I. Capítulo I: El Proyecto

1.1 Presentación del Proyecto

1.1.1 Título del Proyecto

El título del presente proyecto es: Mejoramiento en la gestión de la Información Sacramental de la Parroquia San Martin de Porres.

1.1.2 Carrera / Unidad

Ingeniería Informática.

1.1.3 Facultad

Ciencias y Tecnologías.

1.1.4 Duración del Proyecto

8 meses.

1.1.5 Área/línea de Investigación Priorizado

Tecnologías de la Información y Comunicación / Desarrollo de Sistemas y Software.

1.1.6 Responsable del Proyecto

Carrera de Ingeniería Informática – Taller III.

1.1.7 Entidad Asociada (s)

Parroquia San Martin de Porres de Tarija.

1.2 Personal Vinculado al Proyecto

1.2.1 Director del Proyecto

Apellido Paterno:	**Apellido Materno:**	**Nombre:**	**C.I.:**
Vedia	Mollo	Maritza Tatiana	7116419 Tja.
Carrera: Ingeniería Informática		**Facultad:** Ciencias y Tecnología	
Telf.: Domicilio: Barrio IV Centenario Calle Nilo Martínez Nº 1210	**Celular:** 72962564	**Correo electrónico:** maritzatvm@gmail.com	**Firma:** ……..

Tabla Nº1. Director del Proyecto

1.2.2 Participantes del Equipo de Trabajo

Categoría	**Nombres y Apellidos**	**Carrera/Profesión**	**C.I.**	**Firma**
Director	Maritza Tatiana Vedia	Ingeniería Informática	7116419	
Asesor	Ing. Fernando Cortez	Lic. en Ingeniería Informática		
Asesor	Ing. Silvana Paz	Ingeniera en Informática		

Tabla Nº2. Participantes del Equipo de Trabajo

1.2.3 Equipo de Trabajo de: Empresas/Instituciones/Organizaciones Participantes/Cooperantes

Nombre: Parroquia San Martin de Porres de Tarija			
Dirección: Barrio Defensores del Chaco		**Teléf. Oficina:** 66-52381	
Nombre y Apellidos	**Cargo**	**C.I.**	**Firma**
Irene Sandoval	Secretario Parroquia San Martin de Porres		

Tabla Nº3. Equipo de Trabajo

1.2.4 Actividades Previstas para los Integrantes del Equipo de Investigación

Responsable *	**Actividades**
Director	Jefe de Proyecto Organizar el equipo de trabajo. Planificar las actividades y controlar del cronograma del proyecto. Asignar y gestionar recursos y prioridades a los distintos componentes y actividades del proyecto. Mantener al proyecto enfocado en los objetivos. Realizar el seguimiento a cada etapa del proyecto. Supervisar el desarrollo del proyecto. Presentación final del sistema. Como Analista de Sistemas: Capturar la especificación y validación de requisitos interactuando con los usuarios mediante entrevistas.

	Elaborar el análisis y diseño del Sistema. Elaborar el modelo de datos (Base de Datos del Sistema). Como Programador: Realizar la Programación del Sistema Informático. Como Ingeniero de Software: Elaborar las pruebas funcionales del Sistema Informático. Como Formador o Capacitador: Formar al personal en el uso de las TIC para el manejo del producto final.
Asesor	Asesoramiento en los aspectos tecnológicos para el desarrollo del Proyecto. Asesoramiento en el uso de la Metodología RUP (Utilización UML). Evaluación del documento del proyecto.

Tabla Nº4. Actividades Previstas para los Integrantes del Equipo de Investigación

1.2.4.1 Unidades de Gestión: Organigrama del Equipo del Proyecto

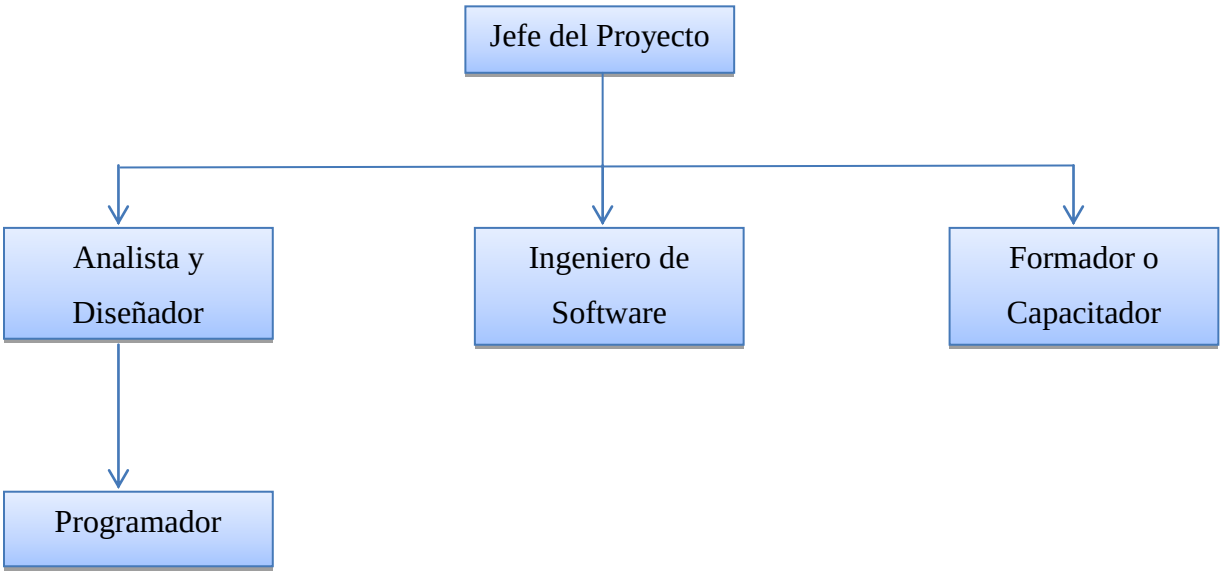


Figura Nº1. Organigrama del Equipo del Proyecto

Jefe de Proyecto: Realiza la planificación y control del cronograma del proyecto, asigna, gestiona recursos y prioridades a los distintos componentes y actividades del proyecto, supervisa el desarrollo del proyecto.

Analista y Diseñador: Capturar la especificación y validación de requisitos interactuando con los usuarios, diseño de los diagramas UML, diseño de la Base de Datos del Sistema.

Ingeniero de Software: Establecer un conjunto de prácticas que aseguren la calidad e integridad del proyecto, realiza la elaboración de pruebas Funcionales del Sistema

Formador o Capacitador: Establecer las estrategias de capacitación, planificar el contenido temático de los cursos de capacitación, formar al personal en el uso para el manejo del producto final.

Programador: Programación del Sistema en base al diseño

1.3 Descripción del Proyecto

1.3.1 Resumen Ejecutivo del Proyecto

Las parroquias de Tarija brindan sacramentos de iniciación cristiana a persona creyentes católicas de esta ciudad sacramentos como: bautismo, comunión, confirmación y matrimonio estos sacramentos son muy importantes para los católicos que son la mayor parte de la ciudad por este motivo las personas los realizan a lo largo de su vida

Al recibir un sacramento la parroquia emite un certificado el cual sirve como prueba de que se recibió dicho sacramento, para recibir otro sacramento y así también sirven para realizar otros trámites.

Por ejemplo el certificado de bautismo sirve a los registros civiles para emitir un certificado de nacimiento o realizar un trámite de modificar algunos datos del mismo.

Debido a esto es muy importante la emisión de un certificado como también almacenar los datos de las personas que reciben dichos sacramentos, la parroquia debe almacenar los datos de forma segura y emitir los mismos en el momento que la población los necesite.

Hoy en día las parroquias no cuentan con un sistema automatizado que les facilité la administración de esta información, las parroquias registran sus datos de forma manual en libros esta manera de almacenarlos no es segura y requiere mucho tiempo a la hora de emitir información a los feligreses.

El presente proyecto pretende mejorar la gestión de la Información Sacramental de la parroquia San Martin de Porres a través de un sistema que nos permita brindar información oportuna y de calidad.

Para lograr el propósito se tienen los siguientes componentes

"SIGES" Sistema Informático de Gestión Sacramental.

Capacitación sobre el uso del sistema.

Sistema Informático de Gestión Sacramental: Al tener un registro automatizado facilitara el trabajo del secretario de la parroquia a la hora de emitir un certificado el tiempo de búsqueda de los datos disminuiría mucho con relación al tiempo que se necesita ahora sin un sistema, los fieles podrán disponer de la información confiable y oportuna en el momento que la soliciten.

Capacitación sobre el uso del sistema: Después de la implementación del sistema de información, se procederá a instruir al personal responsable de la parroquia para la administración del mismo.

La parroquia San Martin de Porres de Tarija es la que nos ha brindado la información requerida para realizar este trabajo.

1.3.2 Descripción, Fundamentación y Justificación del Proyecto

En las parroquias existe gran información de los sacramentos brindados por las mismas, información que es muy importante para los feligreses a la hora de recibir otro sacramento o para realizar un trámite personal, debido a esto las parroquias deben tener los registros almacenados para brindar a los feligreses en el momento que estos los necesiten.

Los registros son manejados en forma manual en libros; esta manera de almacenar los registros genera los siguientes problemas:

- Información no confiable e insegura para brindar a los feligreses debido a que los datos son almacenados en libros existe el peligro que con el pasar de los años la información está dañada.
- Peligro de perder un libro que contiene grandes cantidades de información muy importante para los feligreses
- Demora en la obtención de información, debido a que los datos son registrados de manera manual, buscar información toma mucho tiempo, ya que se debe buscar los datos en libros
- Demora en la obtención y reposición de un certificado
- Información mal clasificada, los datos están desordenados no tienen ningún orden o categoría
- No contar con información que nos permita tomar decisiones.
- No presentar la documentación requerida para realizar un trámite de cambio de algún dato personal, debido a la perdida de la información.
- No se cuenta con información oportuna para los feligreses para recibir otro

sacramento debido al tiempo que toma encontrar esa información.

- Necesidad de presentar el certificado de un sacramento para poder recibir otro sacramento.

