

Para modificar la información de la sociedad de cazadores, captura que se muestra en la figura 45. Los pasos a seguir son:

1. Hacer clic sobre el botón *Editar* (deshabilitado en la figura 45).
2. El panel de la derecha se establece con permiso de escritura.
3. Modificar los datos que se desea.
4. Hacer clic sobre el botón *Guardar.*
5. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello y se establece los datos con permiso de lectura.
   Si en cambio los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.

C.2. Gestión de aspectos de administración

Para acceder a la gestión de aspectos de administración en la ventana principal hacer clic sobre el botón *Aspectos de administración*. Aparecerá la ventana Aspectos de administración, figura 46. En dicha ventana se administran aspectos de diferente índole que son necesarios para poder realizar la gestión las temporadas y de los socios, por lo que es recomendable su gestión en primer lugar. Dichos aspectos son los siguientes:

- Coto de caza.
- Tipos de socio.
- Modalidades de caza.
- Especies cazables.
- Comisiones predefinidas.

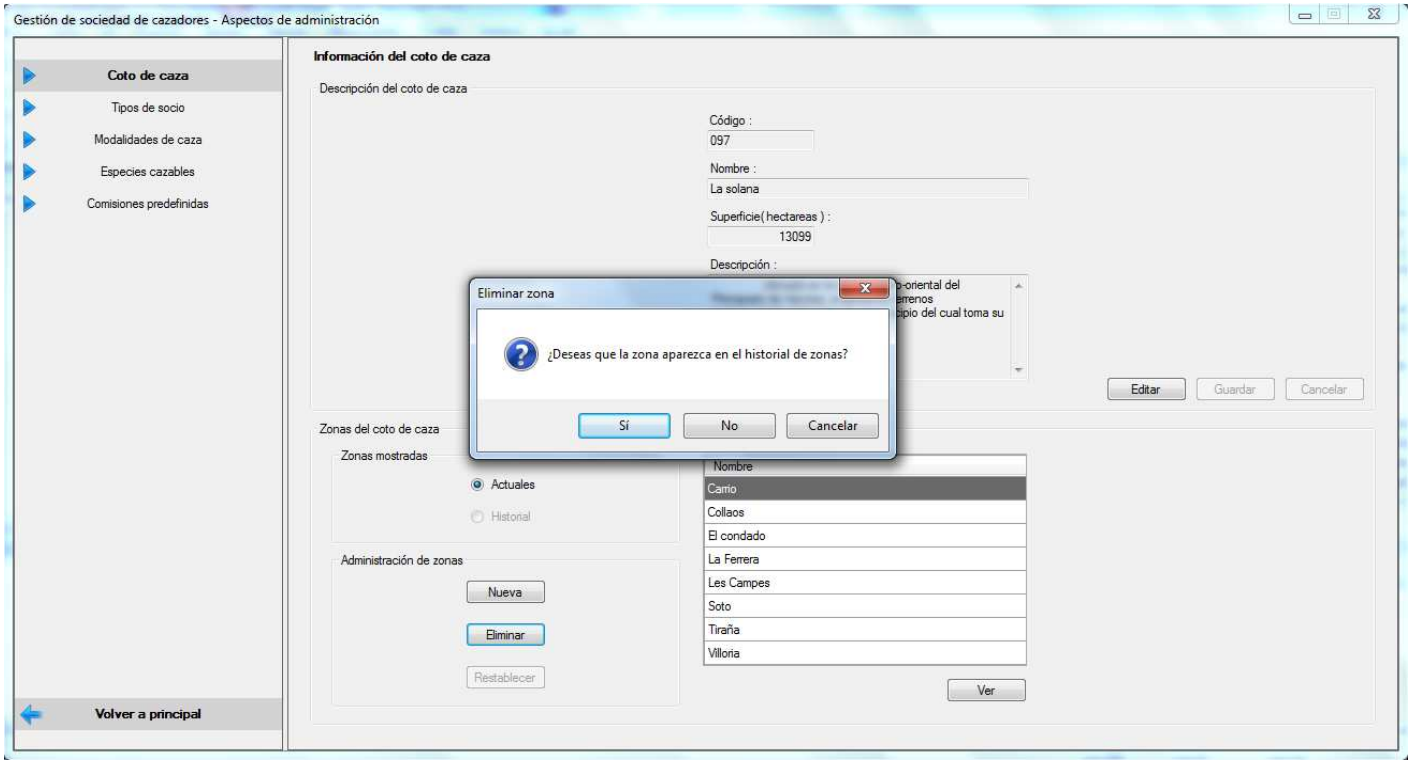

Figura 46. Formulario Aspectos de Administración: Coto de caza.

C.2.1. Gestión del coto de caza

Para acceder a la gestión del coto de caza en la ventana Aspectos de administración hacer clic sobre el botón *Coto de caza* del panel de la izquierda. El aspecto que tiene la ventana Aspectos de administración es el de la figura 46 en la que se muestra el coto de caza. Si el coto de caza no ha sido dado de alta se muestra una ventana donde se debe confirmar dicha alta.

En el panel de la derecha se muestra la información del coto de caza se divide en dos grupos:

- *Descripción del coto de caza*, donde se muestran los datos de especificación del coto de caza.
- *Zonas del coto de caza*, donde se gestionan las zonas en que se divide.

Eliminar zona

La figura 46 muestra la captura de pantalla cuando se está realizando operación de eliminar una zona del coto de caza. Para realizar dicha operación hay que seguir los siguientes pasos.

1. En el grupo *Zonas mostradas* hacer clic sobre el botón *Actuales* para ver las zonas actuales que aparecen en la lista.
2. Seleccionar la zona que se desea eliminar en la lista, en la figura 46 la zona con nombre “Carrio”.
3. Hacer clic sobre el botón *Eliminar* del grupo *Administración de zonas.*
4. Se muestra un mensaje preguntando si se quiere que la zona aparezca en el historial o se elimine del sistema.
5. Hacer clic sobre el botón que se crea conveniente.

Ver zona

Para realizar la operación de ver una zona en detalle habrá dos formas de hacerlo:

1. Seleccionar la zona en la lista de zonas y hacer clic sobre el botón *Ver*.
2. Hacer doble clic sobre la zona.

El aspecto de la ventana Aspectos de administración será el siguiente, figura 47.

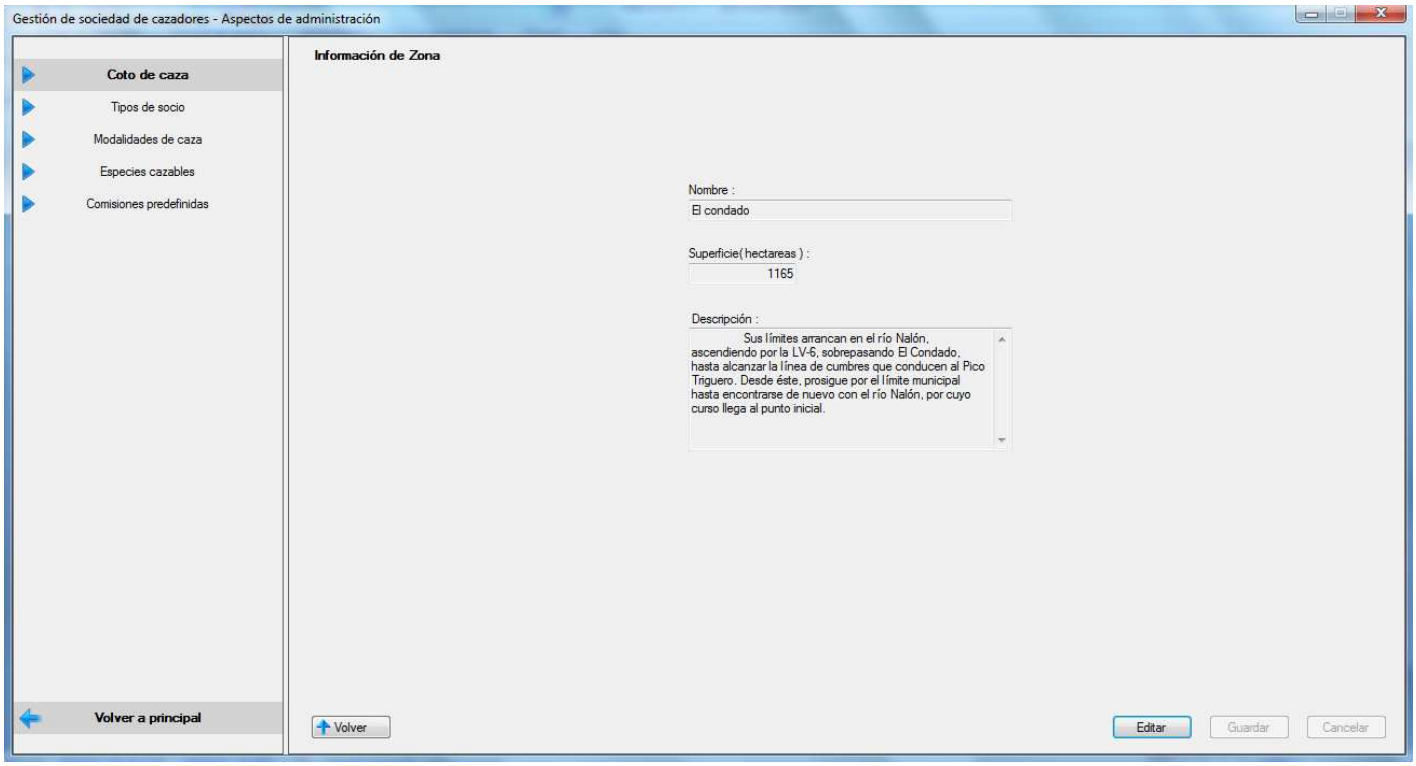

Figura 47. Formulario Aspectos de administración: Zona de caza.

C.2.2. Gestión de tipos de socio

Para acceder a la gestión de tipos de socio en la ventana Aspectos de administración hacer clic sobre el botón *Tipos de socio* del panel de la izquierda. El aspecto que tiene la ventana Aspectos de administración es el de la figura 48 en la que se gestionan los tipos de socios que en que se dividen los socios.

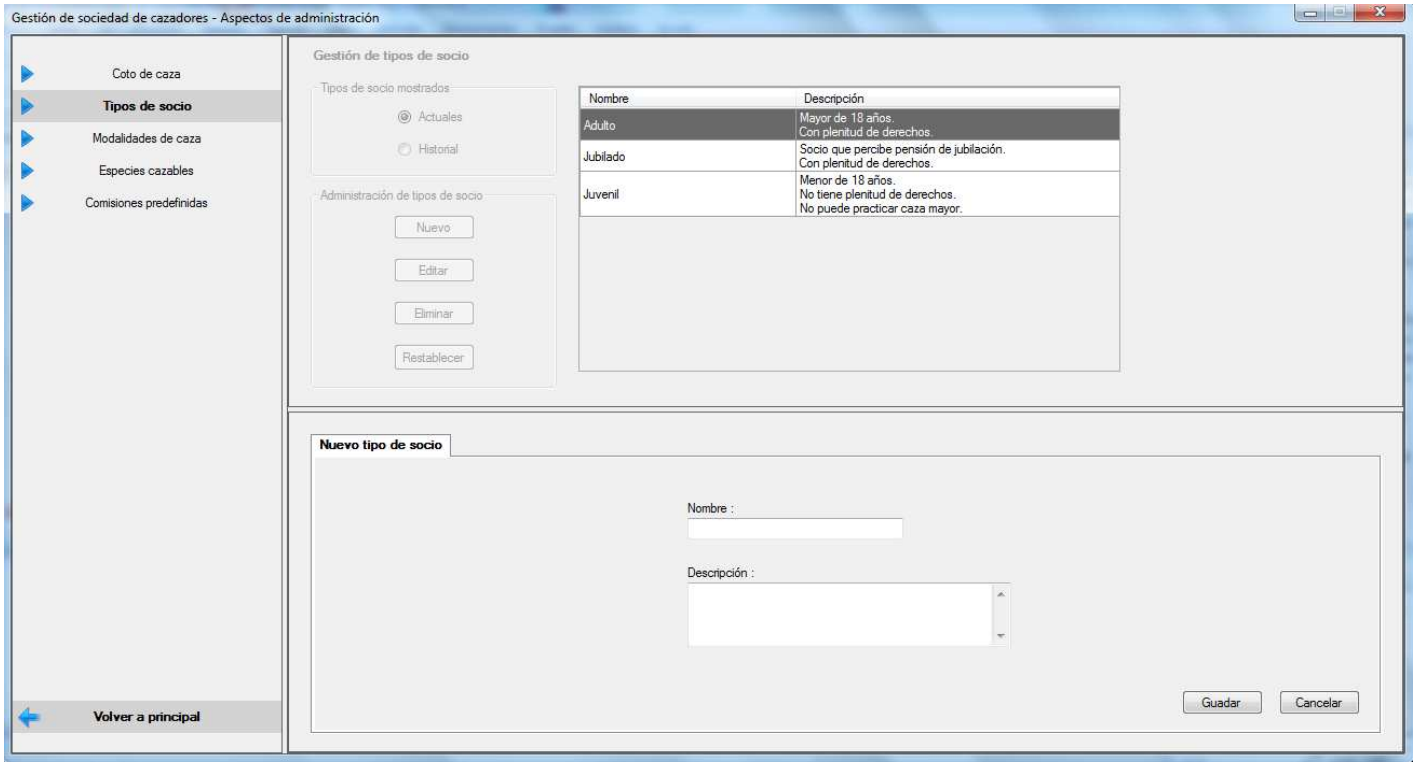

Figura 48. Formulario Aspectos de administración: Tipos de socios

Alta de tipo de socio

La figura 48 muestra la captura de pantalla cuando se está realizando la operación alta de tipo de socio. Para realizar dicha operación hay que seguir los siguientes pasos.

1. En el grupo *Tipos de socios mostrados* hacer clic sobre el botón *Actuales* (deshabilitado en figura 48).
2. En el del grupo *de Administración de tipos de socio* hacer clic sobre el botón *Nuevo* (deshabilitado en la figura 48).
3. Tras habilitarse el panel de abajo se introducen el nombre y descripción del nuevo tipo de socio.
4. Hacer clic sobre botón *Guardar*.
5. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello, se habilita el panel de arriba apareciendo el nuevo tipo de socio en la lista.

Si existe otro tipo de socio con el mismo nombre se muestra un mensaje de error informando de ello y se cancela la operación.

C.2.3. Gestión de modalidades de caza

Para acceder a la gestión de modalidades de caza en la ventana Aspectos de administración hacer clic sobre el botón *Modalidades de caza* del panel de la izquierda. El aspecto que tiene la ventana Aspectos de administración es el de la figura 49 en la que se gestionan las modalidades de caza que los socios pueden practicar.

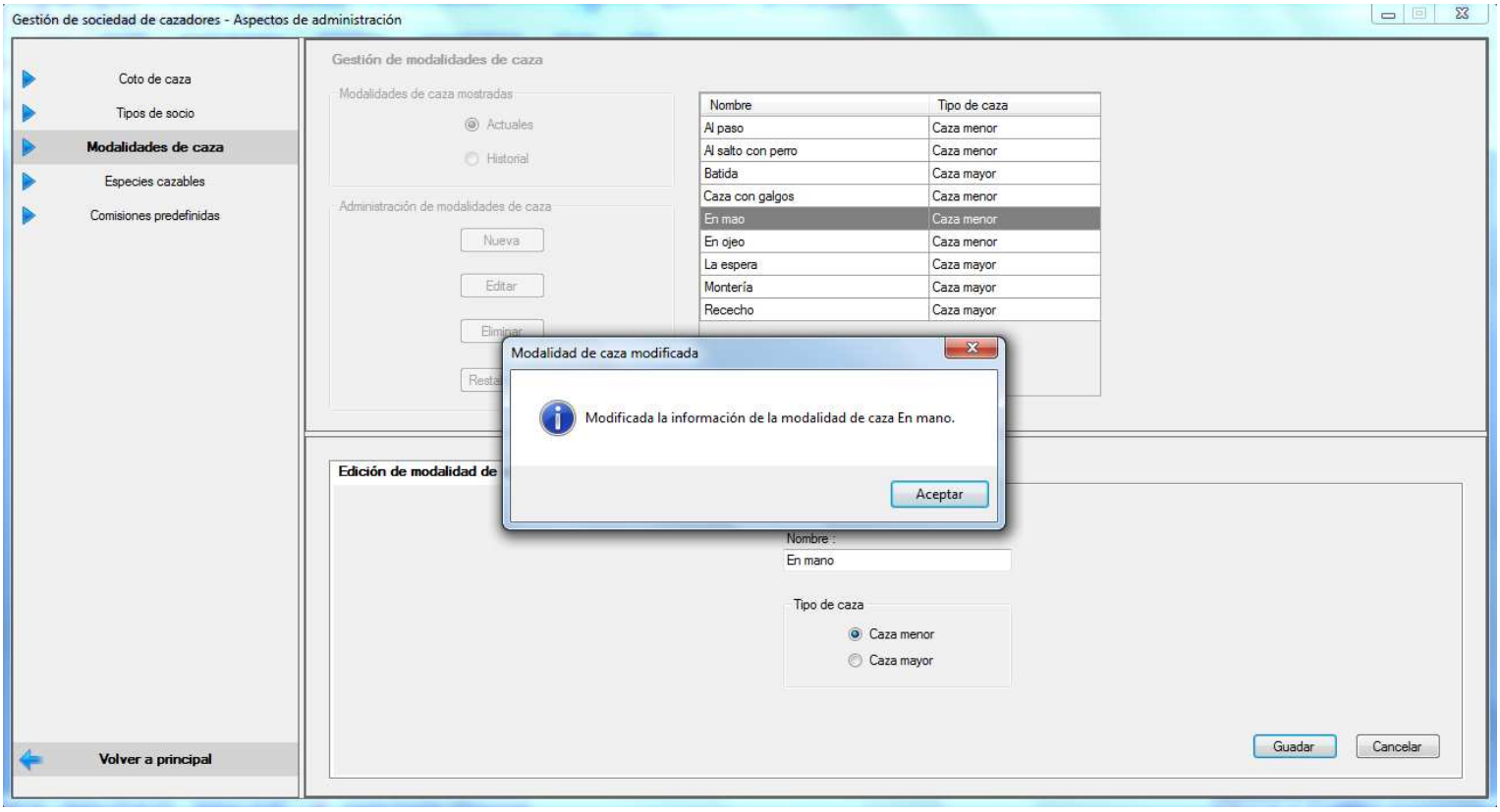

Figura 49. Formulario Aspectos de administración: Modalidades de caza.

Modificación de modalidad de caza

La figura 49 muestra la captura de pantalla cuando se está realizando la operación de modificación de modalidad de caza. Para realizar dicha operación hay que seguir los siguientes pasos:

1. En el grupo *Modalidades de caza mostradas* hacer clic sobre el botón *Actuales*.
2. Seleccionar la modalidad de caza a modificar en la lista, en la figura 49 “En mao”.

3. En el grupo *Administración de modalidades de caza* hacer clic sobre el botón *Editar* (deshabilitado en la figura 49).
4. Tras habilitarse el panel de abajo se modifican los datos que se desea, en la figura 49 se modifica el nombre pasando a ser “En mano”.
5. Hacer clic sobre botón *Guardar*.
6. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello, se habilita el panel de arriba apareciendo los nuevos datos de la modalidad de caza en la lista.
   Si se ha modificado el nombre y coincide con el de otra modalidad de caza se muestra un mensaje de error informando de ello y se cancela la operación.

C.2.4. Gestión de especies cazables

Para acceder a la gestión de especies cazables en la ventana Aspectos de administración hacer clic sobre el botón *Especies cazables* del panel de la izquierda. El aspecto que tiene la ventana Aspectos de administración es el de la figura 50 en la que se gestionan las especies cazables que cazan los socios.

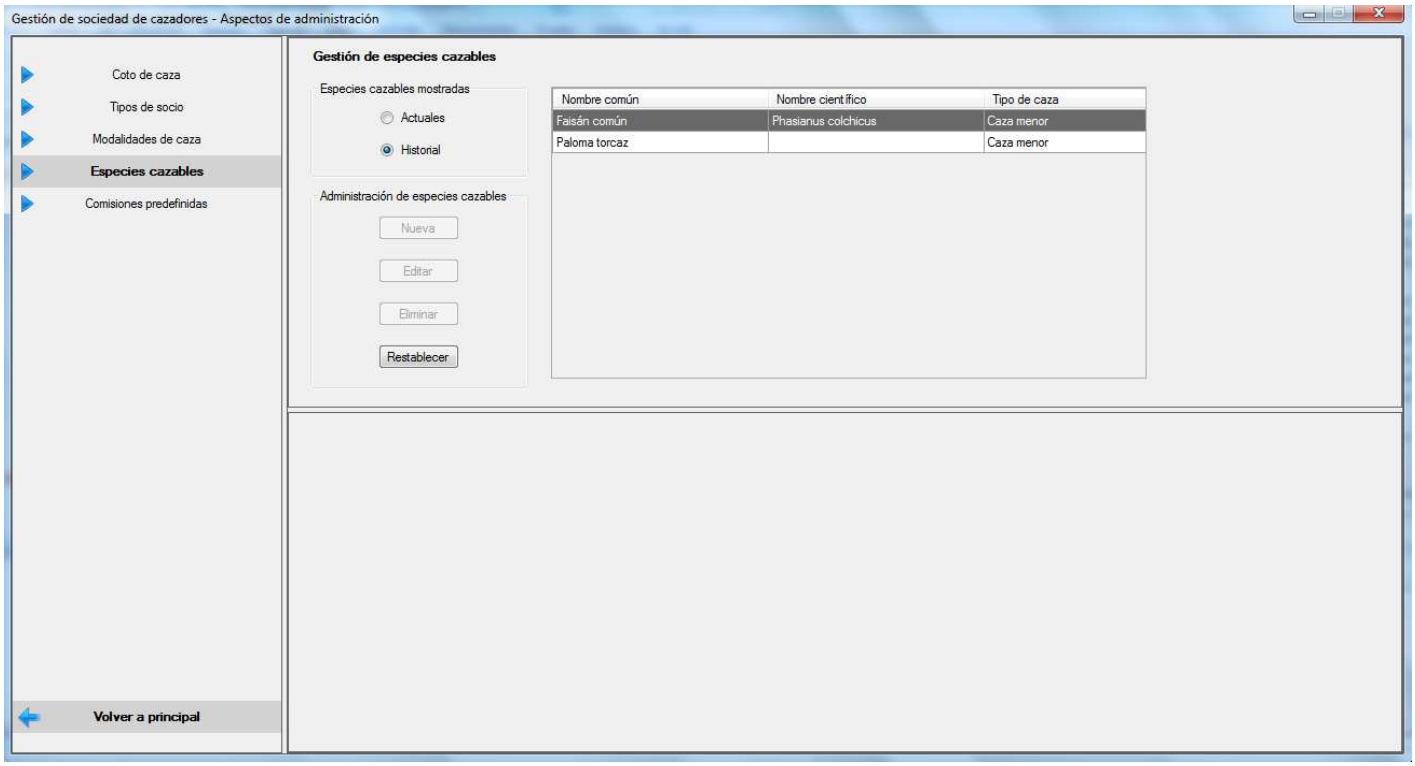

Figura 50. Formulario Aspectos de administración: Especies cazables.

Restablecimiento de especie cazable de caza

La figura 50 muestra la captura de pantalla cuando se está realizando la operación de restablecer una especie cazable. Para realizar dicha operación hay que seguir los siguientes pasos:

1. En el grupo *Especies cazables mostradas* se hace clic sobre el botón *Historial*.
2. Seleccionar la especie cazable que se desea restablecer, en la captura "Faisán común".
3. En el grupo *Administración de especies cazables* hacer clic sobre el botón *Restablecer*.
4. Se muestra un mensaje informando que la operación se ha realizado con éxito.

C.2.5. Gestión de comisiones predefinidas

Para acceder a la gestión de comisiones predefinidas en la ventana Aspectos de administración hacer clic sobre el botón *Comisiones predefinidas* del panel de la izquierda. El aspecto que tiene la ventana Aspectos de administración es el de la figura 51 en la que se gestionan las comisiones predefinidas que podrán formar parte de las juntas directivas que rigen la sociedad de cazadores.

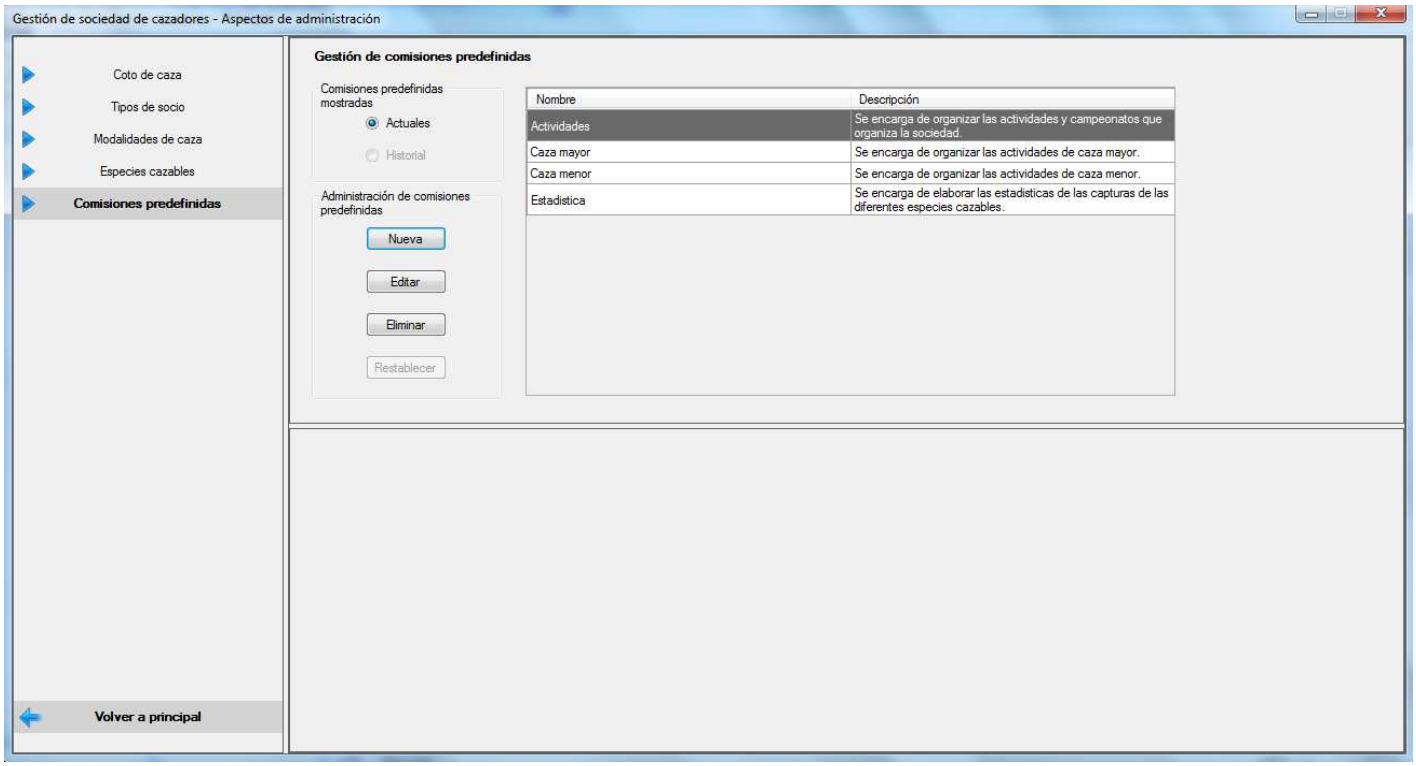

Figura 51. Formulario Aspectos de Administración: Comisiones predefinidas.

Si en figura 51 se hace clic sobre el botón *Volver a principal* se vuelve de nuevo a la ventana principal, figura 45.

C.3. Gestión de temporada

En este apartado se explica cómo se gestiona una temporada de principio a fin. Lo que provocara que también se haga la gestión de juntas directivas y la gestión de socios. Dicha administración constará de la siguiente sucesión de operaciones.

1. Gestión de junta directiva que regirá la nueva temporada.
2. Alta de temporada.
3. Configuración de las cuotas y modalidades.
4. Inicio de temporada.
5. Gestión de temporada en curso:
   - Gestión de socios.
   - Gestión de actividades
   - Gestión de actuaciones en el coto de caza.
   - Gestión de contabilidad.
6. Finalización de temporada.

C.3.1. Gestión de junta directiva que regirá la temporada

Si la nueva temporada es regida por la junta directiva actual no hará falta dar de alta una junta directiva.

Para acceder a la gestión de juntas directivas en la ventana principal, figura 45, hacer clic sobre el botón *Juntas directivas*. Aparecerá la ventana Juntas directivas, figura 52.

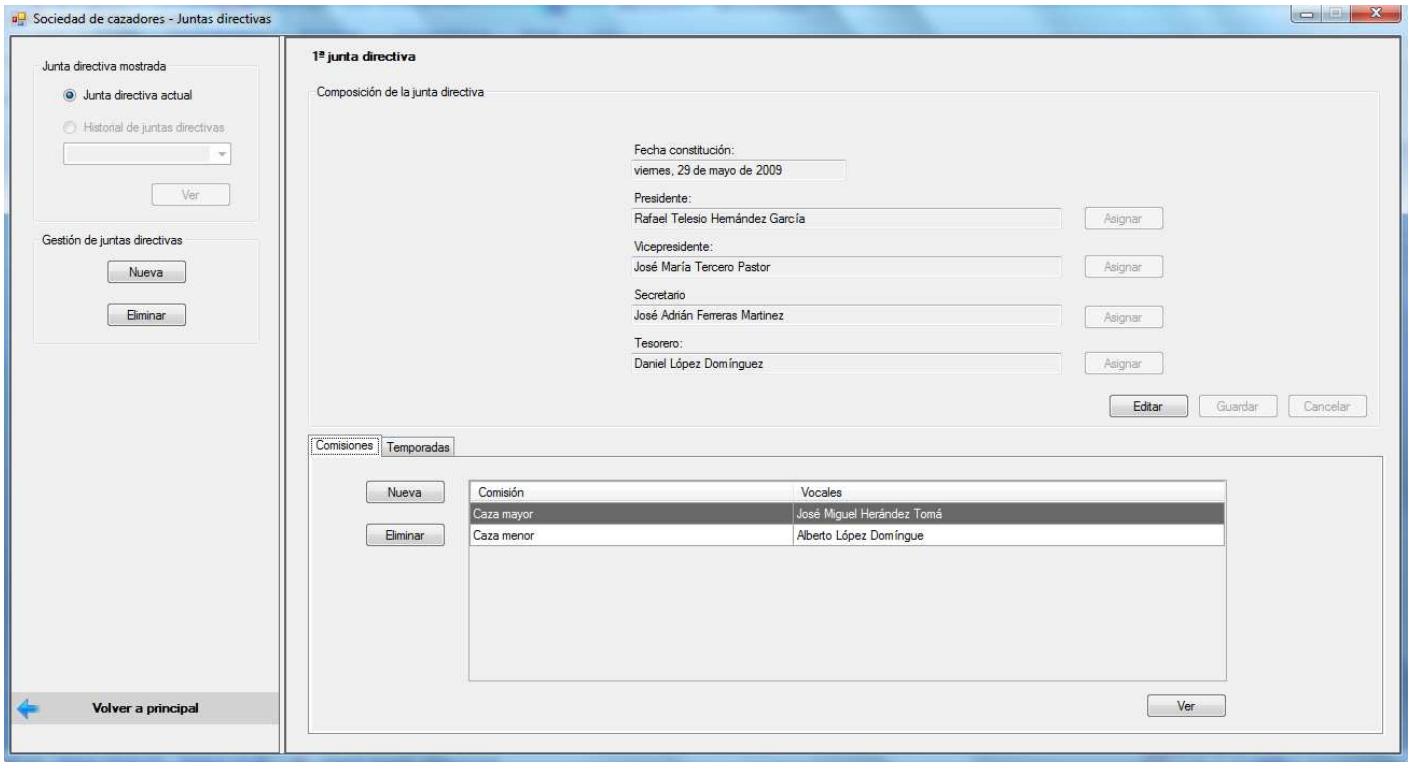

Figura 52. Formulario Juntas directivas.