1.3.2.1 Análisis de Causas del Problemas

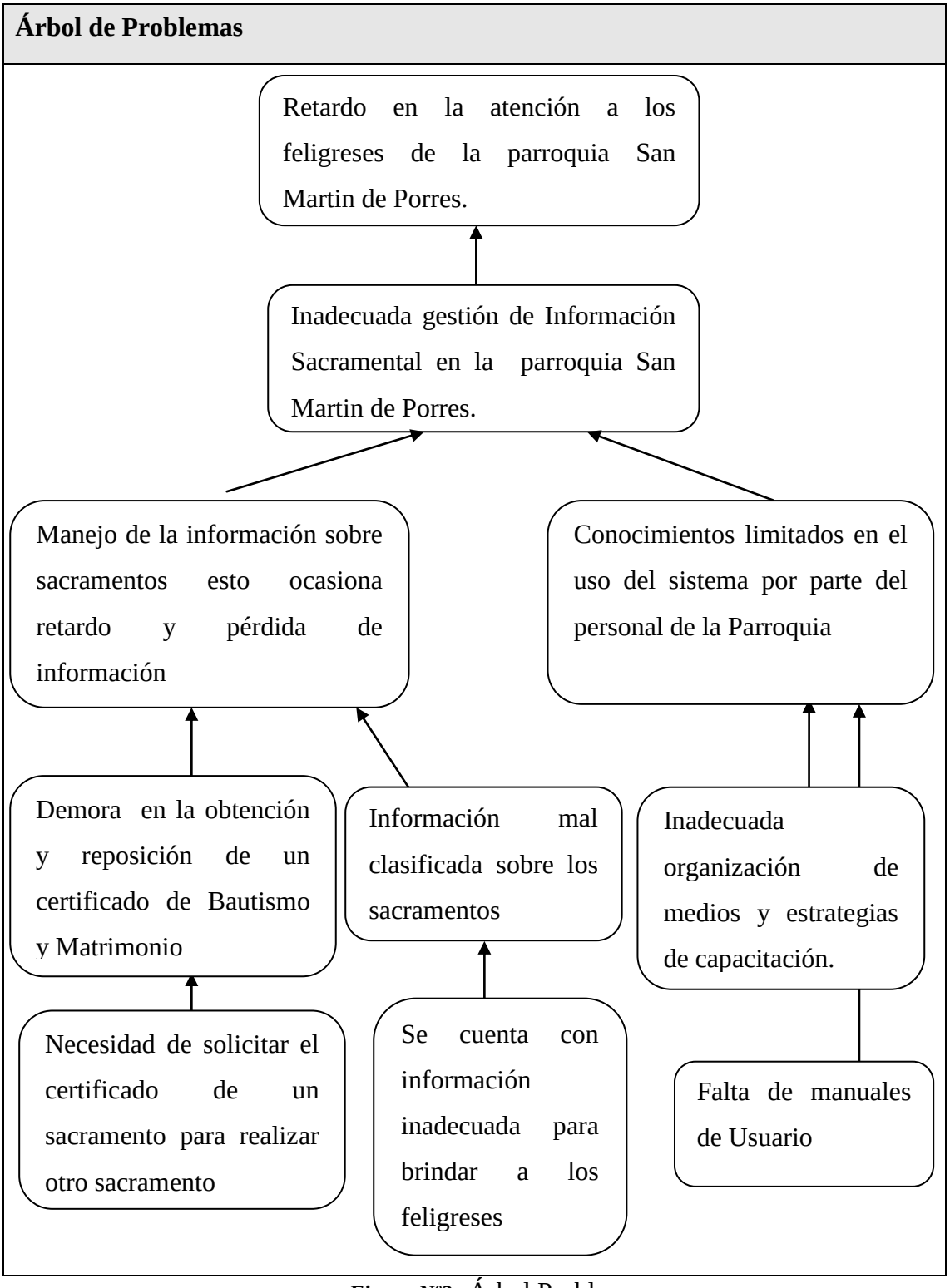


Figura Nº2. Árbol Problema

1.3.2.2 Análisis de Objetivos

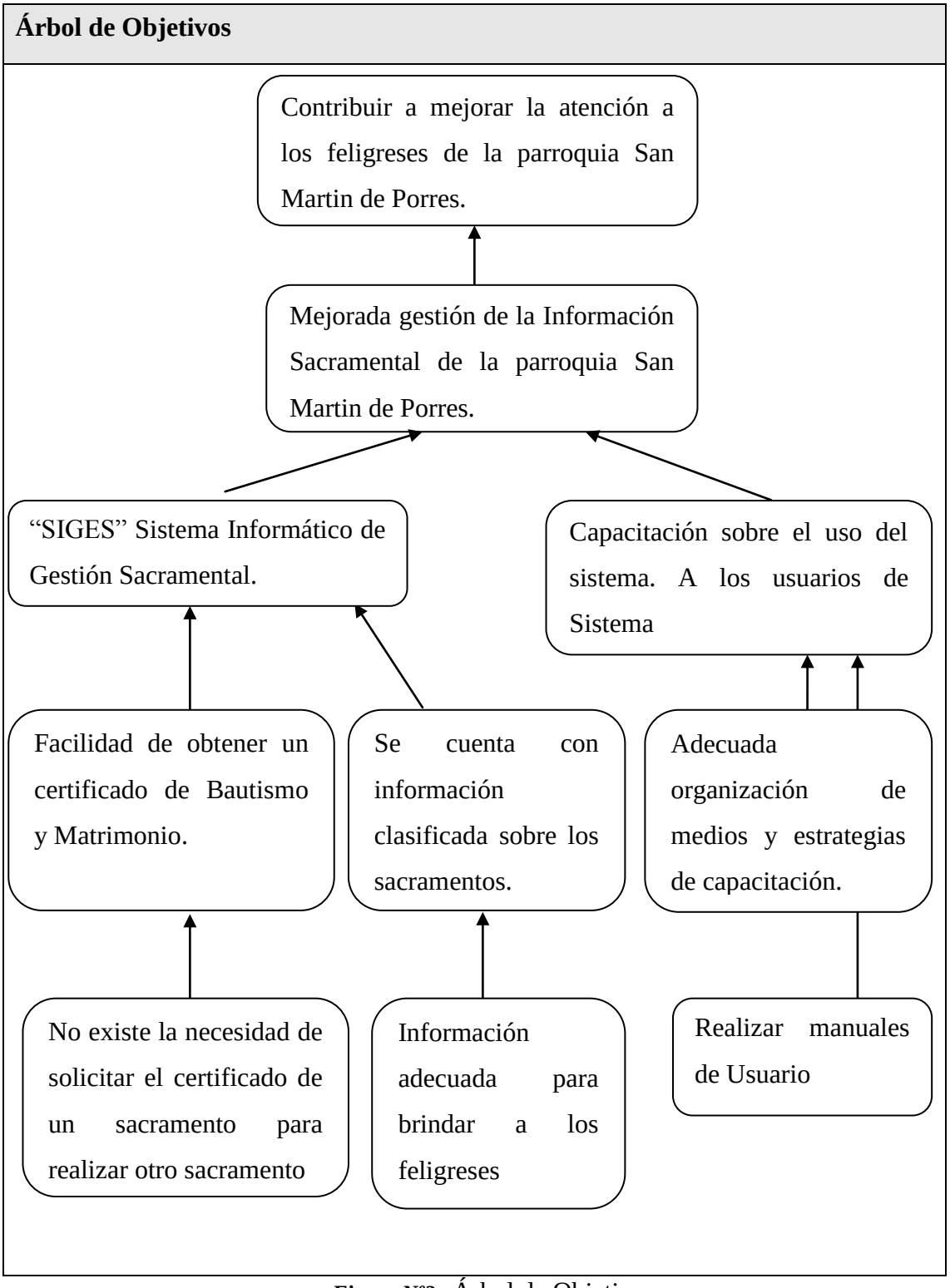


Figura Nº3. Árbol de Objetivos

1.3.3 Objetivos

1.3.3.1 Objetivo General

Mejorar la Gestión de la Información Sacramental de la Parroquia San Martin de Porres.

1.3.3.2 Objetivos Específicos

- "SIGES" Sistema Informático de Gestión Sacramental.
- Capacitación del Personal de la Parroquia San Martin de Porres de Tarija, en el Sistema Informático desarrollado.

1.3.4 Metodología

La forma de trabajo de este proyecto estará basada en:

1.3.4.1 Metodología para el desarrollo de las aplicaciones

Se utilizará la metodología **RUP (Racional Unified Process)**, que mejora considerablemente la calidad de desarrollo del sistema, ya que la misma utiliza el Lenguaje Unificado de Modelado (UML) para preparar todos los esquemas de un sistema software [1].

RUP es un proceso ágil de desarrollo que se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los clientes.

El flujo de trabajo fundamental tiene los siguientes pasos:

- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Programación e Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Requerimientos: En base a las entrevistas se obtendrá la información que refleje las necesidades de los involucrados para la determinación de requerimientos.

Análisis y diseño: En base a la determinación de requerimientos, se estructurará las diferentes vistas (Diagramas, base de datos, Pantallas) de la aplicación, tomando en cuenta metodologías de desarrollo de software.

Programación e Implementación: La programación será modular y orientada a objetos, se utilizarán tecnologías de punta contando con el apoyo de programadores novatos (estudiantes), creando la aplicación informática que tenga el comportamiento deseado.

Pruebas: Antes de desarrollar las pruebas se procederá a la introducción de datos. Introducida esta información al sistema se dará inicio a la fase de pruebas de desarrollo que serán mediante casos de prueba tomados de cada módulo y se realizarán los ajustes necesarios para una correcta validación.

Este proceso se torna repetitivo si se detectan inconsistencias en el sistema implicando el retorno de cualquiera de las fases anteriores para su corrección. [4]

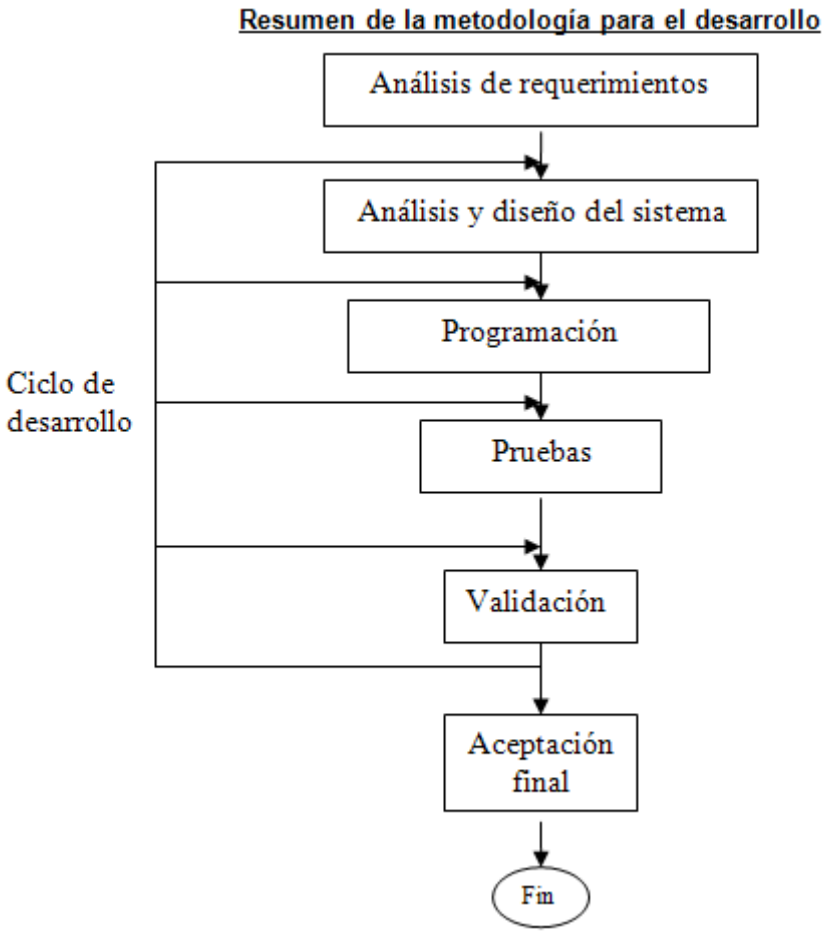


Figura Nº4. Resumen de la Metodología para el Desarrollo

1.3.4.2 Metodología para las Capacitaciones

La metodología utilizada para esta capacitación se basa en el modelo de experiencia "Aprender Haciendo", logrando de esta manera la motivación de ser constructores de su propio conocimiento. Involucra trabajos de completa actividad en Talleres de construcción de contenidos, Laboratorios, Guías, donde se incentiva el trabajo creativo y práctico, experimentándose con procesos o fenómenos a partir de ideas o propuestas teóricas previamente formuladas.

Es una metodología busca desarrollar su sensibilidad frente a problemas reales, estudiar alternativas de solución y evaluar sus implicancias en conjunto con la utilización de tecnologías [7].

Estrategias Didácticas

Descubriendo la Tecnología

Tipo: Sesiones Individuales.

Descripción: Se desarrolla actividades guiadas donde el capacitador hace uso del material guía y van descubriendo por sí solos las potencialidades de los programas y herramientas y que luego ellos mismos documentan haciendo énfasis en cuáles de las experiencias descubiertas son de principal apoyo para su práctica pedagógica.