Ver junta directiva actual

Para ver la junta directiva actual de la sociedad de cazadores se realiza el siguiente paso:

1. En el grupo *Junta directiva mostrada* hacer clic sobre botón *Junta directiva actual.*

En el panel de la derecha se muestra información de la junta directiva que se divide en:

- En grupo *Composición de la junta directiva* donde se muestra la fecha de constitución y los cargos principales.
- En la pestaña *Comisiones* se muestran las comisiones junto a sus vocales responsables.
- En la pestaña Temporadas se muestran las temporadas que ha gestionado.

Alta de junta directiva

Cuando se da de alta una junta directiva esta pasa a ser la junta directiva actual de la sociedad de cazadores, la cual podrá regirá la temporada actual si está en curso y temporadas futuras. Para realizar la operación alta de junta directiva los pasos a seguir serán los siguientes:

1. En el grupo *Gestión de junta directiva* hacer clic sobre el botón *Nueva.*
2. Establecer la fecha de constitución.
3. Asignar los cargos de presidente, vicepresidente, secretario y tesorero. En la figura 53 se muestra cuando se está asignando el cargo de presidente.

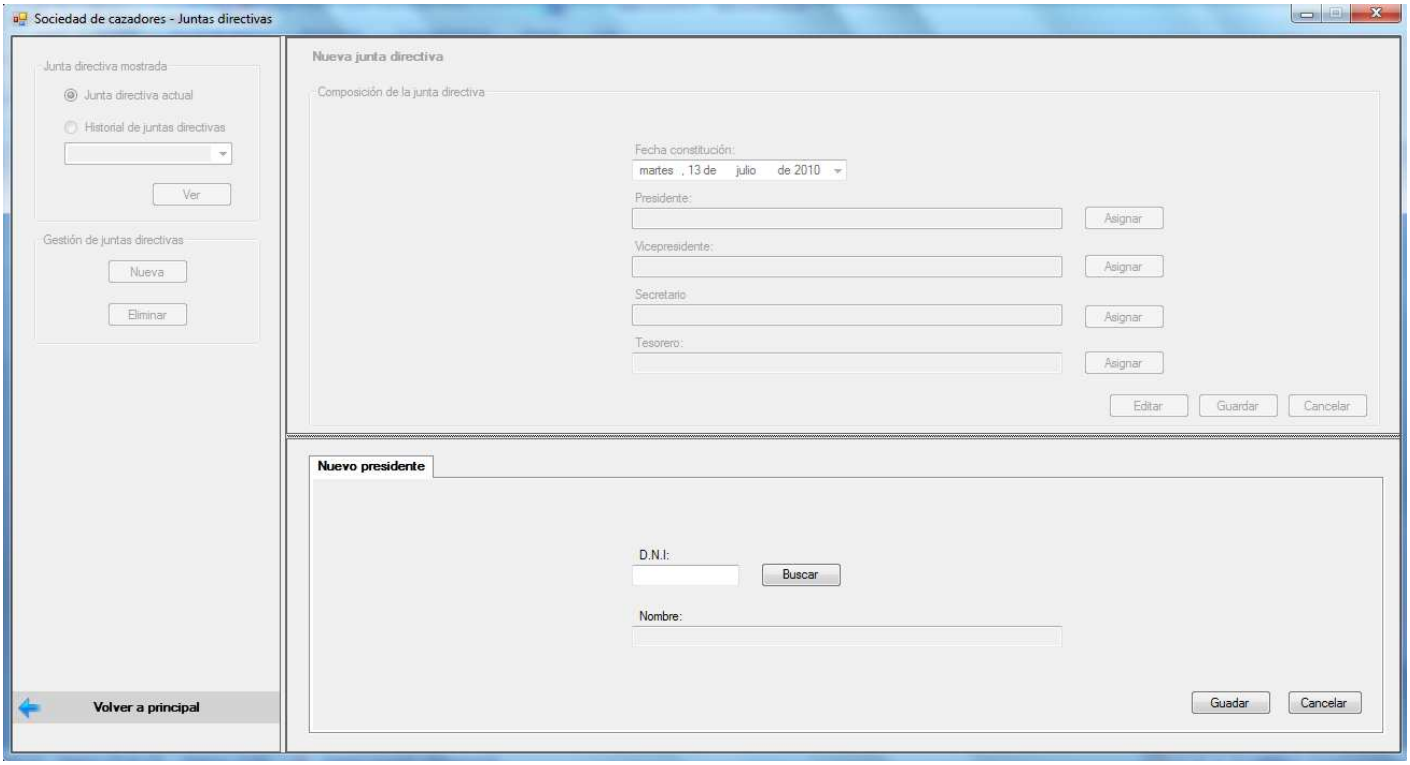

Figura 53. Formulario Juntas directivas: Asignar cargo.

3.1. Tras habilitarse el panel de abajo hacer clic en botón asignar del cargo que se desea añadir, en la figura 53 se ha hecho clic en el botón *Asignar* correspondiente al presidente.

3.2. Introducir el DNI del socio que asume el cargo.

3.3. Hacer clic en el botón *Buscar*.

3.4. Si el DNI introducido es correcto y existe el socio actual el sistema introduce el nombre.
Si el DNI introducido es erróneo o no existe un socio actual se muestra mensaje de error.

3.5. Si se ha encontrado al socio que ocupará el cargo hacer clic en el botón *Guardar* del panel de abajo.

4. Se muestra un mensaje informando que la operación se ha realizado con éxito.
5. Cuando se han establecido todos los cargos hacer clic en el botón *Guarda*r del panel superior.
6. Se muestra un mensaje informando que la operación se ha realizado con éxito, el aspecto de la ventana Juntas directivas que aparecerá será el de la figura 52.

Alta de comisión de junta directiva

Para realizar la operación de alta de comisión de junta directiva se deberá de hacer clic sobre el botón *Nuevo* de la pestaña *Comisiones*, figura 52.

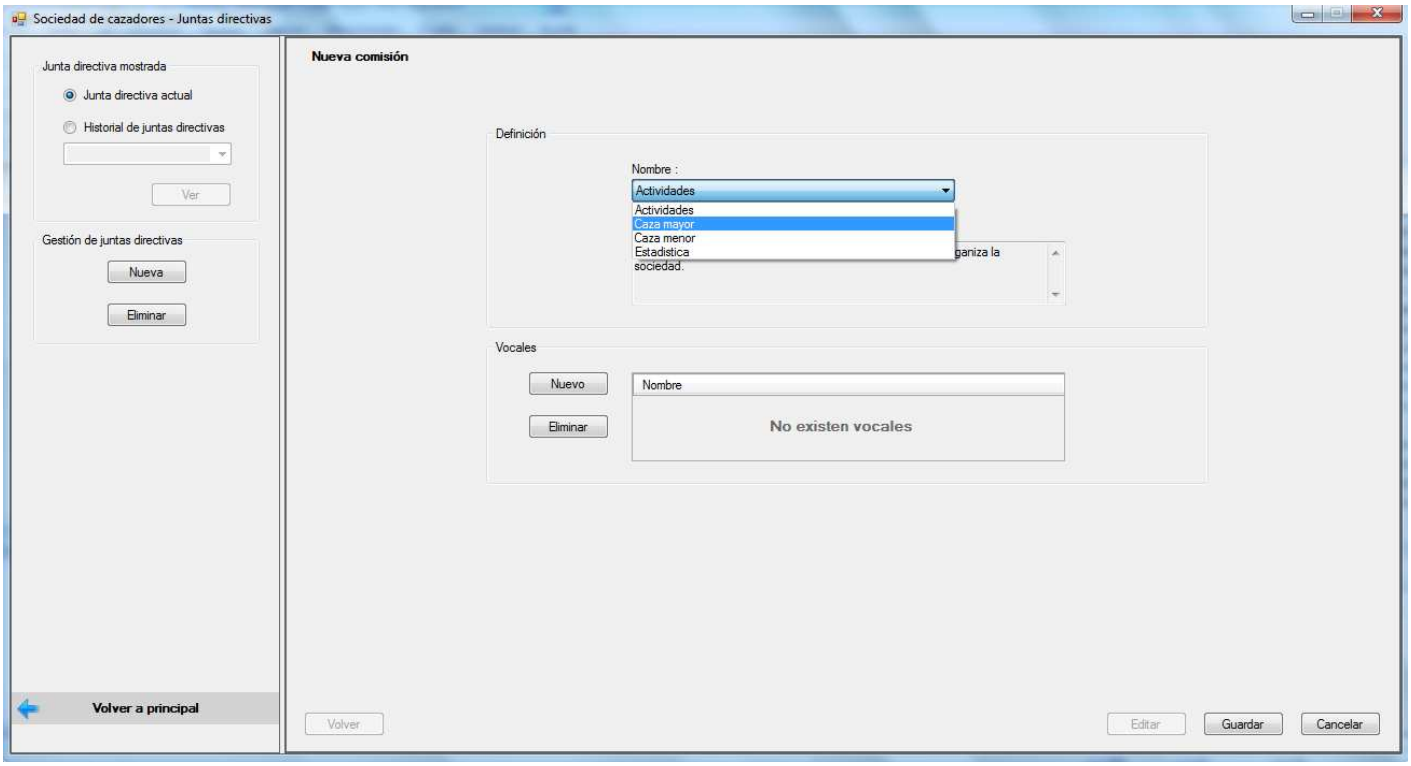

Figura 54. Formulario Junta Directiva: Comisión nueva.

La figura 54 muestra la captura de pantalla cuando se está realizando dicha operación. Los pasos a seguir son los siguientes pasos:

1. Seleccionar la comisión predefinida, en la figura 54 “Caza mayor”.
2. Asignar los vocales mediante el botón *Nuevo.* Se seguirán los mismos pasos que para asignar un cargo de la junta directiva.
3. Hacer clic sobre el botón *Guardar*.
4. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello.
   Si no se ha añadido ningún vocal se muestra un mensaje informando de ello.

C.3.2. Alta de temporada

Una vez administrada la junta directiva que regirá la nueva temporada se deberá crear. Para realizar la operación alta de temporada lo primero que se debe hacer es acceder a la gestión de temporadas, para ello en la ventana principal, figura

45 se hace clic sobre el botón *Temporadas*. Aparecerá la ventana Temporadas, figura 55.

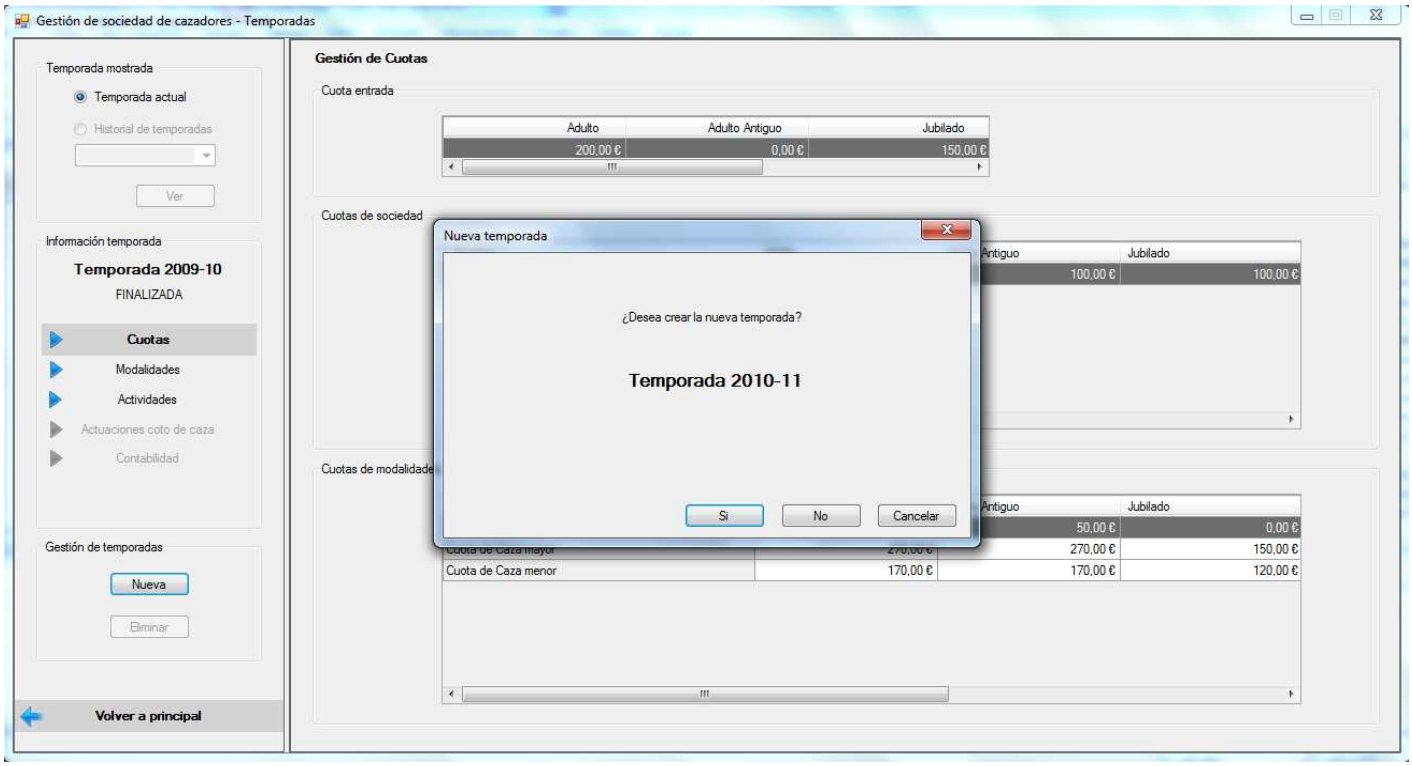

Figura 55. Formulario Temporadas: Nueva temporada.

Para poder dar de alta una temporada se deben cumplir dos condiciones respeto a la temporada que ahora es la actual.

- Debe de estar en estado FINALIZADA.
- Comenzó en el año anterior al actual.

La figura 55 muestra la captura de pantalla cuando se está realizando dicha operación. Los pasos a seguir son los siguientes pasos:

1. En el grupo *Gestión de temporadas* hacer clic sobre el botón *Nueva.*
2. Se muestra un mensaje en el que se debe confirmar si se crea la nueva temporada, figura 55.
3. Si se hace clic sobre el botón S*í* entonces*:*
   - Se da de alta la temporada que pasará a ser la temporada actual.
   - Se asociará la nueva temporada a la junta directiva actual.
   - La nueva temporada estará en el estado de CONFIGURACIÓN DE CUOTAS Y MODALIDADES.

C.3.3. Configuración de cuotas y modalidades

Lo primero que se hace cuando se da de alta una temporada es establecer las cuotas y modalidades que la sociedad de cazadores ha acordado.

C.3.3.1. Gestión de cuotas

Para acceder a la gestión de cuotas en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Cuotas*. El aspecto que tiene la ventana Temporadas es el de la figura 56 en la que se gestionan la cuota de entrada, las cuotas de sociedad y las cuotas de modalidades.