Aspectos Prácticos

Actividad teórico- práctica: Propicia la modalidad del curso teórico con una actividad de la práctica en relación a la temática de estudio. Lo teórico y lo práctico se dan simultáneamente en forma conjunta e interrelacionada y es dirigida directamente por el capacitador participante.

Mecanismos de Trabajo durante la capacitación:

Fase Inicial:

Análisis de la Realidad Social de la Región: donde los Facilitadores intercambian diferentes criterios con los docentes logrando tener una visión clara de las necesidades pedagógicas de los participantes enfocando la capacitación a la presentación de temas individuales.

Diagnóstico de Iniciación: donde se recolecta información mediante evaluaciones de conocimientos que nos permitirá medir el nivel que poseen los participantes en la utilización de las aplicaciones.

Motivación y Sensibilidad: donde los Facilitadores realizan dinámicas para Motivar y sensibilizar sobre la importancia de desarrollar capacidades en el manejo de las TIC’s, y de esta manera los capacitadores puedan aplicar la herramienta y aportar en su conjunto al desarrollo de su comunidad.

Fase de Capacitación:

Investigación y documentación: se promueve en todas las estrategias didácticas aplicadas a la investigación continua y descubrimiento de la tecnología reconociendo su valor e identificando su potencial para ser utilizados en su preparación de material, todo esto desarrollado por el capacitador.

1.3.5 Resultados esperados

Sistema Informático desarrollado probado; que integre de manera eficaz la lógica de procesamiento de información de la sección involucrada y presentado al finalizar la ejecución del proyecto.

Al finalizar el proyecto al menos un 80% de los participantes el secretario y administrador del sistema han sido capacitados.

Ayudar a mejorar la atención en la sección de afiliación, para las empresas empleadoras y sus empleados.

Crear una nueva conciencia tecnológica en el personal de Parroquia San Martin de Porres de Tarija, respecto al gran beneficio que proporcionan.

1.3.6 Transferencia de Resultado

1.3.6.1 Medios y Estrategias para la Transferencia de Resultados

Se realizara el análisis de requerimientos con ayuda del secretario de la parroquia con el motivo de que el estudiante obtenga la información necesaria y oportuna para la realización de este proyecto, además de otros recursos que la organización pueda proporcionar para aminorar la carga económica del desarrollo del proyecto.

Asimism el Departamento de Informática y Sistemas de la Facultad de Ciencias y Tecnología de la Universidad Autónoma Juan Misael Saracho de Tarija se compromete a realizar el seguimiento y control del proyecto y a ceder el uso del sistema informático a la institución para la explotación del mismo.

1.3.6.2 Grupo de Beneficiarios de los Resultados

- Personas (niños, jóvenes, adultos) que necesiten obtener un certificado de algún sacramento realizado en la parroquia
- Parroquia San Martin de Porres de Tarija puede brindar información de manera más confiable a los feligreses.
- El párroco de la Parroquia San Martin de Porres obtiene la información de manera más rápida y le ayuda a tomar decisiones

1.3.7 Cronograma de Actividades

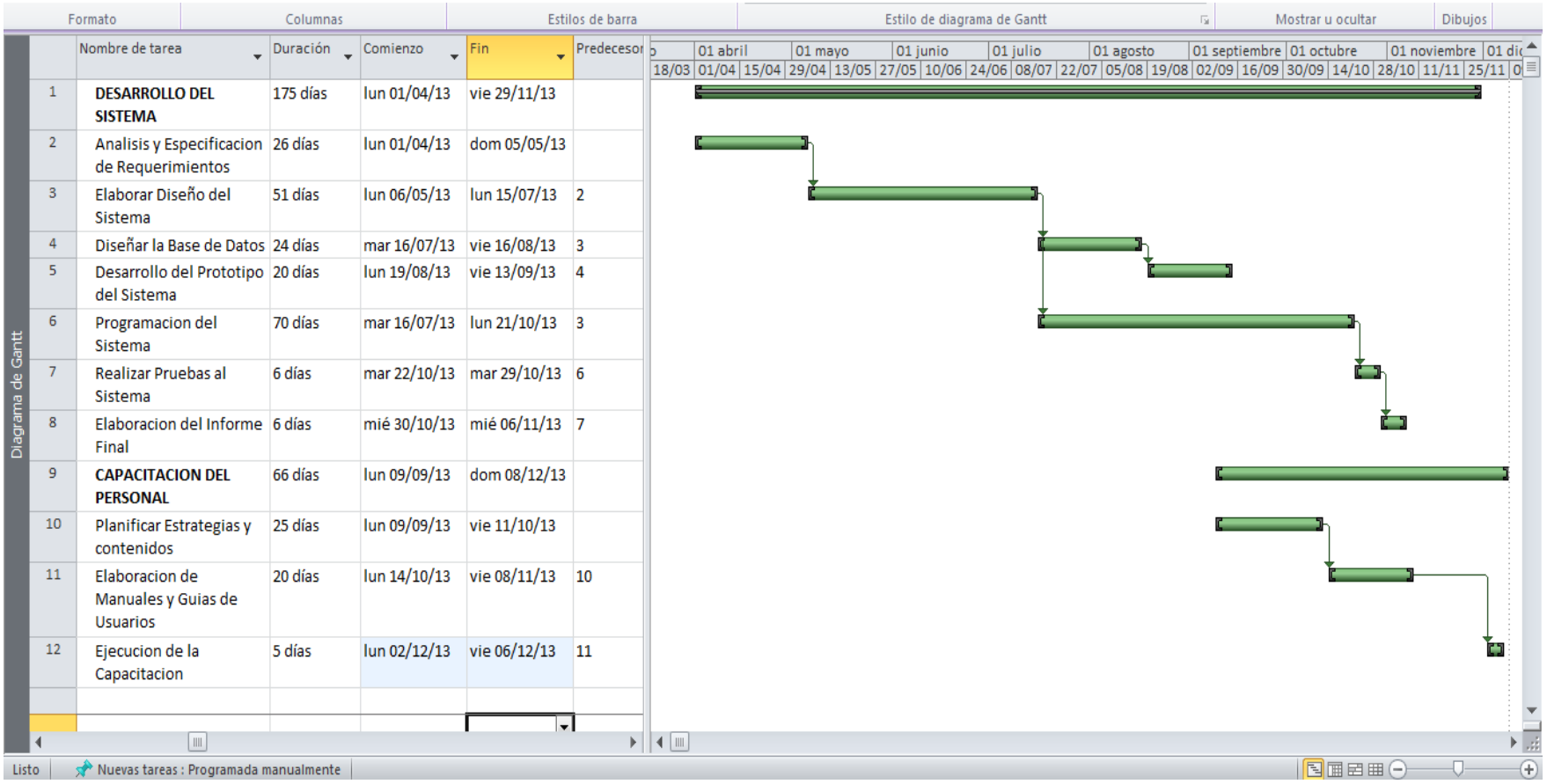


Figura N°5. Cronograma de Actividades

1.3.8 Marco Lógico del Proyecto

Resumen Narrativo del Proyecto	Indicadores	Medios de Verificación	Supuestos
Fin Contribuir a mejorar la atención a los feligreses de la parroquia San Martin de Porres brindando un servicio de calidad en consulta de información requerida.			
Objetivo General (Propósito) Mejorar la gestión de la Información Sacramental de la parroquia San Martin de Porres.	Al finalizar el proyecto se cuenta con información acerca de los feligreses que recibieron algún sacramento. A partir del 2014 el tiempo de respuesta a las consultas de los fieles ha reducido de 2 has. A 15 min. Categoría Calidad	Informe por parte de los docentes de Taller III.	Información confiable y oportuna a la población

Objetivos Específicos (Componentes) **C1.-**"SIGES" Sistema Informático de Gestión Sacramental. **C2.-Capacitacion sobre el uso del sistema.**	**C1.-**Al finalizar el proyecto se ha desarrollado un 100 % del Sistema Informático de Gestión Sacramental realizados en la parroquia, de acuerdo a sus requerimientos. **C2.-** Al finalizar el proyecto se ha capacitado un 90% al personal encargado del manejo y actualización del sistema en la parroquia	✓ Documento de Diseño del sistema aprobado por los docentes de Taller 3 ✓ Informe por parte de la parroquia San Martin de Porres. ✓ Fotografías de la capacitación.	✓ Predisposición por parte de la secretaria de la parroquia para brindar información sobre los problemas y los procesos que realizan para registrar un sacramento. ✓ La capacitación se lleva a cabo en la fecha establecida en el calendario.

Actividades **C1: Sistema registros de sacramentos realizados en una parroquia.** **1.** Llevar a cabo el análisis y especificación de requerimientos. **2.** Elaborar el diseño del Sistema. **3.** Diseñar la base de datos del Sistema. **4.** Desarrollar el Prototipo del	**C1: Sistema Informático Total Bs. 990** ✓ 21100 Comunicación Bs. 150 ✓ 21400 Servicios Teléf. Bs. 340 ✓ 25600 Imprenta Bs. 400 ✓ 32100 Papel de Escritorio Bs. 100	✓ Informe de la ejecución presupuestaria por parte del Director del proyecto ✓ Sistema web listo para ser utilizado	✓ Se cuenta con la información que nos proporcionan las parroquias ✓ Sistema Informático desarrollado sin retrasos ✓ Se cuenta con la disponibilidad de los recursos económicos.

Sistema. **5.** Programación del Sistema en base a patrones de diseño. **6.** Elaborar e implementar pruebas para la validación del sistema. **7.** Elaborar el informe final sobre el Desarrollo del sistema. **C2: Capacitación sobre el uso del sistema** **8.** Planificar el contenido temático de la capacitación. **9.** Capacitar al personal encargado para el manejo y la actualización del sistema.	**C2: Capacitación al Personal Total Bs. 150** 31110 Refrigerios Bs. 150	✓ Manual de Usuario e Instalación impresos	✓ Interés y disponibilidad del personal para ser capacitados en la fecha prevista.

Tabla Nº5. Matriz de Marco Lógico

1.4 Presupuesto / Justificación

ITEM	RUBROS	Aporte Universidad	Otro Aporte	TOTAL (Bs.)
10000	**SERVICIOS PERSONALES**			
	12000 Empleados no Permanentes			0
	Sub total rubro			0
20000	**SERVICIOS NO PERSONALES**			
	21000. Servicios Básicos	0	900	900
	22000. Servicios de transporte			0
	23000. Alquileres			0
	24000. Mantenimiento y reparación			0
	25000. Servicios Profesionales y Comerciales	0	650	650
	Sub total rubro			1550
30000	**MATERIALES Y SUMINISTROS**			
	31000. Alimentos y Productos Forestales	0	150	150
	32000. Productos de Papel, Cartón e Impresos	0	175	175
	33000. Textiles y Vestuario.			0
	34000. Productos Químicos, Combustibles y Lubricantes			0
	39000. Productos Varios.			0
	Sub total rubro			325

40000	**ACTIVOS REALES**			
	43000. Maquinaria y Equipo.			0
	46000. Descripción de estudios y proyectos para inversión			0
	49000. Otros Activos			0
	Sub total rubro			0
	TOTAL			1875
	TOTAL + 40% Incentivo			2625

- **GRUPO 20000. SERVICIOS NO PERSONALES**

  a) SUB GRUPO 21000. Descripción de los gastos de servicios básicos

Partida	**Tipo de servicio básico ***	**Costo**	**Tiempo mes**	**Costo Total**
21100	Comunicación	20	6	120
21200	Energía Eléctrica	8	60	480
21300	Agua			0
21400	Servicios Telefónicos	50	6	300
Total				900

* Se refiere principalmente a los gastos por servicios; como: servicio de correo, radiogramas, servicio telefónico, fax, Internet.

b) SUB GRUPO 25000. Descripción de los gastos en servicios profesionales y comerciales

Partida	Tipo de servicio profesional y comercial *	Cantidad	Costo unitario	Tiempo mes	Costo total
25200	Estudios e Investigaciones				0
25500	Publicidad				0
25600	Imprenta	3000	0.2	8	600
25700	Capacitación de Personal	2	25	1	50
25800	Estudios e Investigaciones Para Proyectos de Inversión				0
Total					650

* Se refiere a gastos por servicios profesionales de asesoramiento especializado, se incluyen, estudios, investigaciones, publicidad, imprenta, fotocopias, capacitación de personal y otros ejecutados por terceros.

- **GRUPO 30000. MATERIALES Y SUMINISTROS**

  c) SUB GRUPO 31000. Descripción de los gastos Alimentos y Productos Agroforestales

Partida	Tipo de material *	Cantidad	Costo/Unitario	Total
31110	Refrigerios y Gastos Administrativos	10	15	150
31200	Alimento para Animales			0
31300	Productos Agroforestales y Pecuarios			0
Total				150

* Se refiere a la adquisición de materiales y bienes como: alimentos y productos agroforestales, alimentos y bebidas para personas (indicar el total de refrigerios), alimentos para animales, productos pecuarios.

d) SUB GRUPO 32000. Descripción del gasto de Productos de Papel, Cartón e Impresos

Partida	Tipo de material *	Cantidad	Costo/Unitario	Total
32100	Papel de Escritorio	5	35	175
32200	Productos de Artes Gráficas, Papel y Cartón			
32300	Libros y Revistas			
32400	Textos de Enseñanza			
32500	Periódicos			
Total				175

* Se refiere a la adquisición de; papel y cartón en sus diversas formas y clases, impresos y publicaciones, periódicos, revistas, libros, fotocopias, etc.

1.4.1. **Curriculum Vitae** (del Director y Equipo de Trabajo: llenar una ficha para cada uno de ellos)

1.4.1.1. **Antecedentes personales**

Vedia **Apellido Paterno**	Mollo **Apellido Materno**	Maritza Tatiana **Nombre**	7116419 **C.I**.
11/04/1987 **Fecha de nacimiento**	Femenino **Sexo**	B. IV Centenario C/ Nilo Martínez Nº 1210 **Dirección**	
Tarija **Ciudad**	66- 54255 **Teléfono Domicilio**	72962564 **Celular**	maritzatvm@gmail.com **Correo electrónico**

II. Capítulo 2: Contexto

2.1 Componente 1: “SIGES” Sistema Informático de Gestión Sacramental.

2.1.1 Marco Teórico

2.1.1.1 Antecedentes

Desde el surgimiento de la teoría de la organización, la función esencial de la información en las organizaciones se ha acentuado. Una organización es un sistema compuesto por tres elementos: personas, materiales e información. Los sistemas de información, por su parte, surgen como sistemas complejos y abiertos que interactúan con otros sistemas y subsistemas como parte de su actuación. Por los años 90, una de las concepciones más defendidas por la gestión de la información fue que las organizaciones son sistemas de información.

El uso de ciertos conceptos tomados de la teoría de sistemas y del campo de la informática llevó a un alto grado de desarrollo entre los sistemas de información. Aunque existen diversas definiciones, hechas desde diferentes enfoques, sobre los sistemas de información, en su gran mayoría tienen puntos en común. El análisis realizado sobre las definiciones más frecuentes efectuadas en la década de los años 90 revela que constituyen un conjunto integrado de procesos, elementos o componentes que –según las estrategias y necesidades de una organización– recopilan, elaboran y distribuyen la información necesaria.

Un sistema moderno de gestión de información exige la aplicación de nuevas tecnologías de información; sin embargo, la tecnología por sí sola no es suficiente para lograr una buena gestión de información. Son diversos los procesos que conforman los sistemas de gestión de información; ellos generan las entradas y salidas del sistema o de otros procesos relacionados; también pueden identificarse, controlarse, corregirse o actualizarse en la medida en que se producen las transformaciones del entorno y evoluciona la organización, como vía incuestionable para garantizar su calidad, eficiencia y mejora continua.

A modo de resumen de este antecedente de marco teórico, puede decirse que los sistemas de gestión de información, en su definición más general, se refieren al

conjunto de todos los componentes necesarios que se interrelacionan, con el objetivo de tramitar y facilitar la información sobre el tema de interés para su consumo en cualquier medio, momento y lugar.

2.1.1.2 Metodología de Desarrollo

2.1.1.2.1 Metodología RUP(Racional Unified Process)

a) Definición: RUP es un marco del proyecto que describe una clase de los procesos que son iterativos e incrementales. Es un proceso de Ingeniería de Software que captura las mejores prácticas del conocimiento de líderes en Ingeniería de Software y que provee un enfoque para asignar tareas y responsabilidades dentro de una organización de desarrollo.

Los **procesos de RUP** estiman tareas y horario del plan midiendo la velocidad de iteraciones concerniente a sus estimaciones originales. Las iteraciones tempranas de proyectos conducidos por RUP se enfocan fuertemente sobre arquitectura del software; la puesta en práctica rápida de características se retrasa hasta que se ha identificado y se ha probado una arquitectura firme.

Nos permite realizar un levantamiento exhaustivo de requerimientos.

Las actividades de RUP se centran en crear y mantener modelos, utilizando UML, en forma efectiva.

- Busca detectar defectos en las fases iníciales.
- Intenta reducir al número de cambios tanto como sea posible.
- Realiza el Análisis y Diseño, tan completo como sea posible.
- Diseño genérico, intenta anticiparse a futuras necesidades.
- Las necesidades de clientes no son fáciles de discernir.
- Existe un contrato prefijado con los clientes.
- El cliente interactúa con el equipo de desarrollo mediante reuniones.

b) Características esenciales.- Los autores de RUP destacan que el proceso de software propuesto por RUP tiene tres características esenciales: está dirigido por los Casos de Uso, está centrado en la arquitectura, y es iterativo e incremental.

- **Está dirigido por los Casos de Uso:** Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema.

  Los Casos de Uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.

  En RUP los Casos de Uso son una herramienta para especificar los requisitos del sistema.

  En el proyecto el modelo de Casos de Uso presenta las funciones del sistema y los actores que lo utilizan. Todo esto mediante los Diagramas de Casos de Uso.

- **Está Centrado en su Arquitectura:** La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.

  La arquitectura involucra los aspectos estáticos y dinámicos más significativos del sistema, está relacionada con la toma de decisiones que indican cómo tiene que ser construido el sistema y ayuda a determinar en qué orden. Además la definición de la arquitectura debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso de desarrollo. La arquitectura se ve influenciada por la plataforma software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados. Muchas de estas restricciones constituyen requisitos no funcionales del

sistema.

En el caso de RUP además de utilizar los Casos de Uso para guiar el proceso se presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento.

Cada producto tiene tanto una función como una forma. La función corresponde a la funcionalidad reflejada en los Casos de Uso y la forma la proporciona la arquitectura. Existe una interacción entre los Casos de Uso y la arquitectura, los Casos de Uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los Casos de Uso requeridos, actualmente y en el futuro. Esto provoca que tanto arquitectura como Casos de Uso deban evolucionar en paralelo durante todo el proceso de desarrollo de software.

- **Es Iterativo e Incremental:** Según el equilibrio correcto entre los Casos de Uso y la Arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo. Para esto, la estrategia que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre Casos de Uso y Arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto.

  Una iteración puede realizarse por medio de una cascada. Se pasa por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas), también existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores.

c) Fases en el ciclo de Desarrollo.- Este proceso de desarrollo considera que cualquier desarrollo de un sistema software debe pasar por cuatro fases que se describirán a continuación, la figura muestra las Fases de desarrollo y los diversos flujos de trabajo involucrados dentro de cada fase con una representación gráfica en cuál de los flujos se hace mayor énfasis según la fase, cabe destacar el flujo de trabajo concerniente al negocio.

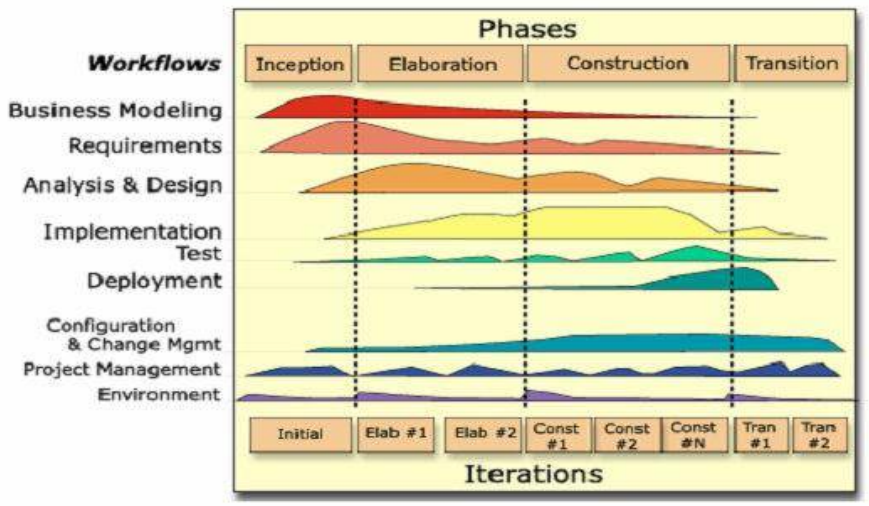


Figura Nº6. Fases en el Ciclo de Desarrollo de R.U.P.

- **Fase 1: Preparación Inicial ("Incepción")**

  Su objetivo principal es establecer los objetivos para el ciclo de vida del producto. En esta fase se establece el caso del negocio con el fin de delimitar el alcance del sistema, saber qué se cubrirá y delimitar el alcance del proyecto.

  El caso de negocio incluye criterios de éxito, la evaluación de riesgos, y la estimación de los recursos necesarios, y un plan de la fase que muestre las fechas previstas e hitos importantes.

- **Fase 2: Preparación Detallada ("Elaboración")**

  Su objetivo principal es plantear la arquitectura para el ciclo de vida del producto. En esta fase se realiza la captura de la mayor parte de los requerimientos funcionales, manejando los riesgos que interfieran con los objetivos del sistema, acumulando la información necesaria para el plan de

construcción y obteniendo suficiente información para hacer realizable el caso del negocio.

El resultado de la fase de elaboración es:

Un modelo de caso de uso (por lo menos 80% completo) - todos los casos de uso y actores deben haber sido identificados-, y se han desarrollado la mayoría de las descripciones de casos de uso.

Requerimientos suplementarios que capturan los requerimientos no funcionales o cualquier requerimiento que no se asocie a un caso de uso específico.

- **Fase 3: Construcción ("Construcción")**

  Su objetivo principal es alcanzar la capacidad operacional del producto. En esta fase a través de sucesivas iteraciones e incrementos se desarrolla un producto software, listo para operar, éste es frecuentemente llamado versión beta.

- **Fase 4: Transición ("Transición")**

Su objetivo principal es realizar la entrega del producto operando, una vez realizadas las pruebas de aceptación por un grupo especial de usuarios y habiendo efectuado los ajustes y correcciones que sean requeridos.

Éste incluye:

- Operación en paralelo con un sistema anterior que el nuevo sistema esté sustituyendo.
- La conversión de las bases de datos operacionales.
- Entrenamientos y capacitación de los usuarios y la gente de mantenimiento.

2.1.1.2.2 UML (Lenguaje Unificado de Modelado).

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema [5].

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.

Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las Metodologías de desarrollo.

a) Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Aunque UML está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo (workflow) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

b) Un modelo UML está compuesto por tres clases de bloques de construcción:

- Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.).
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones.

c) UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

d) UML ofrece notación y semántica estándar:

UML prescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos. Previamente, un diseño orientado a objetos podría haber sido modelado con cualquiera de la docena de metodologías populares, causando a los revisores tener que aprender las semánticas y notaciones de la metodología empleada antes que intentar entender el diseño en sí.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

En UML hay 13 tipos diferentes de diagramas. Para comprenderlos de manera concreta, es útil categorizarlos jerárquicamente.

Los ***Diagramas de Estructura*** enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases, representan la estructura estática en términos de clases y relaciones.
- Diagrama de componentes, representan los componentes físicos de una aplicación.
- Diagrama de objetos, representan los objetos y sus relaciones, corresponden a diagramas de colaboración simplificados sin la representación del envío de

mensajes.

- Diagrama de estructura compuesta.
- Diagrama de despliegue, representan el despliegue de los componentes sobre los dispositivos físicos.
- Diagrama de paquetes, muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones.

Los ***Diagramas de Comportamiento*** enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades, representan el comportamiento del sistema en términos de acciones.
- Diagrama de casos de uso, representan funcionalidad del sistema desde el punto de vista del usuario.
- Diagrama de estados, representan el comportamiento de una clase en término de estados.

Los ***Diagramas de Interacción*** son un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia, son una representación temporal de los objetos y sus interacciones.
- Diagrama de comunicación, que es una versión simplificada del Diagrama de colaboración, son una representación espacial de los objetos, enlaces e interacciones entre ellos.
- Diagrama de tiempos.
- Diagrama global de interacciones o Diagrama de vista de interacción.

2.1.1.2.2.1 Tipos de Diagramas Utilizados

2.1.1.2.2.1.1 Diagramas 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 encargarán del funcionamiento y la relación entre uno y otro.

2.1.1.2.2.1.2 Diagramas de Casos de Uso

Que captura las interacciones de los casos de uso y los actores. Describe los requisitos funcionales del sistema, la forma en la que las cosas externas (actores) interactúan a través del límite del sistema y la respuesta del sistema.

2.1.1.2.2.1.3 Diagramas de Actividades

Que se usa para modelar el comportamiento de un sistema, y la manera en que este comportamiento está relacionado con un flujo global del sistema. Se usan los caminos lógicos que sigue un proceso basado en varias condiciones, concurrencia en el proceso, los datos de acceso, interrupciones y otras alternativas del camino lógico para construir un proceso, sistema o procedimiento.

2.1.1.2.2.1.4 Diagramas de Secuencias

Que es una representación estructurada del comportamiento como una serie de pasos secuenciales a lo largo del tiempo. Se usa para representar el flujo de trabajo, el paso de mensajes y cómo los elementos en general cooperan a lo largo del tiempo para lograr un resultado.

2.1.1.2.2.1.5 Diagrama de Paquetes

Muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.

2.1.1.2.2.1.6 Diagrama de Componentes

Ilustra los fragmentos de software, controladores embebidos, etc. que conformarán un sistema. Un diagrama de componentes tiene un nivel de abstracción más elevado que un diagrama de clase - usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución. Estos son bloques de construcción, como así eventualmente un componente puede comprender una gran porción de un sistema.

2.1.1.2.2.1.7 Diagrama de Despliegue

Muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos, y la construcción interna puede ser representada por nodos o artefactos embebidos. Como los artefactos se ubican en los nodos para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue.

Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

2.1.1.3 Herramientas de Construcción de Software

2.1.1.3.1 Eclipse Ganimedes

Es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

Eclipse es ahora desarrollado por la Fundación Eclipse, una organización

independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

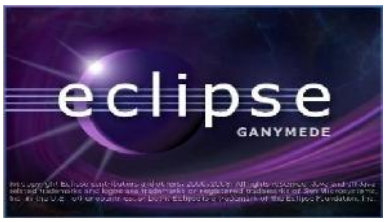


Figura Nº7. Eclipse Ganimedes

2.1.1.3.2 Macromedia Dreamweaver

Adobe Dreamweaver es una aplicación en forma de estudio (basada en la forma de estudio de Adobe Flash) enfocada a la construcción y edición de sitios y aplicaciones Web basadas en estándares.

Creado inicialmente por Macromedia (actualmente producido por Adobe Systems). Es el programa de este tipo más utilizado en el sector del diseño y la programación web, por sus funcionalidades, su integración con otras herramientas como Adobe Flash y, recientemente, por su soporte de los estándares del World Wide Web Consortium. Dreamweaver ha tenido un gran éxito desde finales de los 90 y actualmente mantiene el 90% del mercado de editores HTML.

Permite la conexión a un servidor, a base de datos, soporte para programación en ASP, PHP, Javascript, cliente FTP integrado entre otros.

Figura Nº8. Macromedia Dreamweaver

2.1.1.3.3 Visual Paradigm for UML

Es una herramienta "modelado visual" de UML. La herramienta está diseñada para una gama amplia de usuarios como ser ingenieros de software, analistas de sistemas, analistas comerciales y arquitectos de sistemas o para cualquiera que está interesado en construir sistemas, software de gran potencia que usan un acercamiento orientado a objeto fiablemente.

Además, VP-UML apoya las últimas normas de anotación de UML.

Figura Nº9. Visual Paradigm for UML

2.1.1.3.4 Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pagés (JSP) de Sun Microsystems.

Tomcat tendremos que decidir cómo usarlo, y, si seleccionamos las opciones 2 o 3, también necesitaremos instalar un adaptador de servidor web.

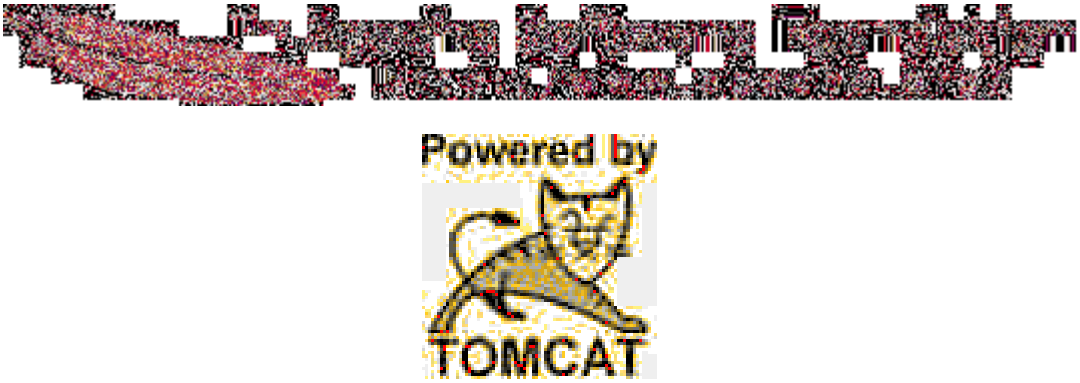


Figura Nº10. Tomcat

2.1.1.3.5 pgAdmin III PostgreSQL Tools

PgAdmin es un proyecto de software libre publicado bajo la licencia de PostgreSQL. El software está disponible en fuente y el formato binario de la red de servidores espejos de PostgreSQL.

Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG [4].

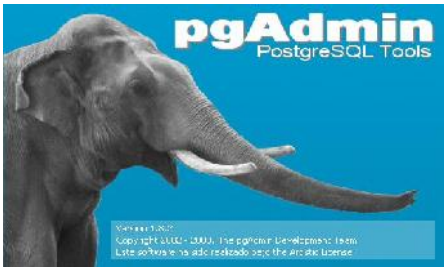


Figura Nº11. PgAdmin III PostgreSQL Tools

2.1.1.3.6 HTML *(HyperText Markup Language)*

HTML (Lenguaje de Marcas de Hipertexto) es un lenguaje estático para el desarrollo de sitios web (Lenguaje de Marcas Hipertextuales). Es un lenguaje de fácil aprendizaje siendo el despliegue del mismo rápido. Así, el HTML es una aplicación del estándar ISO 8879:1986(SGML), que se formalizó en 1990 con la aparición de la Word Wide Web.

2.1.1.3.7 *XML (Markup Language)*

Es un metalenguaje para la descripción y estructuración de datos utilizando marcas (Markup Language). Metalenguaje significa un lenguaje para definir otros lenguajes: XHTML, WML, etc.

El Objetivo de XML es separar de un documento o Información.

Facilidad de Administración (separar contenido, 1ógica y presentación).

Muchas aplicaciones Web que extraen información de BD, la convierten en tablas, perdiendo información de los campos.

2.1.1.3.8 Java

Java es un lenguaje originalmente desarrollado por un grupo de ingenieros de Sun, utilizado por Netscape posteriormente como base para Java script. Si bien su uso se destaca en el Web, sirve para crear todo tipo de aplicaciones (locales, intranet o internet).

Java es un lenguaje de objetos, independiente de la plataforma.

Java posee ciertas características que hoy día se consideran estándares en los lenguajes OO:

- Objetos.
- Clases.
- Métodos.
- Subclases.
- Herencia simple.
- Enlace dinámico.
- Encapsulamiento.

Java es un lenguaje que ha sido diseñado para producir software:

- Confiable: Minimiza los errores que se escapan a la fase de prueba.
- Multiplataforma: Los mismos binarios funcionan correctamente en Windows/95/NT/XP/VISTA, Linux, Unix/Motif y Power/Mac.
- Seguro: Applets recuperados por medio de la red no pueden causar daño a los usuarios.
- Orientado a objetos: Beneficioso tanto para el proveedor de bibliotecas de clases como para el programador de aplicaciones.
- Robusto: Los errores se detectan en el momento de producirse, lo que facilita la depuración.

Figura Nº12. Java

2.1.1.3.9 Arquitectura y Diseño: Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

El patrón MVC se ve frecuentemente en *aplicaciones web*, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Para el diseño de aplicaciones con sofisticados interfaces se utiliza el patrón de diseño Los elementos del patrón son tres:

- Modelo: datos y reglas de negocio.
- Vista: muestra la información del modelo al usuario.
- Controlador: gestiona las entradas del usuario.

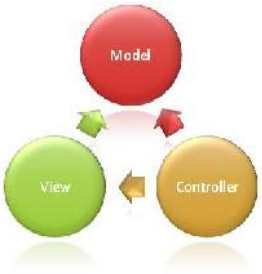


Figura Nº13. Modelo Vista Controlador (MVC)

2.1.1.3.9.1 Componentes Patrón MVC

- **Modelo:** Corresponde a la parte de la aplicación que implementa la lógica del Dominio de Datos de la aplicación, además de incorporar la persistencia de datos. Frecuentemente, los objetos del modelo recuperan y almacenan los

estados del modelo en la base de datos. Aquí se hace el levantamiento de los objetos que el sistema deba utilizar y es el proveedor de los recursos al Controlador

- **Vista:** Es el componente que despliega la interfaz de usuario de aplicación. Normalmente (no siempre), esta UI es creada a partir de los datos del modelo (Un ejemplo puede ser una vista de edición de una tabla de Productos que muestra cuadros de texto, listas desplegables y casillas de verificación basado en el estado actual de los objetos Producto). Esta puede ser una web HTML, un XML, un archivo binario, etc.
- **Controlador** El controlador maneja la interacción del usuario, trabaja con el Modelo, y por último selecciona la vista a renderizar en la interfaz de usuario. En una aplicación MVC, la vista sólo despliega información; el controlador maneja y responde los inputs e interacción del usuario. Éste escucha los cambios a la vista y se los envía al modelo.