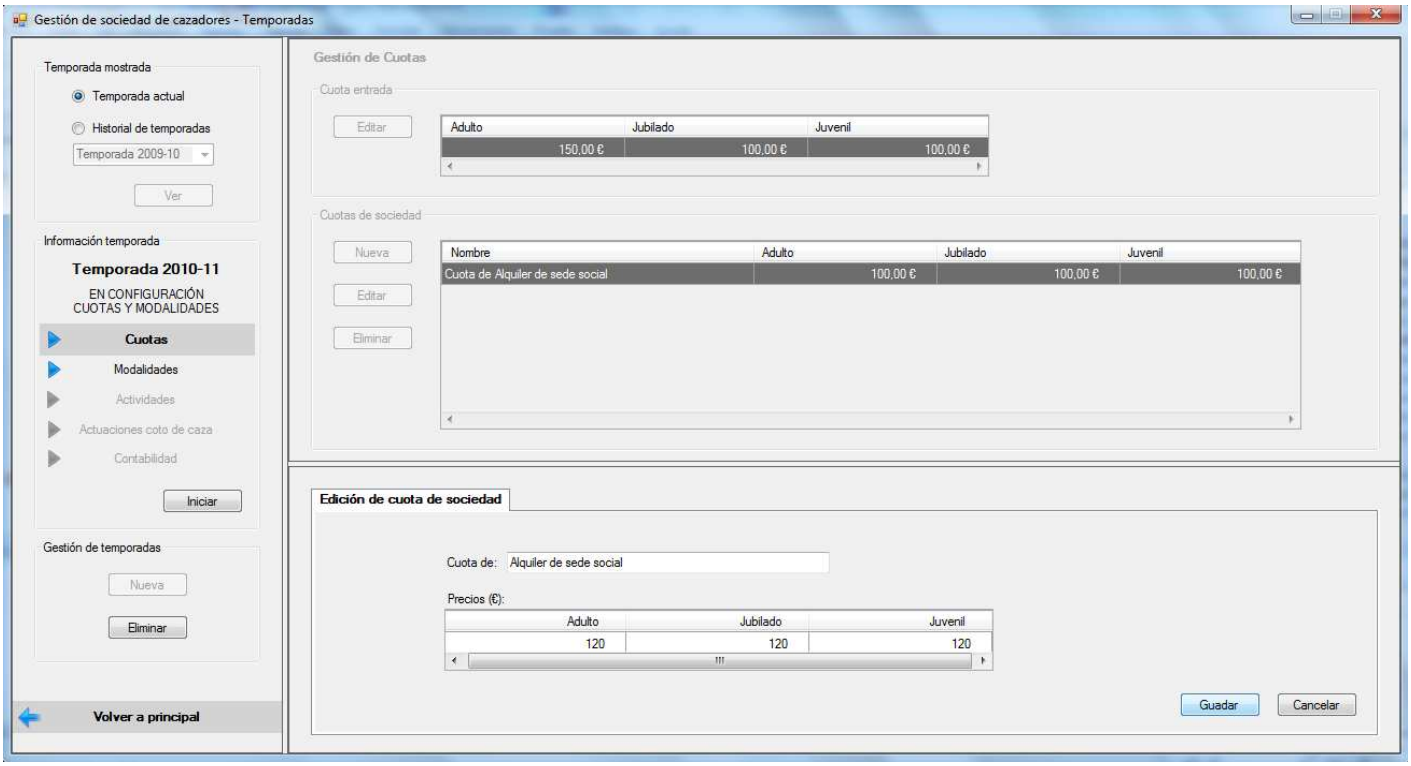

Figura 56. Formulario Temporadas: Cuotas.

Modificación de cuota de sociedad

La figura 56 muestra la captura de pantalla cuando se está realizando la operación de modificación de una cuota de sociedad. Para realizar dicha operación hay que seguir los siguientes pasos:

1. En el grupo *Cuotas de sociedad* seleccionar la cuota de sociedad a modificar, en la figura 46 “Cuota de Alquiler de sede social”.

2. En el grupo *Cuotas de sociedad* hacer clic sobre el botón *Editar* (deshabilitado en la figura 56).
3. Tras desplegarse el panel de abajo (cubre el grupo *Cuotas de modalidades*) se modifica el nombre o los precios de la cuota. En esta captura se ha modificado los precios de cuotas que han pasado de 100 € a 120 €.
4. Hacer clic sobre botón *Guardar*.
5. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello, se pliega el panel de abajo y muestran los cambios de la cuota en la lista.
   Si los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.
   Si se ha modificado el nombre y coincide con el de otra cuota se muestra un mensaje error informando de ello y se cancela la operación.

C.3.3.2. Gestión de Modalidades

Para acceder a la gestión de modalidades en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Modalidades*. El aspecto que tiene la ventana Temporadas es el de la figura 57 en la que se gestionan las modalidades que los socios podrán practicar.

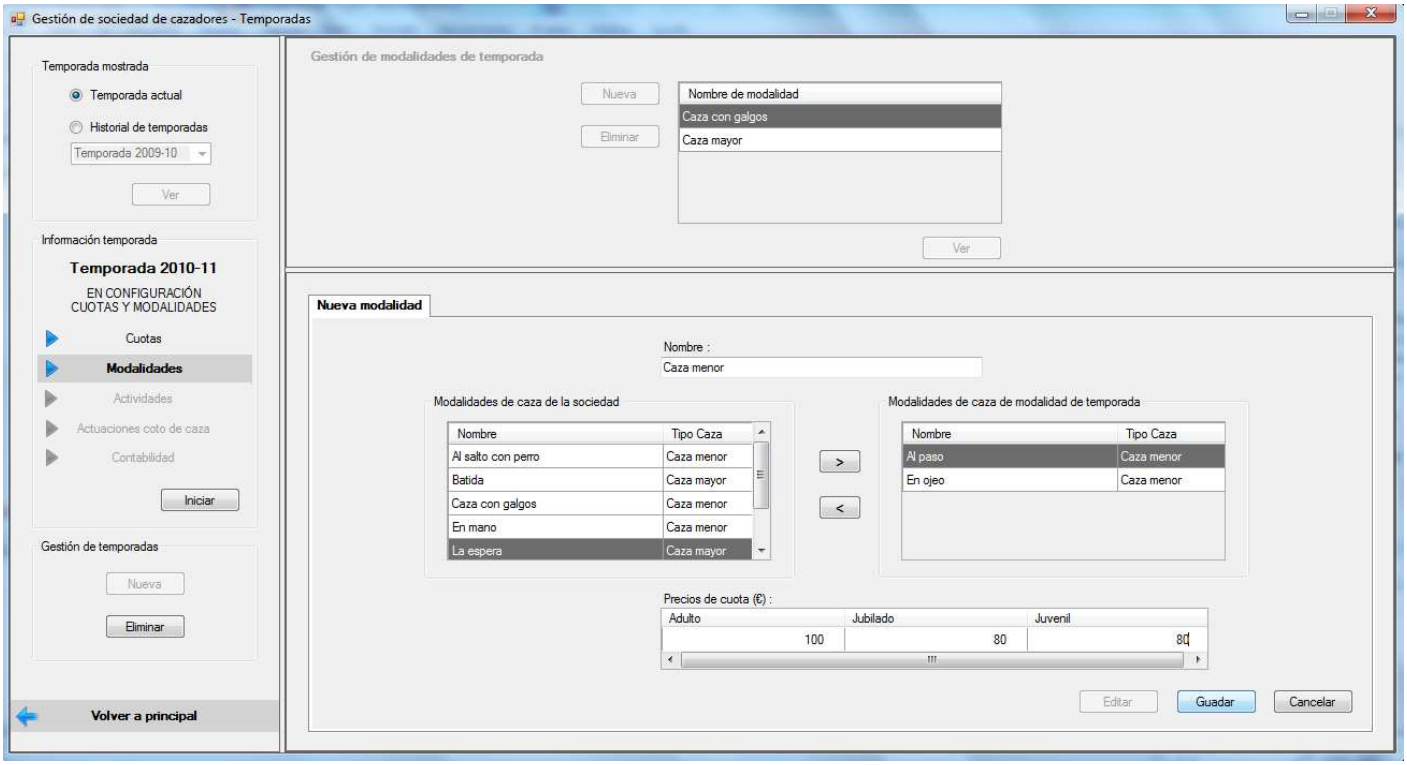

Figura 57. Formulario Temporadas: Modalidades.

Alta de modalidad

La figura 57 muestra la captura de pantalla cuando se está realizando la operación de alta de modalidad. Para realizar dicha operación hay que seguir los siguientes pasos:

1. En el panel superior hacer clic sobre el botón *Nuevo* (deshabilitado en la figura 57).
2. Tras habilitarse el panel de abajo se introducen el nombre.
3. Añadir las modalidades de caza que conforman la modalidad de temporada.
   3.1. En el grupo *Modalidades de caza de la sociedad* seleccionar la modalidad de caza del grupo de.
   3.2. Hacer clic en el botón >.
4. Introducir los precios de la cuota que el socio deberá pagar por practicar la modalidad.
5. Hacer clic sobre botón *Guardar*.
6. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello, en lista de modalidades se añade la nueva modalidad que se muestra en el panel de abajo con permiso de lectura.
   Si los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.

Si existe otra modalidad con el mismo nombre o las mismas modalidades de caza se muestra un mensaje de error informando de ello y se cancela la operación.

C.3.4. Inicio de temporada

Una vez que se ha terminado de configurar las cuotas y modalidades ya se puede iniciar la temporada para poder realizar el resto de gestiones de la misma y comenzar con la gestión de los socios que van a participar.

Para iniciar la temporada en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Iniciar*, figura 58.

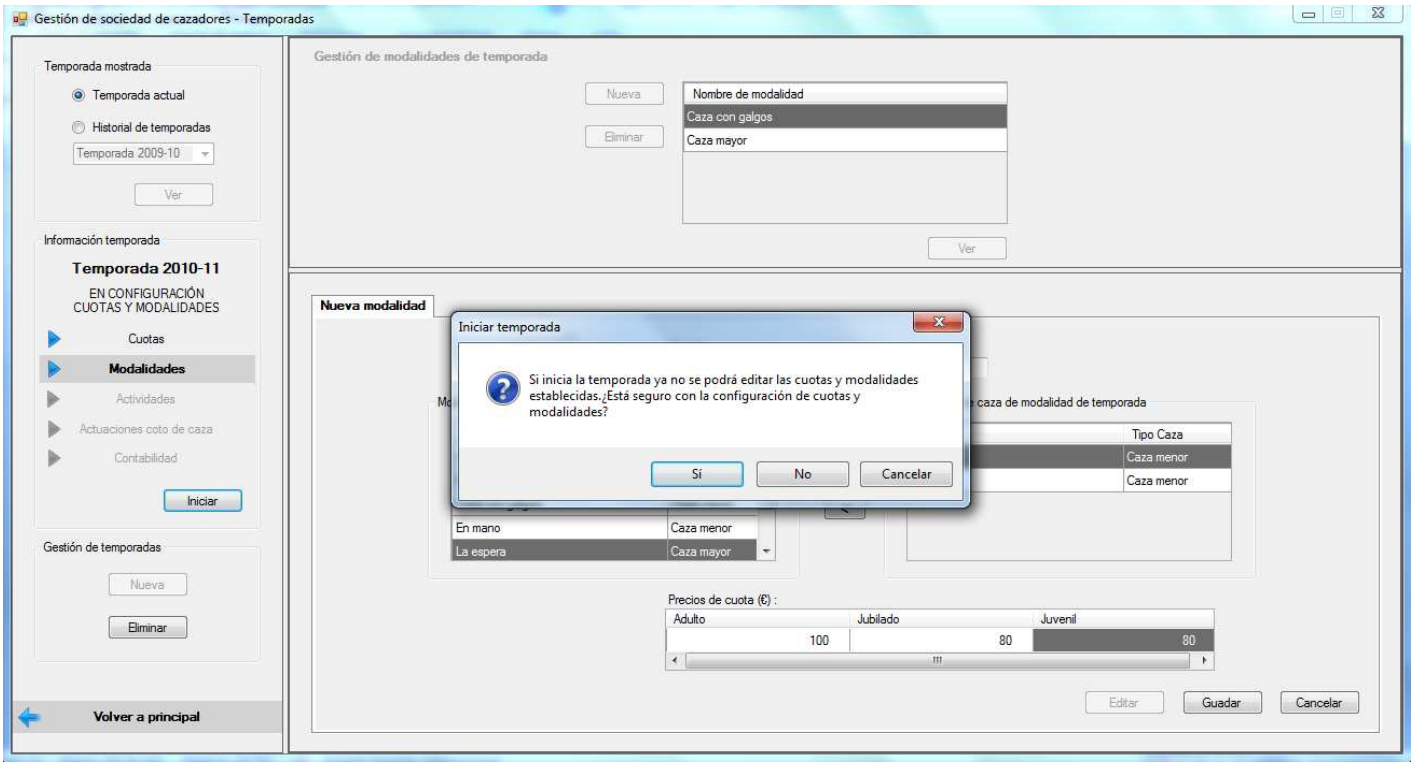

Figura 58. Formulario Temporadas: Inicio de temporada.

La figura 58 muestra el mensaje de confirmación de iniciar temporada. Si se confirma mediante el botón *Sí* entonces:

- No se podrán modificar ni las cuotas ni las modalidades establecidas.
- La temporada estará en el estado de EN CURSO.
- Se habilitarán el resto de opciones de una temporada.

C.3.5. Gestión de temporada en curso

C.3.5.1. Gestión de socios

Para acceder a la gestión de socios en la ventana principal, figura 45 hacer clic sobre el botón *Socios*. Aparecerá la ventana Socios, figura 46, donde se realizaran todas las gestiones referentes a los socios que conforman la sociedad de cazadores

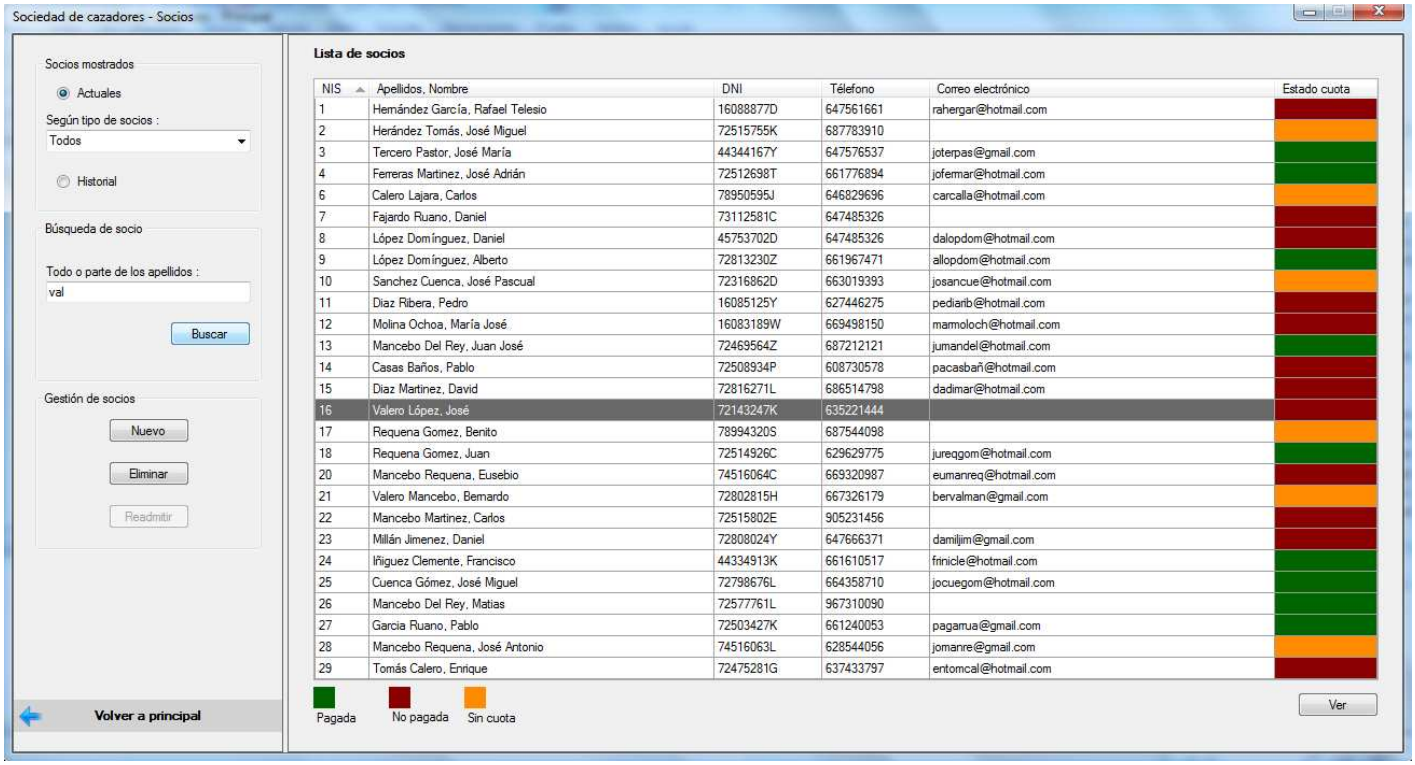

Figura 59. Formulario Socios.

La ventana Socios funciona de la siguiente forma, en el panel de la derecha se realiza la administración de los socios y en el panel de la izquierda se muestran los socios en relación a dichas gestiones.

En el grupo *Socios mostrados* se seleccionan los socios que se desean mostrar:

1. Hacer clic sobre el botón *Actuales* si se desean ver los socios actuales o sobre el botón *Historial* si se desean ver los socios antiguos.
2. Si se ha hecho clic sobre el botón Actuales, se puede seleccionar el conjunto de socios que se desean mostrar en función del tipo al que pertenece o seleccionar ver todos.

En el grupo *Búsqueda de socio* se puede buscar de forma rápida un socio para ello los pasos a seguir son:

1. Introducir parte de los apellidos del socio a buscar, en la figura 59 se introdujo “val”.
2. Hacer clic sobre el botón *Buscar*.
3. Se selecciona en la lista el socio buscado, en la figura 59 el socio con “Valero Mancebo Bernardo”.

En los socios mostrados en la lista se indica el estado en que se encuentra el estado de la cuota de la temporada actual.

- Pagada, color verde.
- No pagada, color rojo.
- Sin confirmar, color naranja, el socio no ha confirmado que va ha participar en la temporada actual.

Alta de socio

Para realizar la operación dar de alta un socio la temporada actual deberá de estar en estado EN CURSO, es decir tras haber establecido las cuotas y modalidades. Para realizar dicha operación hay que seguir los siguientes pasos:

1. En el grupo *Gestión de socios* hacer clic sobre el botón *Nuevo* de la izquierda (deshabilitada en la figura 59).
2. Introducir el DNI del nuevo socio (en modo lectura en la figura 59).
3. Hacer clic sobre el botón *Buscar* (deshabilitado en la figura 59).
   3.1. Si existe un socio con el DNI introducido se muestra un mensaje error, figura 60.

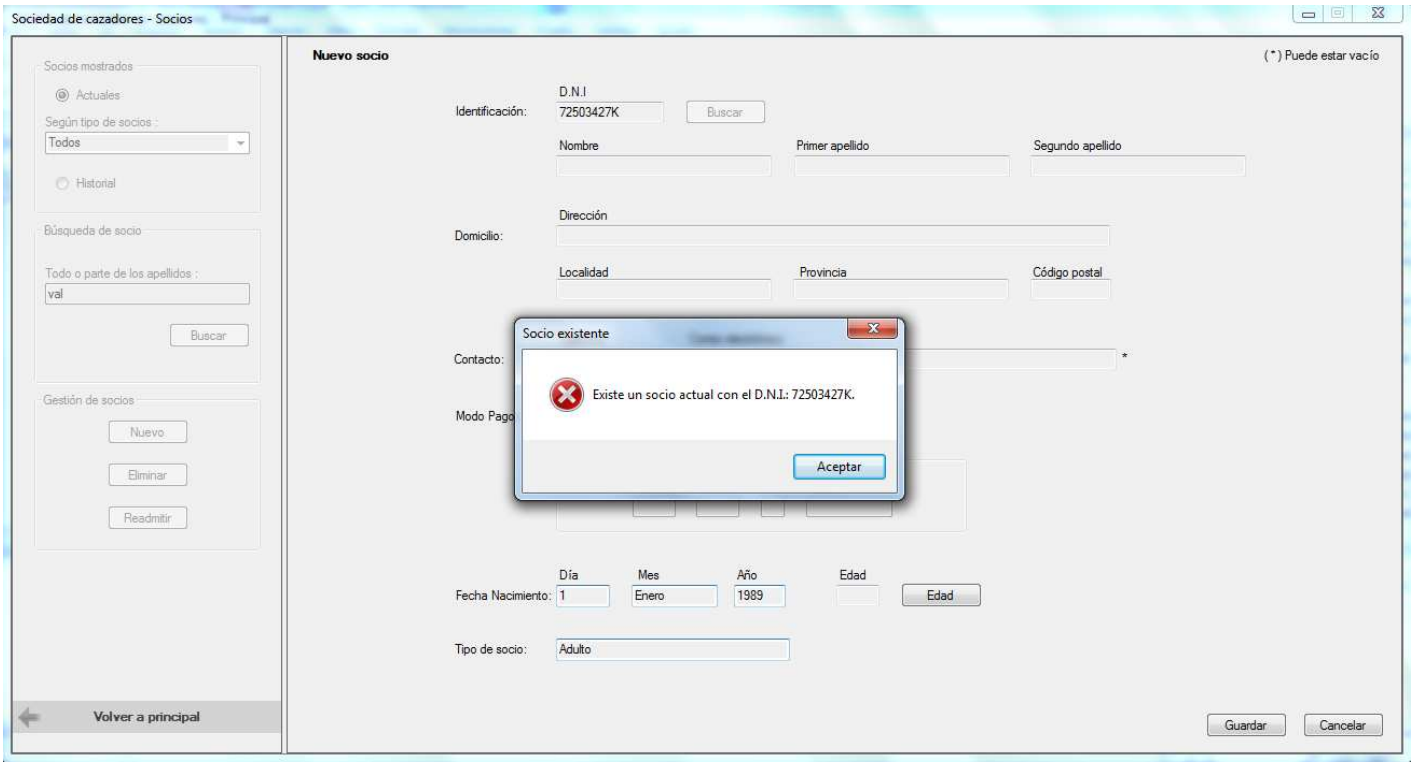

Figura 60. Formulario socios: DNI socio existente.

3.2. Si no existe otro socio se introducen los datos que identificarán al nuevo socio.

4. Hacer clic en el botón *Guardar.*

5. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello, dando el NIS que identificará al socio en la sociedad.

   Si los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes, figura 61.

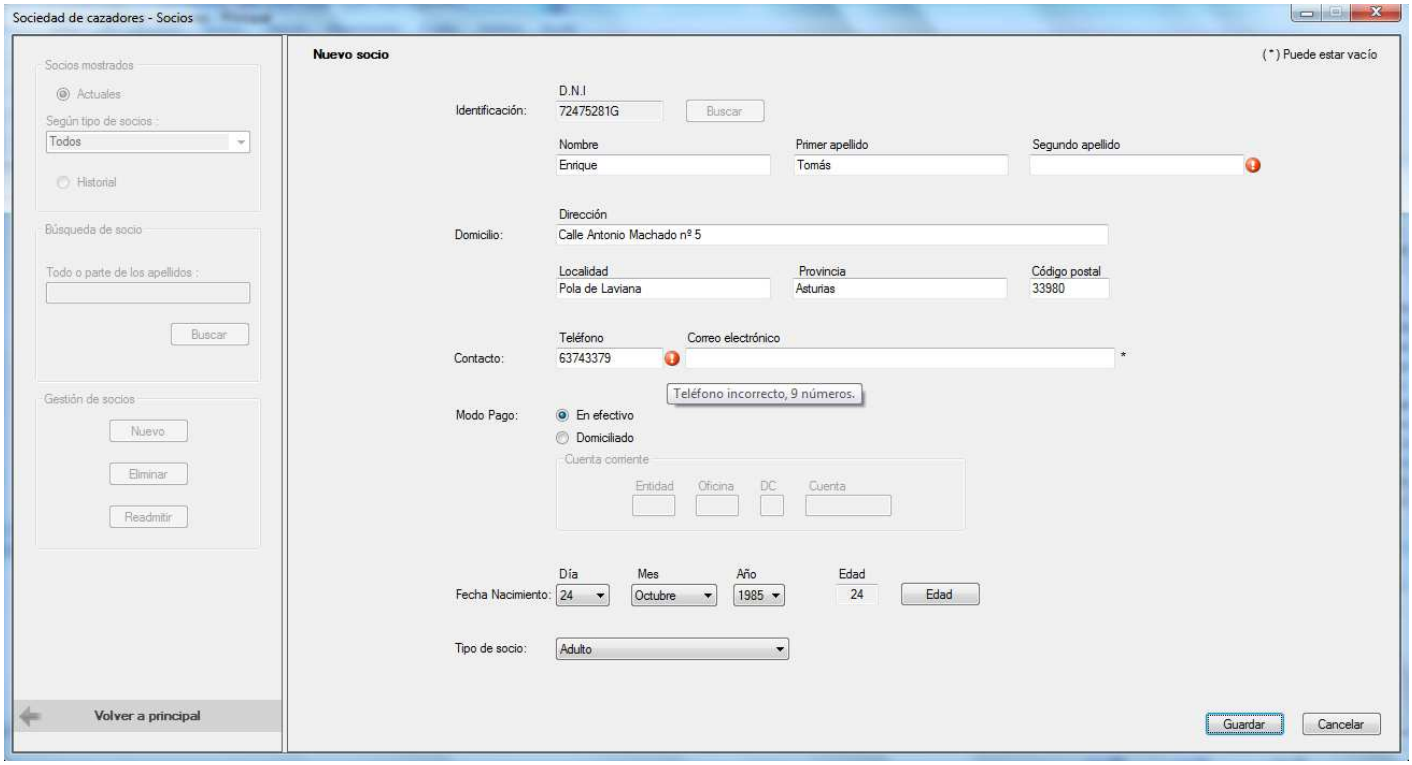

Figura 61. Formulario socios: Datos nuevo socio erróneos.

Ver socio

Para realizar la operación de ver un socio en detalle habrá dos formas de hacerlo, figura 59:

1. Seleccionar al socio en la lista de socios y hacer clic sobre el botón *Ver*.
2. Hacer doble clic sobre el socio.

La ventana que aparecerá es la ventana Socio, figura 62.

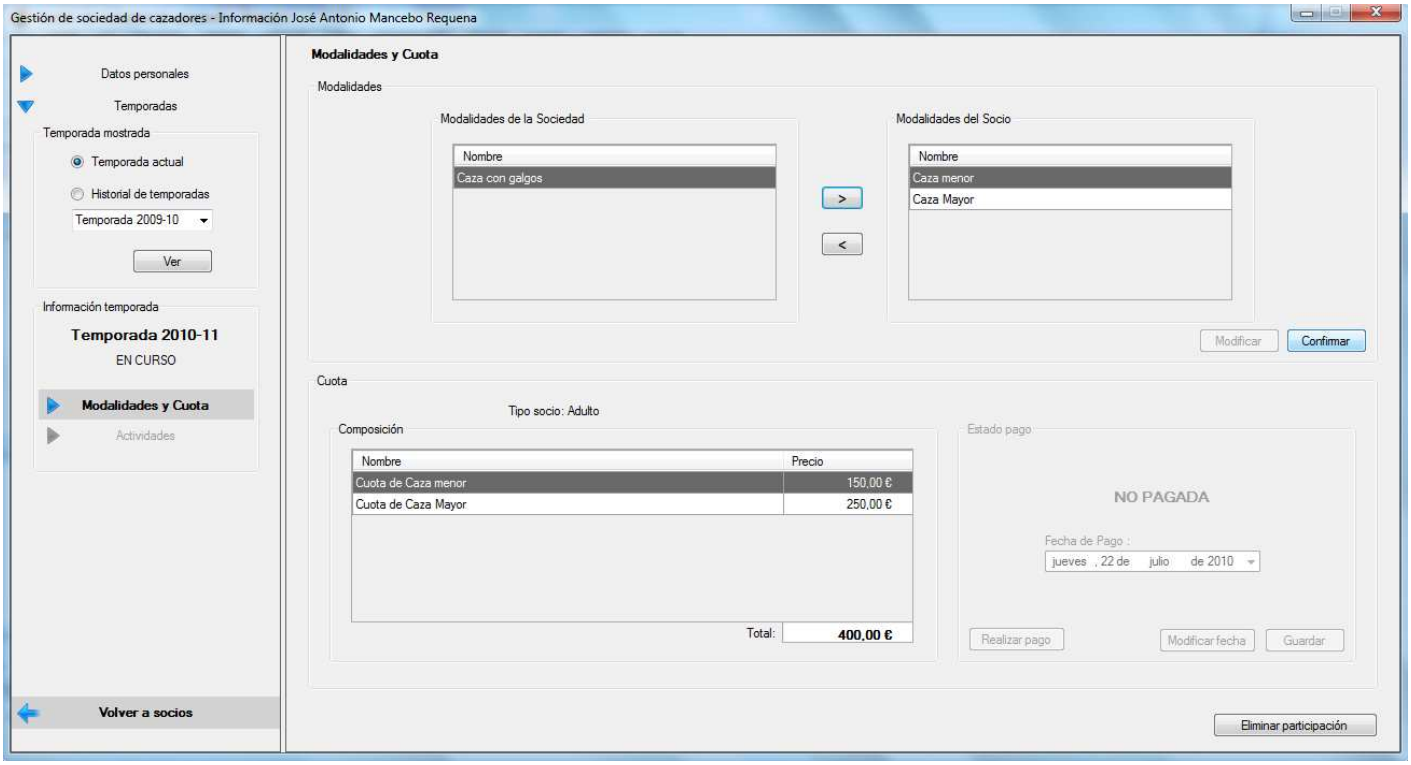

Figura 62. Formulario Socio: Configuración de modalidades y cuota.

Configuración de modalidades y cuota

La figura 62 muestra la captura de pantalla cuando se está realizando la operación de configuración de modalidades y cuota. Para realizar dicha operación hay que seguir los siguientes pasos:

1. Hacer clic en el botón *Temporadas*.
2. Hacer clic en el botón *Temporada actual*.
3. Hacer clic en el botón *Modalidades y Cuota*.
   3.1. Si el socio no ha confirmado hasta ahora que desea participar en la temporada actual se muestra un mensaje en el que se debe confirmar su participación.
4. Añadir las modalidades de temporada que el socio dese practicar.
   4.1. En el grupo de *Modalidades de la sociedad* seleccionar la modalidad.
   4.2. Hacer clic en el botón >.
   4.3. En el grupo *Composición* se añade la cuota de modalidad en la lista de cuotas que conforman la cuota del socio,
5. Hacer clic sobre el botón *Confirmar*.
6. Se habilita el Grupo *Estado pago*.

Realizar pago de cuota

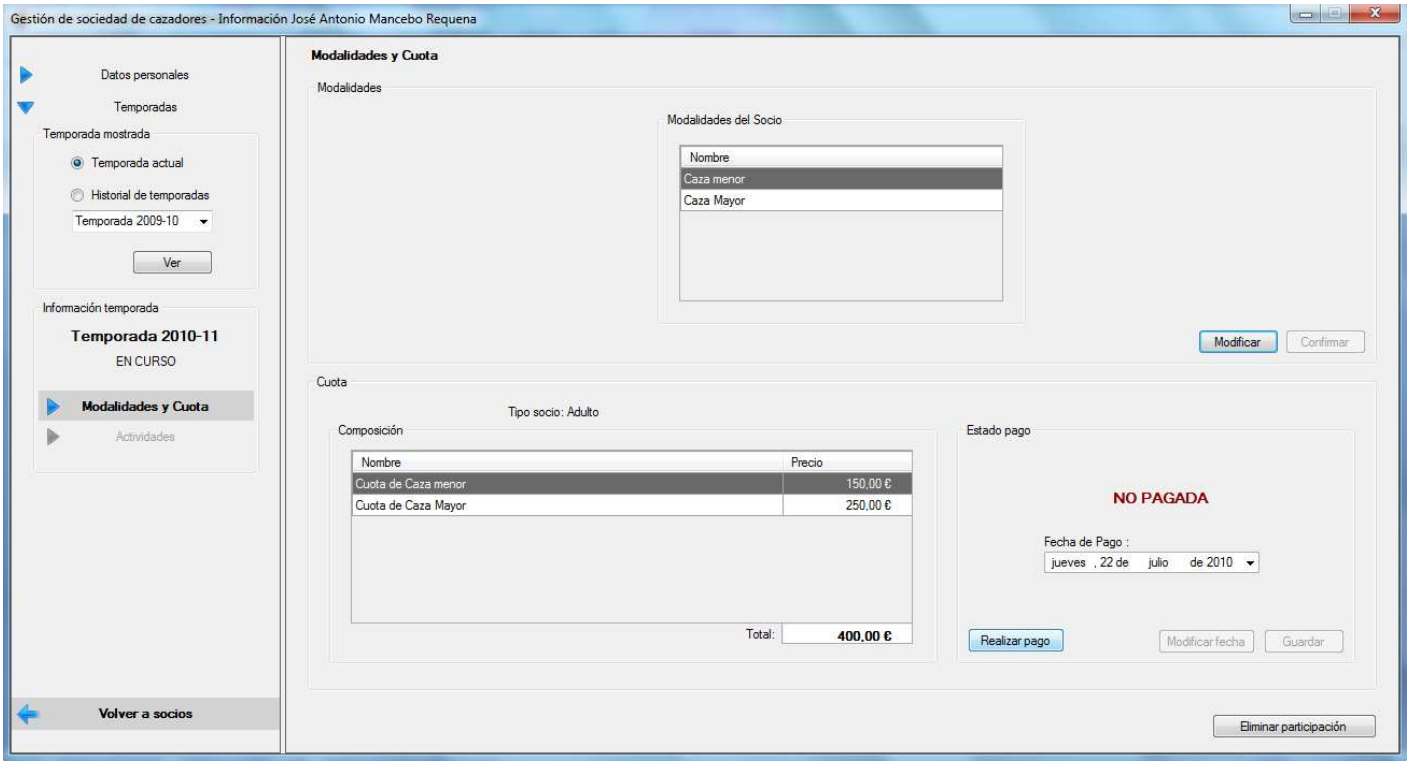

Figura 63. Formulario Socio: Pago cuota.