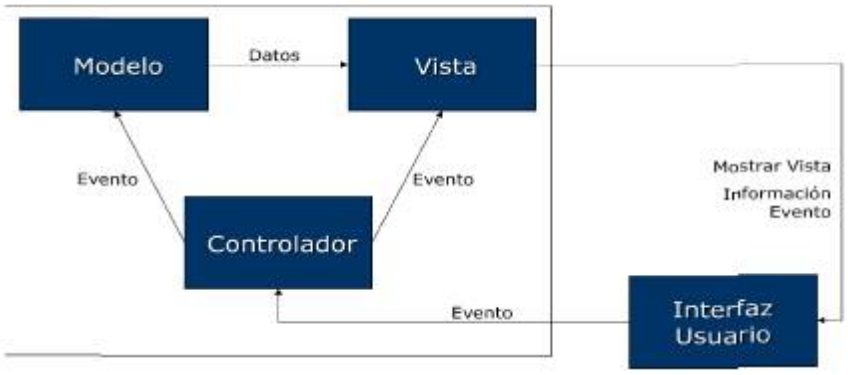


Figura Nº14. Patrón MVC

2.1.1.3.10 Hibérnate

Hibérnate ofrece la *Persistencia Relacional para Java*, que para los no iniciados, proporciona unas muy buenas maneras para la persistencia de sus objetos de Java a y desde una base de datos subyacente. Más que ensuciar con SQL tus objetos y convertir consultas a y desde los objetos de primera magnitud, Hibérnate puede preocuparse de todo ese maremágnum por ti. Tú utilizas solamente a los objetos, Hibérnate se preocupa del SQL y de que las cosas terminan en la tabla correcta.

Permite Trabajar con software orientado a objetos y bases de datos relacionales puede hacernos invertir mucho tiempo en los entornos actuales.

Hibérnate es una herramienta que realiza el *mapping* entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java. El término utilizado es ORM (object/relational mapping) y consiste en la técnica de realizar la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa.

Hibérnate no solo realiza esta transformación sino que nos proporciona capacidades para la obtención y almacenamiento de datos de la base de datos que nos reducen el tiempo de desarrollo.

Figura Nº15. Hibérnate

2.1.1.4 Base de Datos

2.1.1.4.1 Componentes principales de una Base de Datos

Los principales componentes de una base de datos son:

- Datos: Los datos son la Base de Datos propiamente dicha.
- Hardware: Se refiere a los dispositivos de almacenamiento en donde reside la base de datos así como los dispositivos periféricos (Unidad de Control, Canales de Comunicación, etc.) necesarios para su uso.
- Software: Está constituido por un conjunto de programas que se conoce como Sistema Manejador de Base de Datos (DBMS), manejando éste todas las solicitudes formuladas por los usuarios a la base de datos.
- Usuarios: Normalmente identificándose 3 tipos:
    - El programador de aplicaciones.
    - El usuario Final.
    - El Administrador de la Base de Datos quien se encarga del control general del Sistema de Base de Datos.

2.1.1.4.2 Herramienta PostgreSQL 8.3 para el manejo de la BD

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley.

PostgreSQL es una derivación libre (OpenSource), y utiliza el lenguaje SQL92/SQL99, así como otras características.

PostgreSQL es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos.

2.1.2 Plan de Desarrollo de Software

2.1.2.1 Introducción

El presente documento es un Plan de Desarrollo del Software que sentará las bases para el desarrollo del proyecto, es una versión preparada para ser incluida en la propuesta elaborada en respuesta al proyecto de la asignatura de Taller III de la Carrera de Ingeniería Informática de la Facultad de Ciencias y Tecnología de la Universidad Autónoma "Juan Misael Saracho". Este documento provee una visión global del enfoque de desarrollo propuesto.

El proyecto ha sido propuesto por el Universitario Maritza Tatiana Vedia Mollo, basado en la Metodología RUP (Rational Unified Process), en la que se procederá a cumplir con las cuatro fases que marca la metodología. Es importante destacar esto puesto que utilizaremos la terminología RUP en este documento. Se incluirá el detalle para las fases de Inicio y Elaboración y adicionalmente se esbozarán las fases posteriores de Construcción y Transición para dar una visión global de todo proceso.

El enfoque desarrollo propuesto constituye una configuración del proceso RUP de acuerdo a las características del proyecto, seleccionando las actividades a realizar y los artefactos (entregables) que serán generados. Este documento es a su vez uno de los artefactos de RUP.

2.1.2.1.1 Propósito

El propósito del Plan de Desarrollo de Software es proporcionar la información necesaria para controlar el proyecto. En él se describe el enfoque de desarrollo del software.

Los usuarios del Plan de Desarrollo del Software son:

- El Director del proyecto, que lo utiliza para organizar la agenda y necesidades de recursos y para realizar su seguimiento.
- El Docente para evaluar el cumplimiento del proyecto.

2.1.2.1.2 Alcance

Aplicando el Plan de Desarrollo Software obtenemos una herramienta importante para realizar nuestro plan de trabajo el cual coadyuvará al cumplimiento de nuestros objetivos en el tiempo propuesto gracias al cronograma de actividades establecido.

2.1.2.1.3 Resumen

Después de esta introducción, el resto del documento está organizado en las siguientes secciones:

Vista General del Proyecto proporciona una descripción del propósito, alcance y objetivos del proyecto, estableciendo los artefactos que serán producidos y utilizados durante el proyecto.

Organización del Proyecto describe la estructura organizacional del equipo de desarrollo.

Gestión del Proceso explica los costos y planificación estimada, define las fases e hitos del proyecto y describe cómo se realizará su seguimiento.

Planes y Guías de aplicación proporciona una vista global del proceso de desarrollo de software, incluyendo métodos, herramientas y técnicas que serán utilizadas.

2.1.2.2 Vista General del Proyecto

2.1.2.2.1 Propósito, Alcance y Objetivos

La información que a continuación se incluye ha sido extraída de las diferentes

reuniones, entrevistas y charlas que se han realizado con los usuarios destino de la Parroquia San Martin de Porres de Tarija desde el inicio del proyecto.

2.1.2.2.1.1 Propósito

Con el presente proyecto se pretende Mejorar "La Gestión de la Información Sacramental en la Parroquia San Martin de Porres".

2.1.2.2.1.2 Alcance

El Plan de Desarrollo del Software describe el plan global usado para el desarrollo de SIGES "Sistema Informático de Gestión Sacramental", un Sistema Informático que está comprendido por los siguientes Módulos:

- **Módulo Administrador.**
- **Módulo Sacramental.**
- **Módulo Reportes.**
- **Módulo Backup.**

Analizando factores predominantes dentro de la institución, podemos llegar a la conclusión de que la implementación del sistema automatizado dará beneficios claramente perceptibles, dando soluciones a problemas que arrastra; este sistema proyectará una solución a mediano plazo que beneficiará a dicha organización, pero lo más importante mejorará la atención a los feligreses.

Esta propuesta de sistema (Software) contiene una serie de alternativas de mejoramiento para las expectativas futuras de la parroquia, las cuales se detallan a continuación:

- Brindar seguridad al sistema mediante una clave de ingreso, permitiendo el acceso al mismo sólo al personal autorizado.
- Opciones que permitan el registro computarizado de los sacramentos que realizan los feligreses en la parroquia, además de realizar copia de seguridad de datos.
- Desarrollar un manual de usuario y de instalación del sistema.

2.1.2.2.1.3 Objetivos

2.1.2.2.1.3.1 Objetivo General

Mejorar “La Gestión de la Información Sacramental en la Parroquia San Martin de Porres”

2.1.2.2.1.3.2 Objetivos Específicos

- Llegar a brindar un sistema rápido y eficiente para la manipulación de registros de los feligreses.
- Conseguir mayor organización en los registros por sacramento.
- Obtener un sistema que permita obtener reportes de forma rápida y sencilla.
- Otorgar a los usuarios total seguridad en el resguardo de su información, permitiendo el acceso sólo a usuarios autorizados.
- Crear una base de datos de acuerdo a las necesidades de la institución.
- Diseñar una interfaz de usuario de fácil manejo.

2.1.2.2.2 Suposiciones y Restricciones

Acontecimientos que deben ocurrir para que el proyecto sea ejecutado con éxito pero que están totalmente fuera del ámbito del control del equipo de proyecto.

2.1.2.2.2.1 Suposiciones

Suponemos que la Parroquia San Martin de Porres de Tarija cuenta con:

- Equipo de computación. En el que incluye un computador, impresora.
- Un ambiente adecuado para la implementación del sistema.
- La disponibilidad de otorgar toda la información necesaria para la creación del software.
- Personal con conocimientos básicos de computación.
- Cuenta con equipos en los cuales se podría implementar el sistema.
- Presupuesto suficiente para la implementación del Sistema.
- Interés de la institución para utilizar el Sistema.
- La formación del personal encargado se lleve en la fecha determinada.
- Se presentarán en la fecha indicada o prescrita por el docente los diferentes

documentos como el Plan de Desarrollo de Software y los componentes descritos en el presente documento.

2.1.2.2.2.2 Restricciones

Limitaciones generalmente fuera del ámbito de control del equipo de proyecto que pueden afectar negativamente a su alcance.

- El sistema será restringido, sólo usuarios privilegiados podrán acceder al sistema.
- Para la manipulación de la base de datos, sólo podrán acceder el personal autorizado.
- Deberá contar como mínimo con un sistema operativo Windows XP/Vista/Seven o Linux.
- El administrador deberá contar con el conocimiento necesario para el manejo del sistema.
- No se incluirá ningún reporte contable.
- El sistema no abarca el control de otras áreas de la Parroquia.
- No contar con un Servidor Web que presente las características necesarias para la ejecución del Sistema.
- No contar con los fondos suficientes para llevar a cabo la Capacitación a los administradores del Sistema.

2.1.2.2.3 Entregables del proyecto

A continuación se indican y describen cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los entregables. Esta lista constituye la configuración de RUP desde la perspectiva de artefactos que se propone para este proyecto.

Es preciso destacar que de acuerdo a la filosofía de RUP (y de todo proceso iterativo e incremental), todos los artefactos son objeto de modificaciones a lo largo del proceso de desarrollo, con lo cual, sólo al término del proceso podríamos tener una versión definitiva y completa de cada uno de ellos. Sin embargo, el resultado de cada

iteración y los hitos del proyecto están enfocados a conseguir un cierto grado de completitud y estabilidad de los artefactos. Esto será indicado más adelante cuando se presenten los objetivos de cada iteración.

2.1.2.2.3.1 Plan de desarrollo del Software.

Es el presente documento.

2.1.2.2.3.2 Modelo de Casos de Uso del Negocio

Es un modelo de las funciones de negocio vistas desde la perspectiva de los actores externos (Agentes de registro, solicitantes finales, otros sistemas etc.) permite situar al sistema en el contexto organizacional haciendo énfasis en los objetivos. Este modelo se representa con un Diagrama de Casos de Uso usando estereotipos específicos para este modelo.

La definición del conjunto de procesos del negocio es una tarea crucial, ya que define los límites del proceso de modelado posterior, consideramos los objetivos estratégicos de la organización, teniendo en cuenta que esos objetivos serán descompuestos en un conjunto de sub-objetivos más concretos, para la identificación de procesos de negocio. Se presentan los modelos definidos en RUP como modelo del negocio (modelo de casos de uso del negocio y de objetos del negocio).

2.1.2.2.3.2.1 Introducción

El Modelo de Caso de Uso del Negocio es un artefacto de la disciplina Requisitos en la metodología RUP la cual estamos implementando.

2.1.2.2.3.2.2 Propósito

Comprender la Estructura y la Dinámica de la Organización.

Comprender problemas actuales e identificar posibles mejoras.

2.1.2.2.3.2.3 Alcance

Describe los Procesos de Negocio y los Clientes.