La figura 63 muestra la captura de pantalla cuando se está realizando la operación de pago de cuota en el grupo de *Estado de pago*. Para realizar dicha operación hay que seguir los siguientes pasos:

1. Seleccionar la fecha en que se realiza el pago.
2. Hacer clic sobre el botón *Realizar pago*.
3. Se cambiará el estado de pago a “PAGADA” (en verde),
4. Se habilitaran los botones *Modificar fecha* y *Guardar* por si se quiere modificar la fecha de pago,
5. Se hará visible el botón *Eliminación de pago*, por si se deshacer el pago por alguna circunstancia.

Si el socio confirmo participar en la temporada actual pero por algún motivo no desea participar se puede eliminar la participación mediante el botón *Eliminar participación*.

C.3.5.2. Gestión de actividades

Para acceder a la gestión de actividades en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Actividades*. El aspecto que tiene la ventana Temporadas es el de la figura 64 en la que se gestionan las actividades de sociedad y campeonatos de cazadores que la sociedad de cazadores realiza.

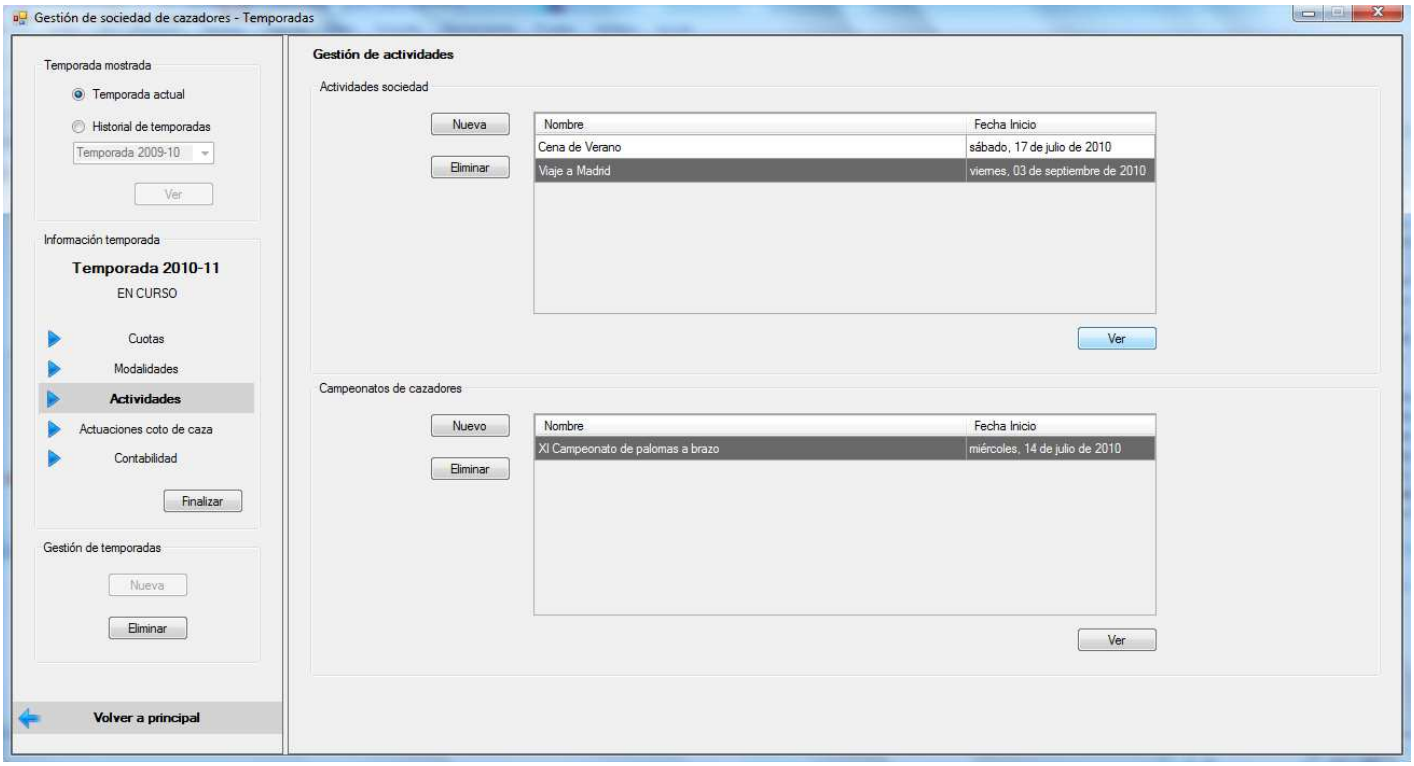

Figura 64. Formulario Temporadas: Actividades.

Ver actividad

Para realizar la operación de ver una actividad en detalle habrá dos formas de hacerlo, figura 64:

1. Seleccionar la actividad en la lista de actividades de sociedad o en la lista de campeonatos de cazadores y hacer clic sobre el botón *Ver*.
2. Hacer doble clic sobre la actividad a ver.

La ventana que aparecerá es la ventana Actividad, figura 65.

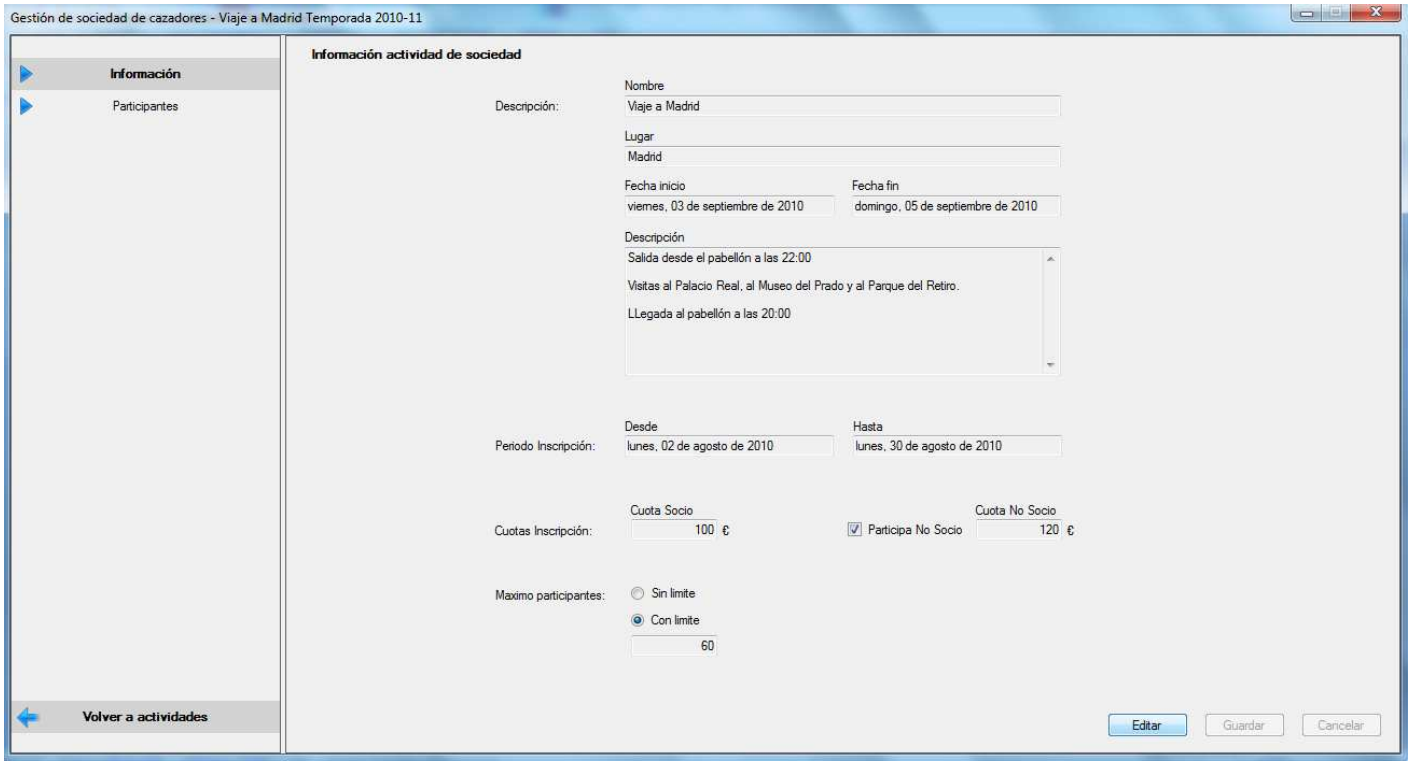

Figura 65. Formulario Actividad: Edición de datos actividad de sociedad.

Modificación de información de actividad

Para acceder a la información de la actividad en la ventana Actividad hacer clic sobre el botón *Información* del panel de la izquierda.

Para realizar la operación de modificación de de información de actividad los pasos a seguir son los siguientes:

1. Hacer clic sobre el botón *Editar*.
2. Tras establecerse los datos con permiso de escritura se modifican los datos que se desea.
3. Hacer clic sobre botón *Guardar*.
4. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello y muestra los datos con permiso de lectura.
   Si los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.
   Si se modifica el nombre y la fecha de inicio y coinciden con el de otra actividad se muestra un mensaje de error informando de ello y se cancela la operación.

Participantes actividad

Para acceder a los participantes de la actividad en la ventana Actividad hacer clic sobre el botón *Participantes* del panel de la izquierda, el cual estará deshabilitado si no ha comenzado el periodo de inscripción establecido. El aspecto de la ventana Actividad será el siguiente, figura 66.

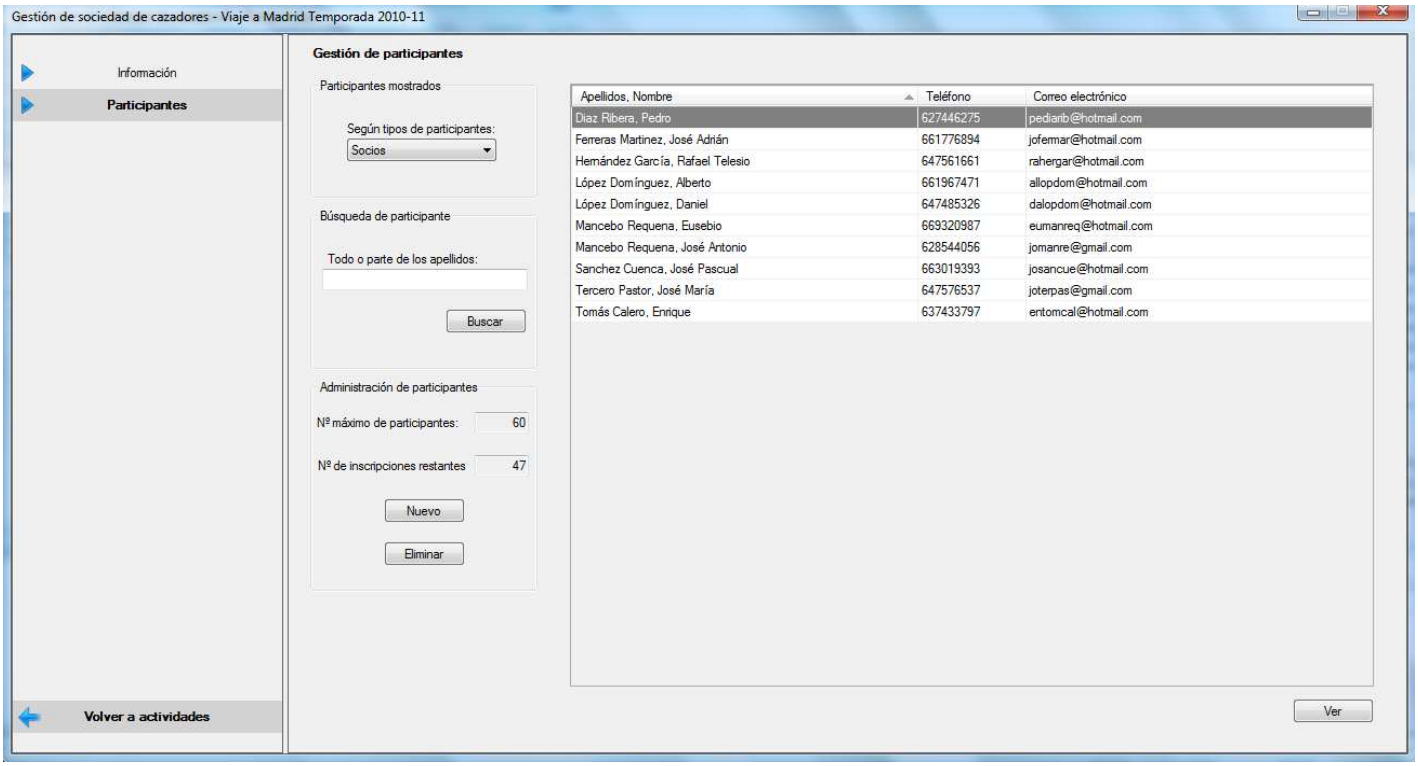

Figura 66. Formulario Actividad: Participantes.

En el grupo *Participantes mostrados* se selecciona los participantes que se quieren mostrar: todos, socios y no socios, en la figura 66 se muestran los participantes socios.

Inscripción de participante actividad

Para realizar la operación de inscribir un participante en el grupo *Administración de participantes* hacer clic sobre el botón *Nuevo*, figura 66. El aspecto de la ventana Actividad será el siguiente, figura 67.

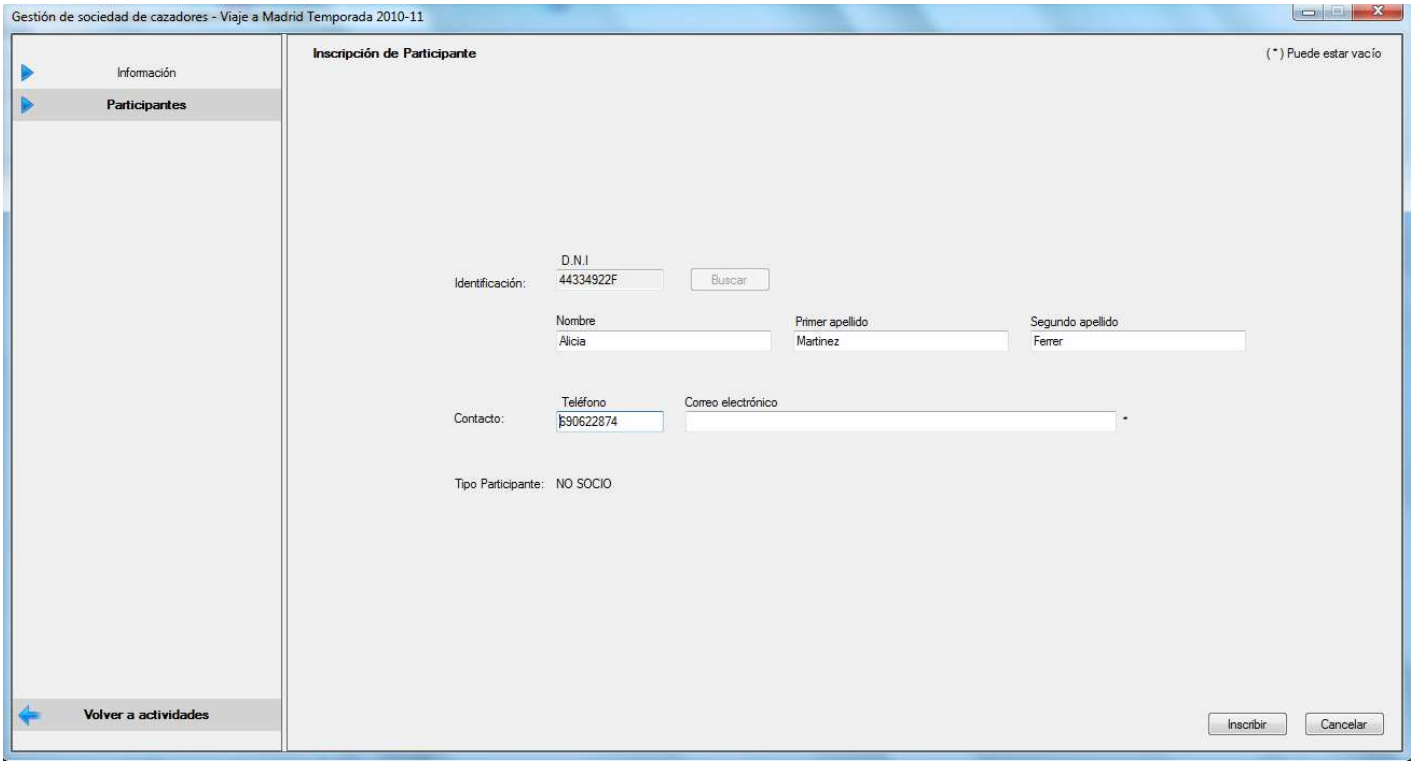

Figura 67. Formulario Actividad: Inscripción de participante.

Los pasos a seguir para realizar dicha operación son los siguientes.

1. Introducir el DNI del nuevo participante (en modo lectura en la figura 67).
2. Hacer clic sobre el botón *Buscar* (deshabilitado en la figura 67).
3. Si existe un socio o un participante no socio con el DNI introducido se rellenan el resto de campos que lo identifican en la actividad.
4. Si el nuevo participante no ha participado en la sociedad se introducen los datos que se requieren.
5. Hacer clic sobre el botón inscribir.
6. Si la operación se ha realizado con éxito se muestra un mensaje informando de ello.
   Si los datos introducidos son incorrectos se muestran indicadores donde se informa de los errores existentes.

Clasificación campeonatos cazadores.

Si se quiere acceder a la clasificación de un campeonato se debe hacer clic sobre el botón Clasificación que aparece en el panel de la derecha de la ventana actividad, figura 68 a continuación mostrada.

Para acceder a la clasificación de un campeonato de cazadores en la ventana Actividad (campeonato de cazadores) hacer clic sobre el botón *Clasificación* del panel de la izquierda, el cual estará deshabilitado si no ha comenzado el campeonato. El aspecto de la ventana Actividad será el siguiente, figura 68.

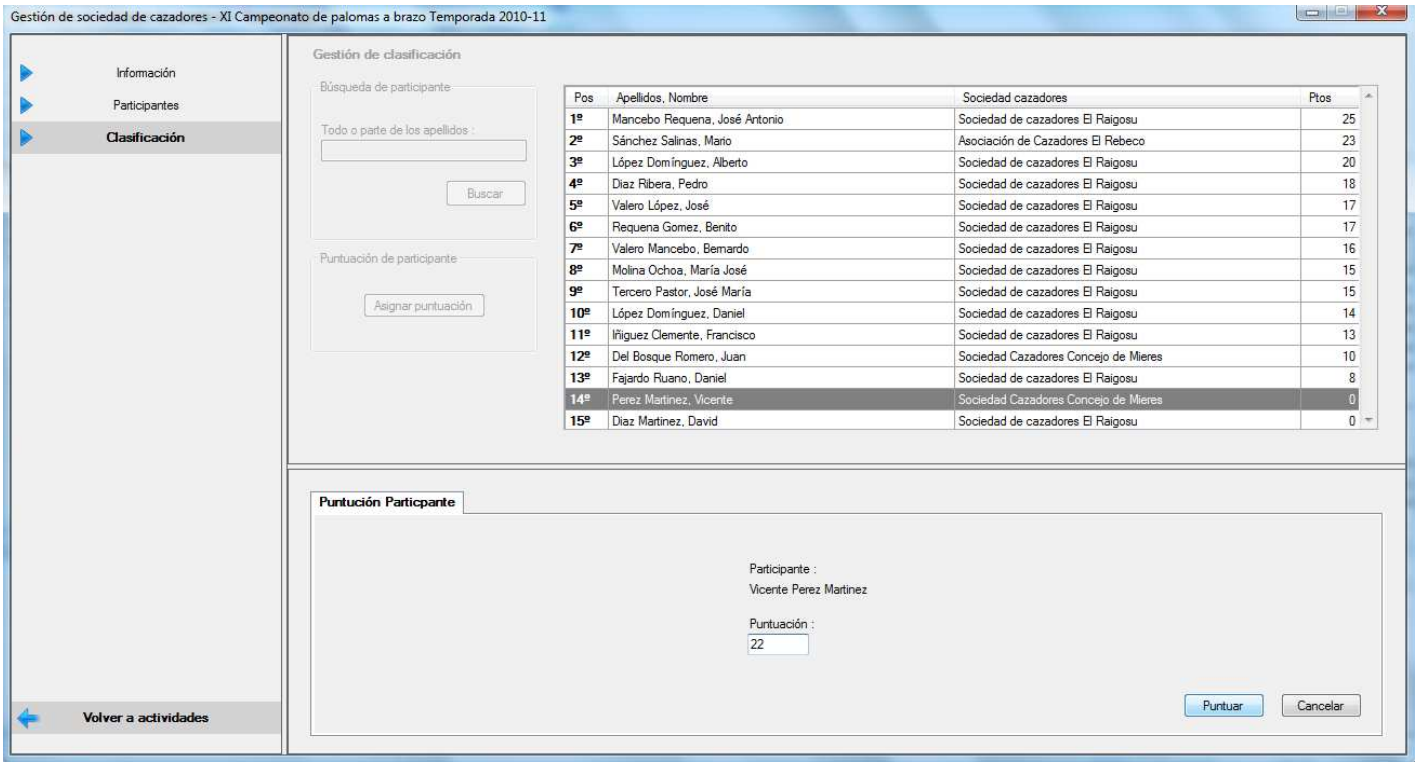

Figura 68. Formulario Actividad: Clasificación Campeonato.

Asignación de puntuación

La figura 68 muestra la captura de pantalla cuando se está realizando la operación de asignación de puntuación. Para realizar dicha operación hay que seguir los siguientes pasos:

1. Seleccionar al participante a puntuar en la lista de participantes, en la figura 68 “Pérez Martínez, Vicente”.
2. Hacer clic sobre el botón *Asignar puntuación* (deshabilitado en la figura 68).
3. Tras desplegarse el panel de abajo se introducirá la nueva puntuación del participante.
4. Hacer clic sobre el botón *Puntuar*.
5. Si la operación se ha realizado con éxito se muestra mensaje informando de ello.

   Si la puntuación es errónea se muestra un indicador informando de ello.

C.3.5.3. Gestión de actuaciones en el coto.

Para acceder a la gestión de actuaciones en el coto en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Actuaciones coto de caza*. El aspecto que tiene la ventana Temporadas es el de la figura 69 en la que se gestionan las mejoras y repoblaciones que se realizan en el coto de caza de la sociedad de cazadores en una temporada.

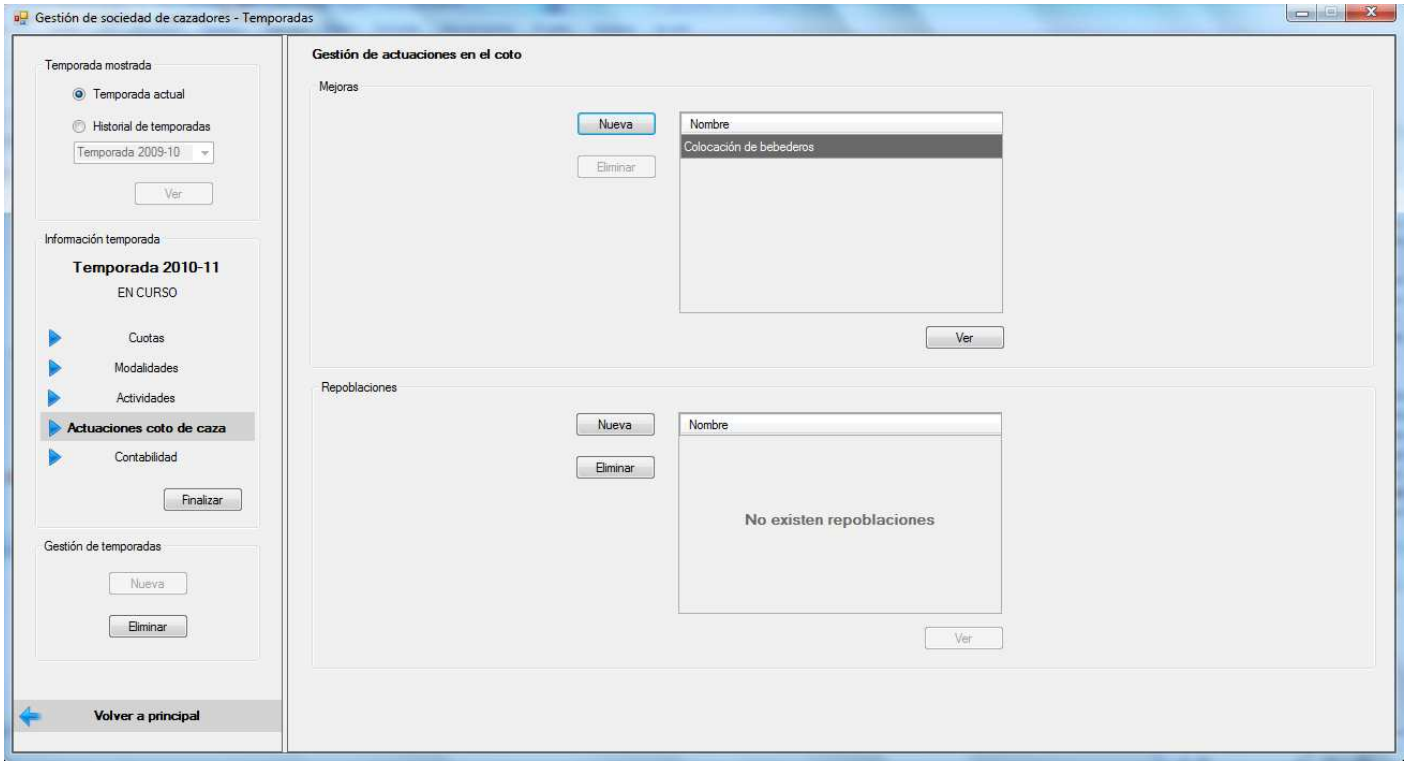

Figura 69. Formulario Temporadas: Actuaciones coto de caza.

Alta de repoblación

Para realizar la operación alta de repoblación en el grupo *Repoblaciones* hacer clic sobre el botón *Nueva*, figura 69. El aspecto de la ventana Temporadas será el siguiente, figura 70.

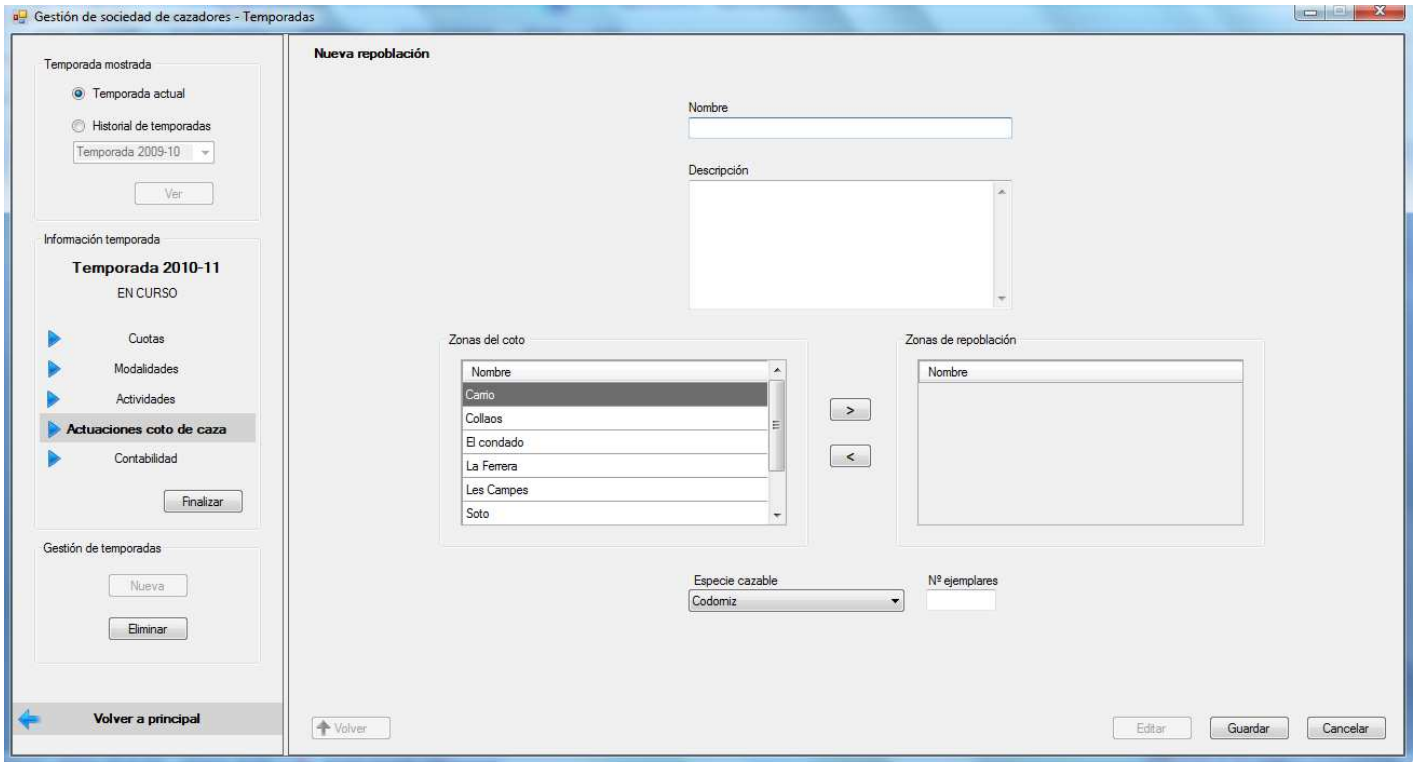

Figura 70. Formulario Temporadas: Nueva repoblación.

Si no se desea seguir con la operación de alta de repoblación hacer clic en el botón *Cancelar* la ventana temporadas volverá a tener el aspecto de la figura 69.

C.3.5.4. Gestión de contabilidad

Para acceder a la gestión de contabilidad en la ventana Temporadas en el grupo *Información temporada* hacer clic sobre el botón *Contabilidad*. El aspecto que tiene la ventana Temporadas es el de la figura 71 en la que se gestiona la contabilidad de la temporada. Se muestran los ingresos y los gastos así como el balance en el que se encuentra.