Identifica y Describe los Procesos de Negocio según los Objetivos de la Organización.

Definir un caso de uso del negocio para cada proceso de negocio.

2.1.2.2.3.2.4 Diagramas de Casos de Uso del Negocio

2.1.2.2.3.2.4.1 Modelo Casos de Uso del Negocio Feligreses

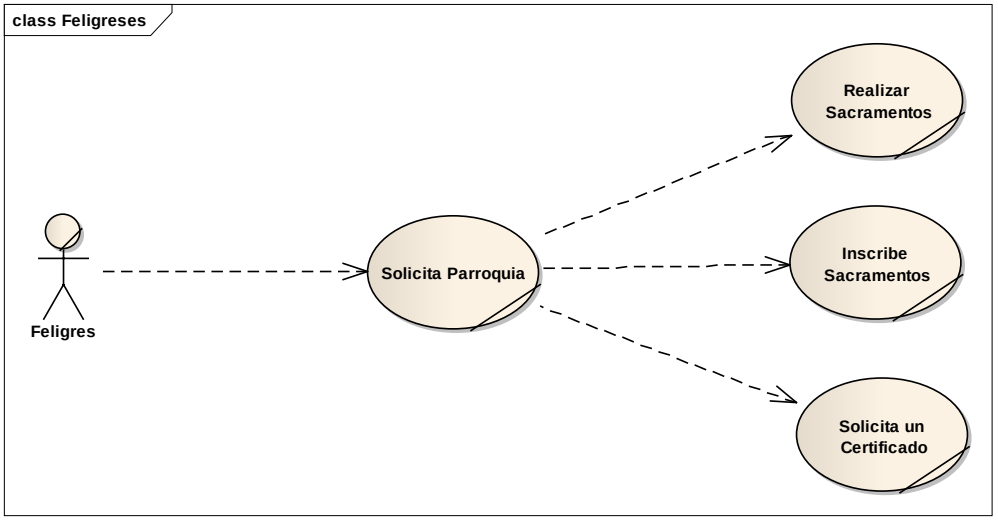


Figura Nº16. Modelo Casos de Uso del Negocio Feligreses

2.1.2.2.3.2.4.2 Modelo Casos de Uso del Negocio Sacerdotes

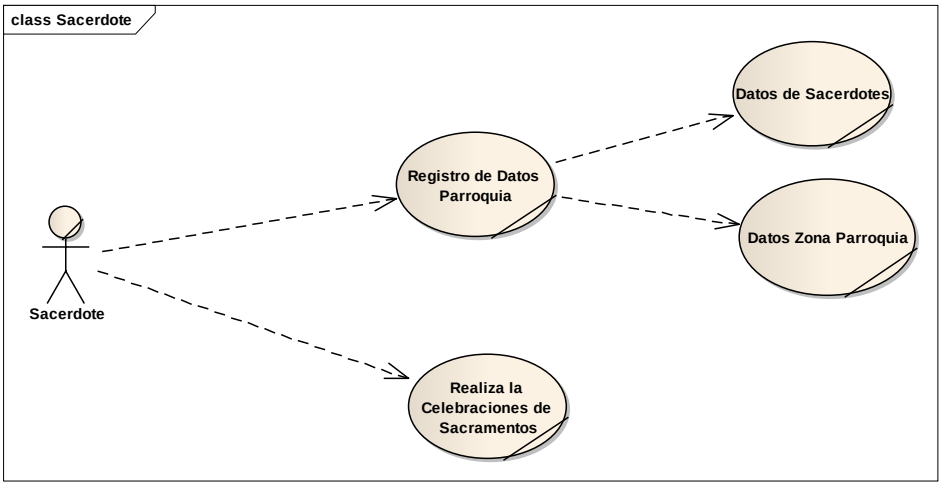


Figura Nº17. Modelo Casos de Uso del Negocio Sacerdotes

2.1.2.2.3.2.4.3 Modelo Casos de Uso del Negocio Secretario

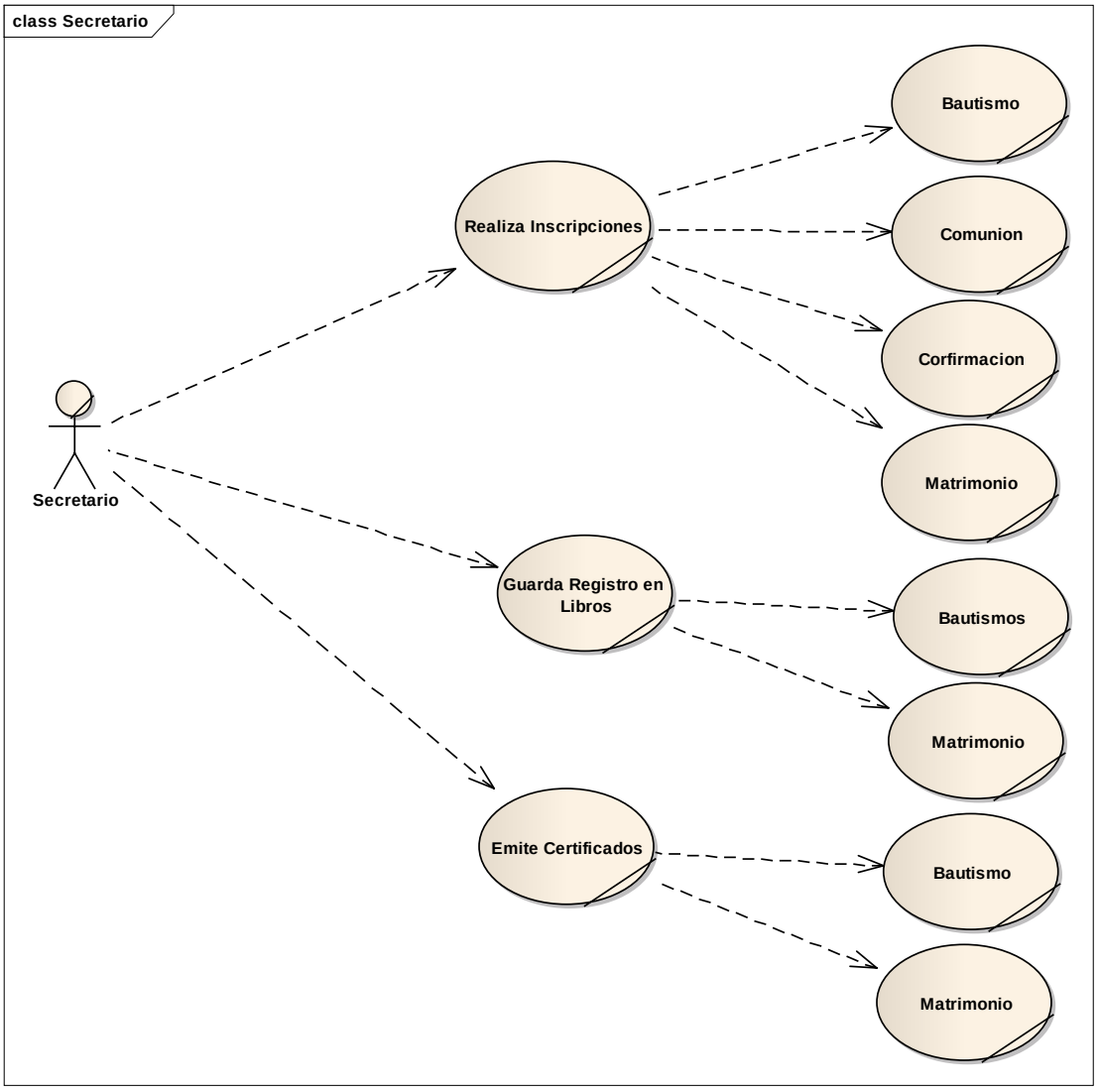


Figura N°18. . Modelo Casos de Uso del Negocio Secretario

2.1.2.2.3.3 Modelo de Objetos del Negocio

Es un modelo que describe la realización de cada caso de uso del negocio, estableciendo los actores internos, la información que en términos generales manipulan y los flujos de trabajo (workflows) asociados al caso de uso del negocio. Para la representación de este modelo se utilizan Diagramas de Colaboración (para mostrar actores externos, internos y las entidades (información) que manipulan, un Diagrama de Clases para mostrar gráficamente las entidades del sistema y sus relaciones, y Diagramas de Actividad para mostrar los flujos de trabajo.

2.1.2.2.3.3.1 Introducción

El Modelo de Objetos del Negocio es un artefacto de la disciplina Requisitos en la metodología RUP la cual estamos implementando.

2.1.2.2.3.3.2 Propósito

Comprender la Estructura y la Dinámica de la Organización.

Comprender los Procesos del negocio de la Organización.

2.1.2.2.3.3.3 Alcance

Describe el comportamiento de los procesos de negocio.

Identificar y definir los objetos del negocio.