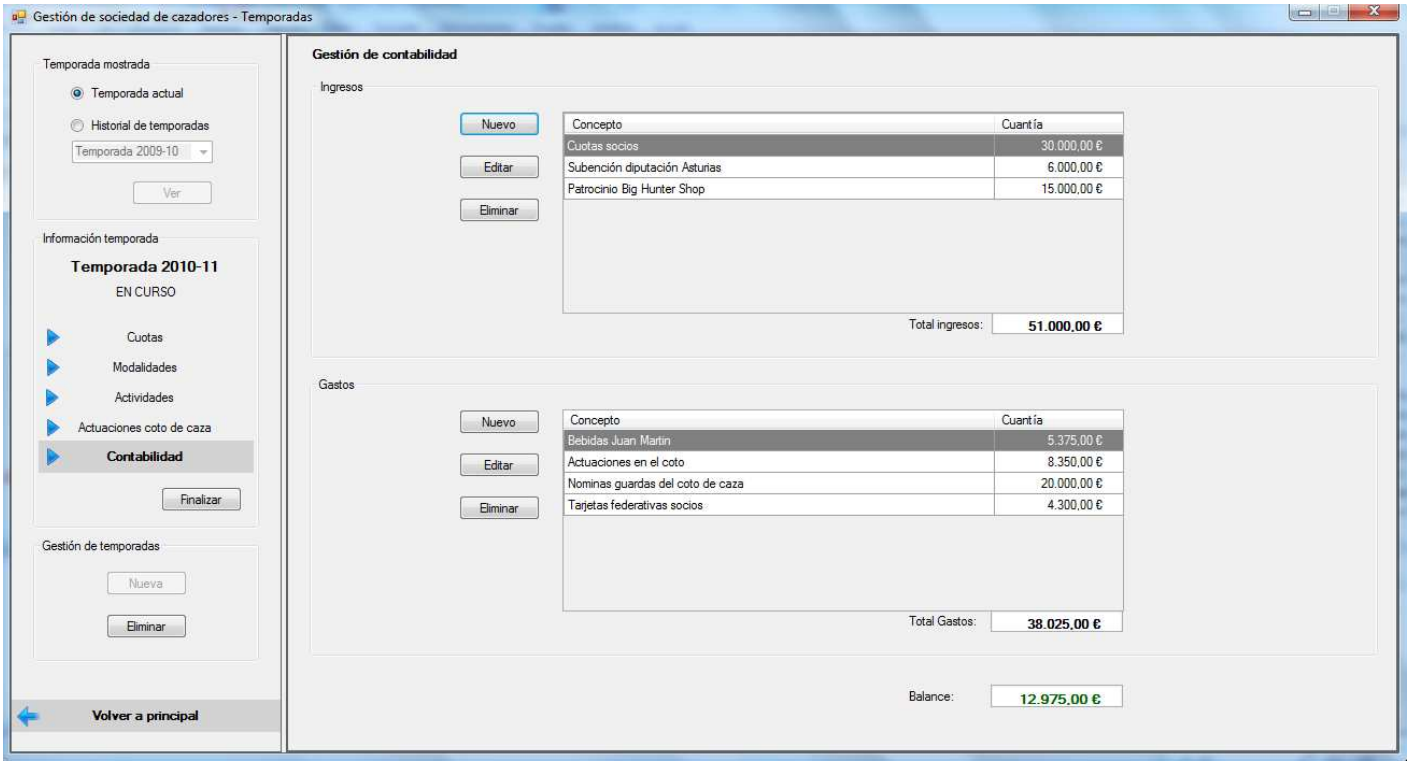

Figura 71. Formulario Temporadas: Contabilidad.

C.3.6. Finalización de temporada

Cuando ya se han terminado de realizar todas las gestiones de una temporada y se quiere realizar la operación finalización de temporada en la ventana Temporadas en el grupo *Información temporada* hacer clic en el botón *Finalizar.*

Si existen socios con la cuota no pagada o sin confirmar no se permitirá realizar dicha operación. Se mostrará un mensaje informando que socio tiene una situación irregular, figura 72.

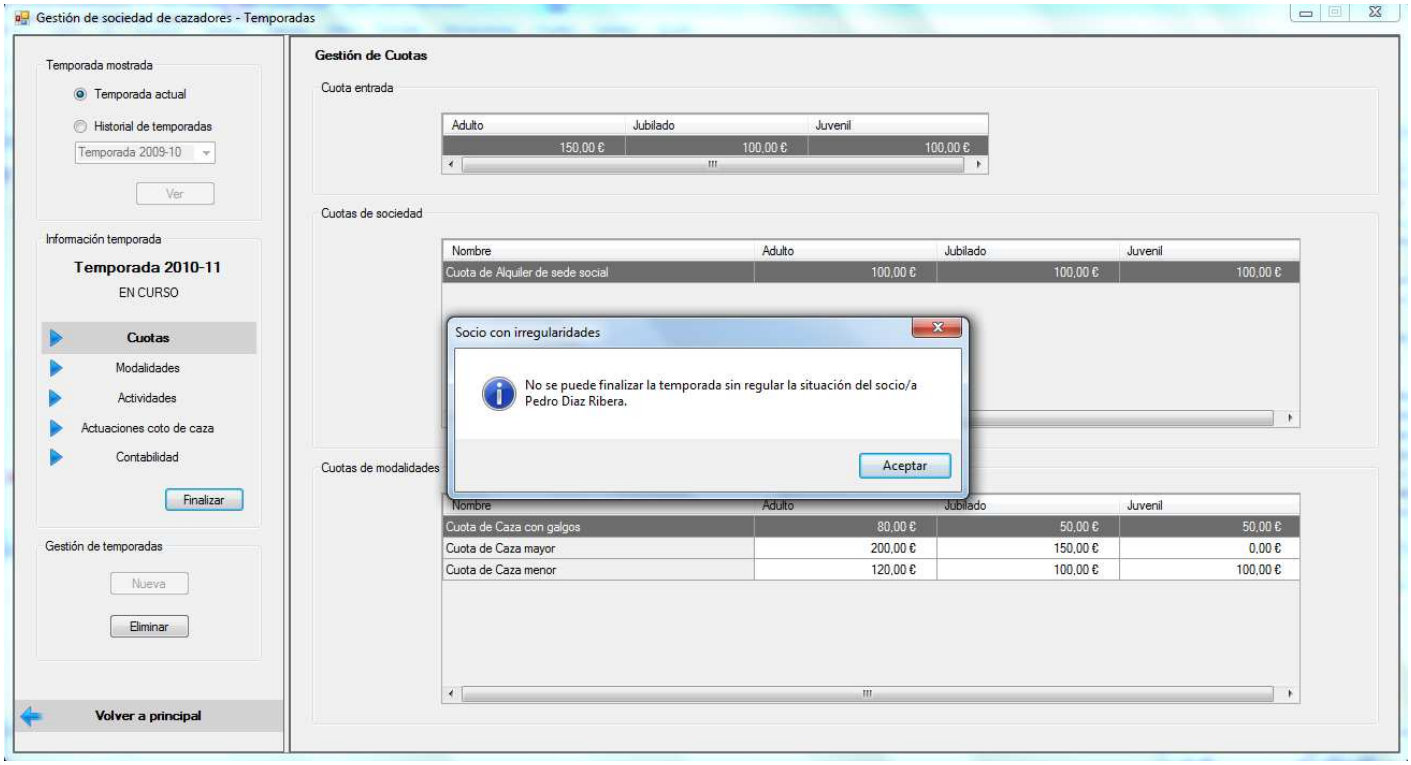

Figura 72. Formulario Temporadas: Finalizar temporada actual.

Para regular la situación de los socios actuales se accederá a la ventana Socios.

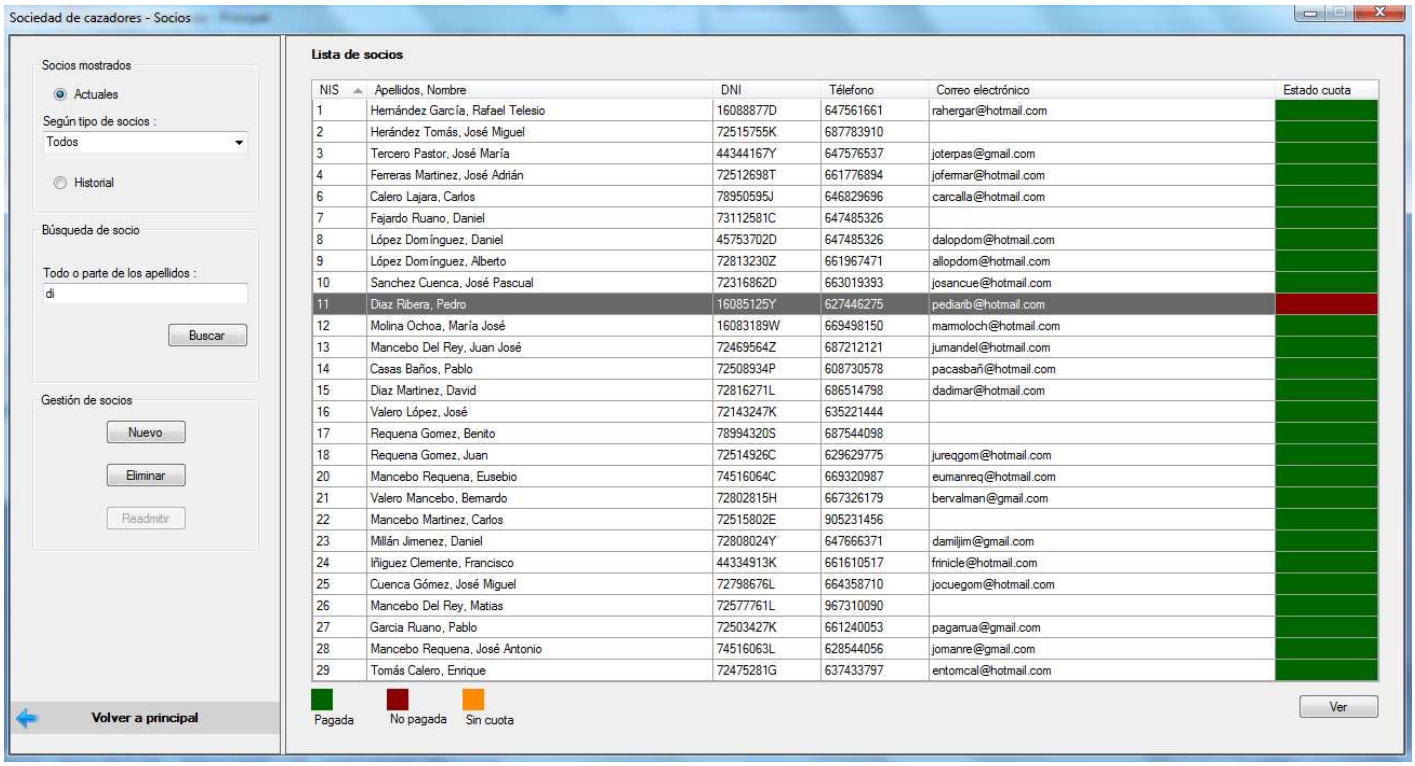

Figura 73. Formulario Socios. Eliminar socio.

Para regular la situación de un socio hay dos opciones:

- Realizar el pago de la cuota o dar al socio.
- Dar de baja al socio que pasará al historial.

Baja de socio

Para realizar la operación baja de socio los pasos a realizar son los siguientes:

1. Seleccionar el socio a dar de baja, en la figura 73 el socio con NIS 11.
2. Hacer clic sobre el botón Eliminar del grupo *Gestión de socios*.
3. Se muestra un mensaje donde se deberá confirmar si se desea dar de baja.
4. Si se confirma la baja del socio, el socio pasará al historial.

Si un socio que fue eliminado, socio antiguo quiere en un futuro formar de nuevo parte de la sociedad pero tiene impagos no podrá ser readmitido hasta que salde su deuda.

C.4. Acceso al historial temporadas

Para acceder a la información de temporadas anteriores a la actual se accederá a la ventana Temporadas.

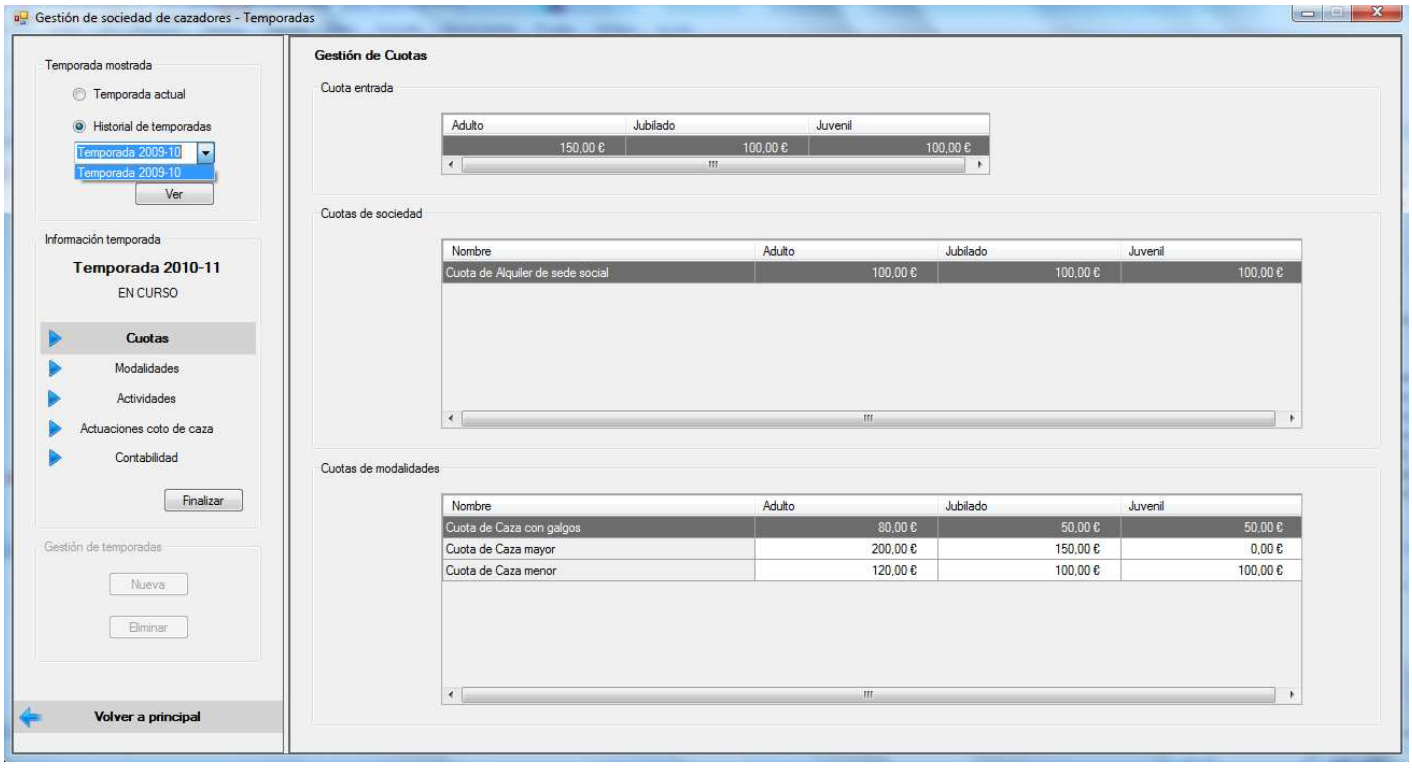

Figura 74. Formulario Temporadas: Acceso a historial.

Para ver la información de una temporada anterior los pasos a seguir son los siguientes pasos:

1. En el grupo *Temporada mostrada* hacer clic sobre el botón *Historial* correspondiente.
2. En el grupo *Temporada mostrada* Seleccionar en la lista despegable la temporada que se desea ver.
3. En el grupo *Temporada mostrada* Hacer clic sobre el botón *Ver*.
4. En el grupo *Información temporada* navegar mediante los botones, los cuales nos permitirán visualizar la información y no modificarla.

Para acceder a la información de juntas directivas anteriores a la actual o a las participaciones del socio en temporadas anteriores los pasos a seguir son similares.