2.1.2.2.3.3.4 Diagramas de Objetos del Negocio

2.1.2.2.3.3.4.1 Modelo de Objeto del Negocio: Feligrés solicita Sacramento Bautismo

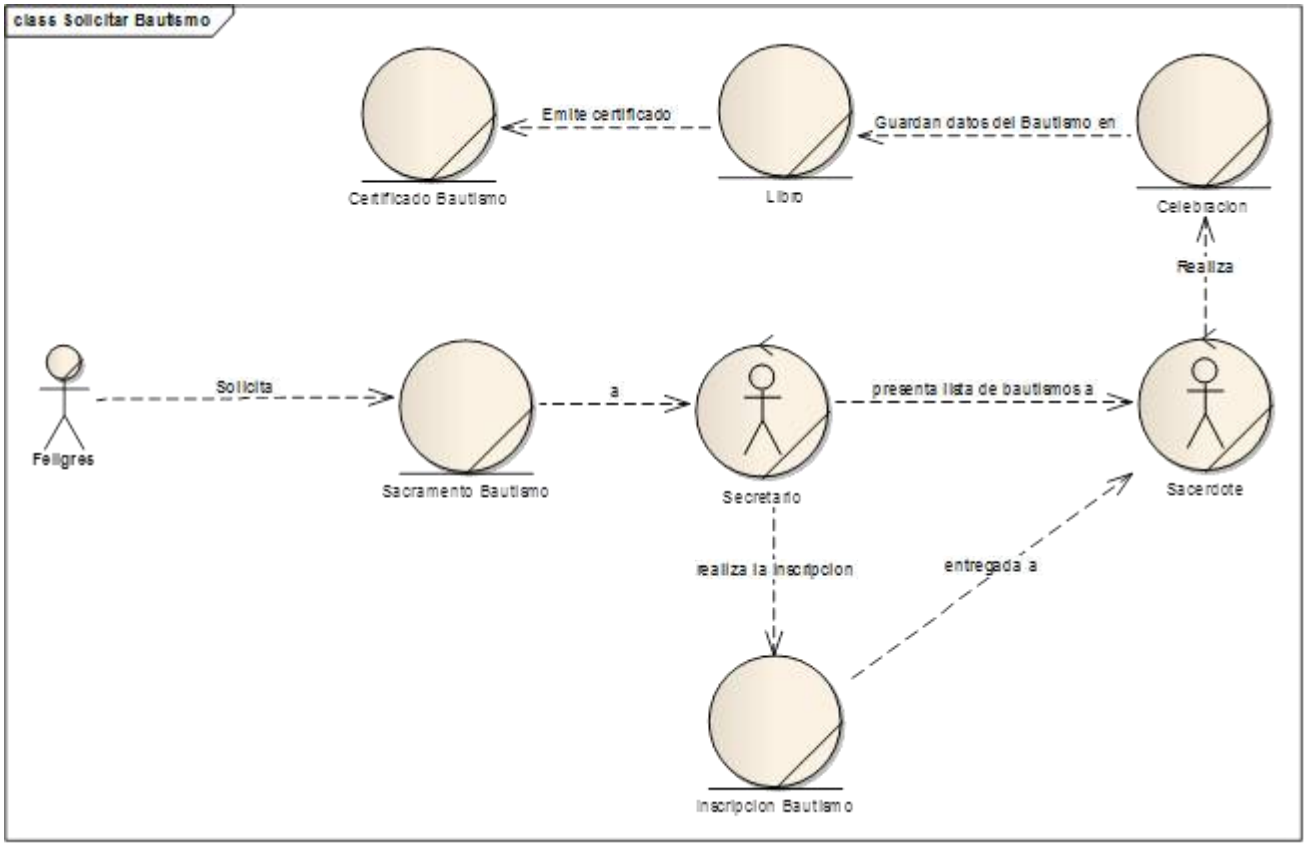


Figura Nº19. Modelo de Objeto del Negocio: Feligrés Solicita Sacramento Bautismo

2.1.2.2.3.3.4.2 Modelo de Objeto del Negocio: Feligrés solicita Certificado de Bautismo

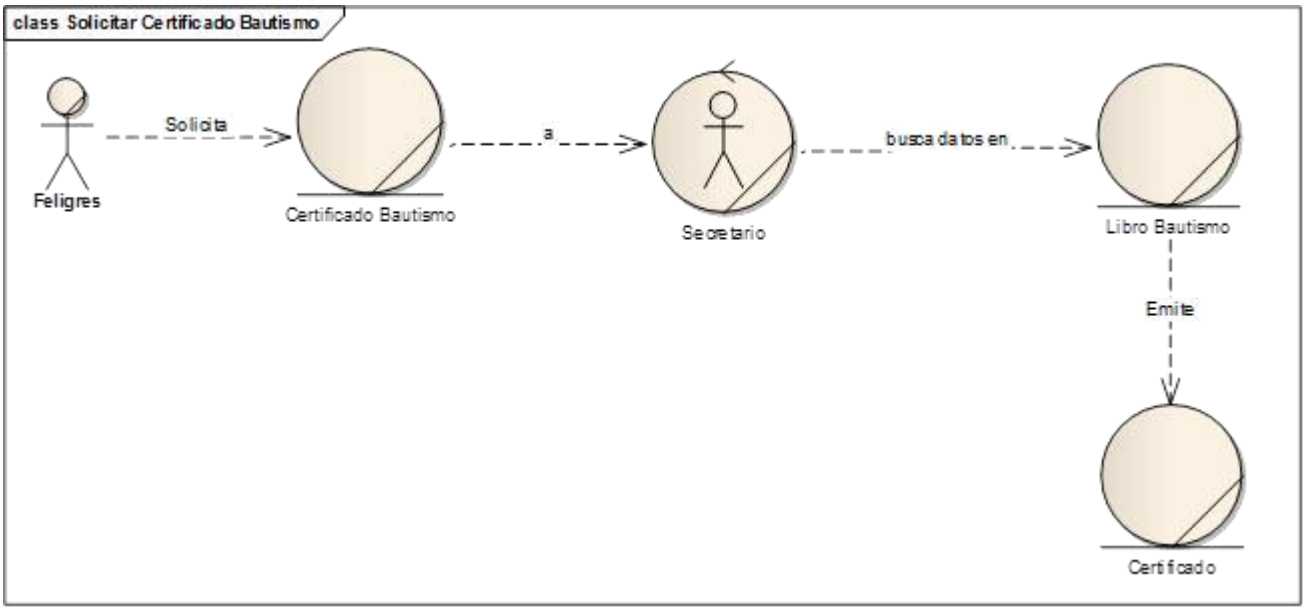


Figura Nº20. Modelo de Objeto del Negocio: Feligrés Solicita Certificado de Bautismo

2.1.2.2.3.3.4.3 Modelo de Objeto del Negocio: Feligrés solicita Sacramento Comunión

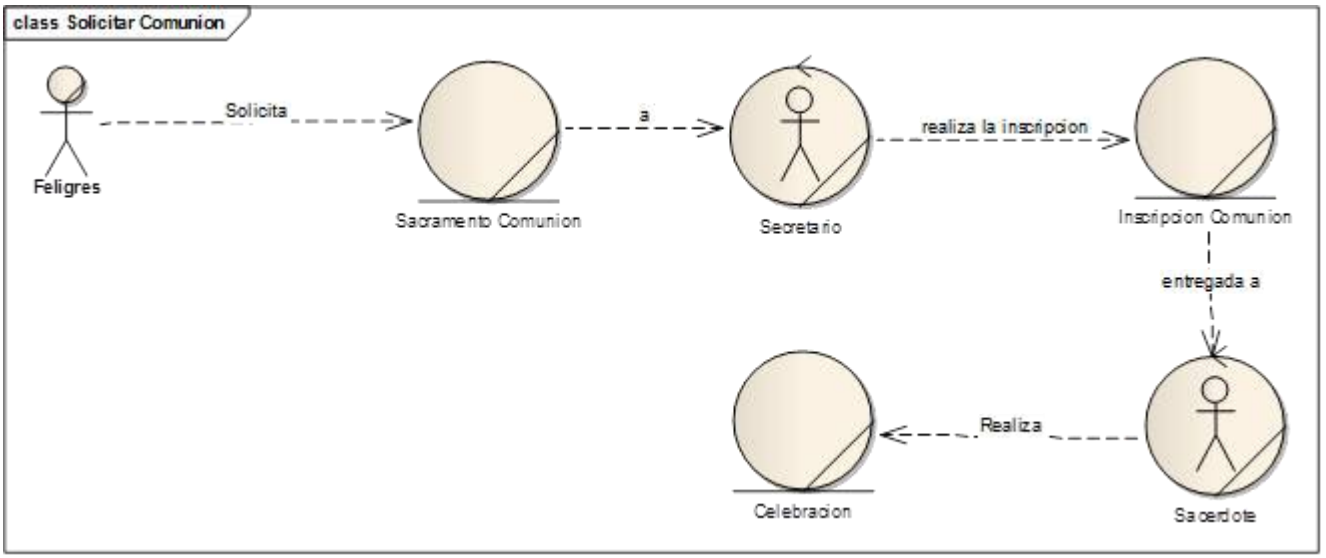


Figura 46. Modelo de Objeto del Negocio: Feligrés Solicita Sacramento Comunión

2.1.2.2.3.3.4.4 Modelo de Objeto del Negocio: Feligrés solicita Sacramento Confirmación

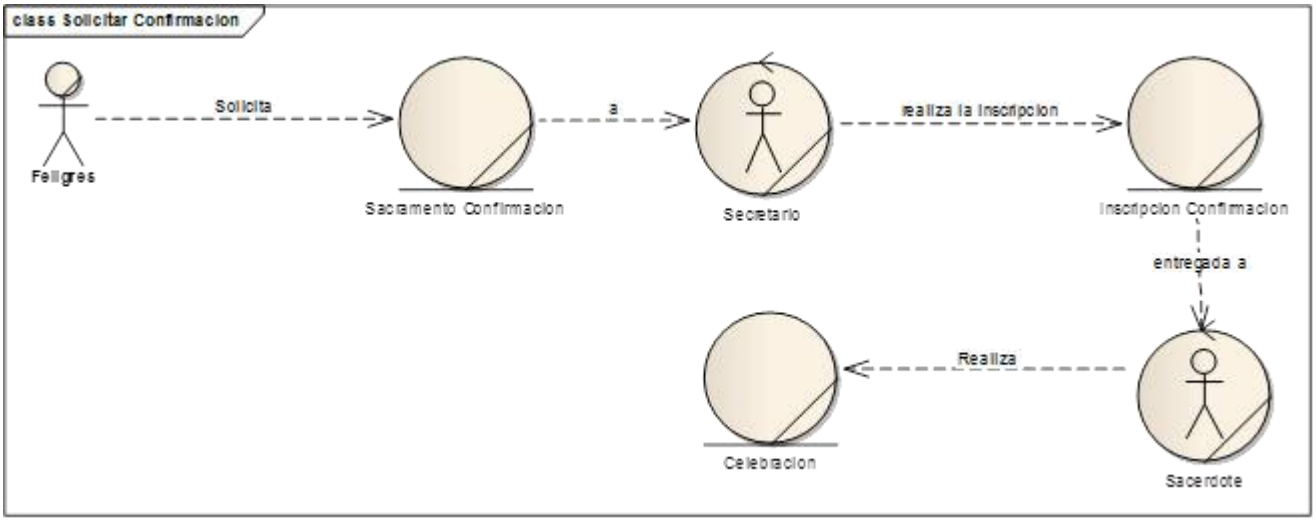


Figura Nº21. Modelo de Objeto del Negocio: Feligrés Solicita Sacramento Confirmación

2.1.2.2.3.3.4.5 Modelo de Objeto del Negocio: Feligrés solicita Certificado Matrimonio

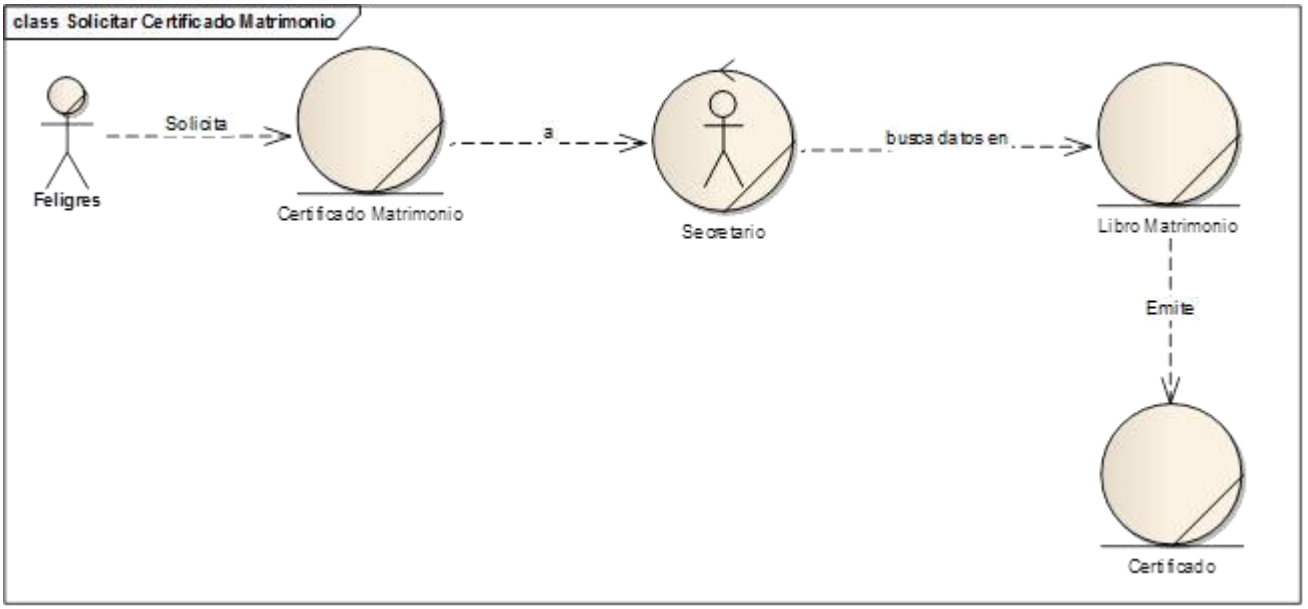


Figura Nº22. . Modelo de Objeto del Negocio: Feligrés Solicita Certificado Matrimonio

2.1.2.2.3.3.4.6 Modelo de Objeto del Negocio: Feligrés solicita Sacramento Matrimonio

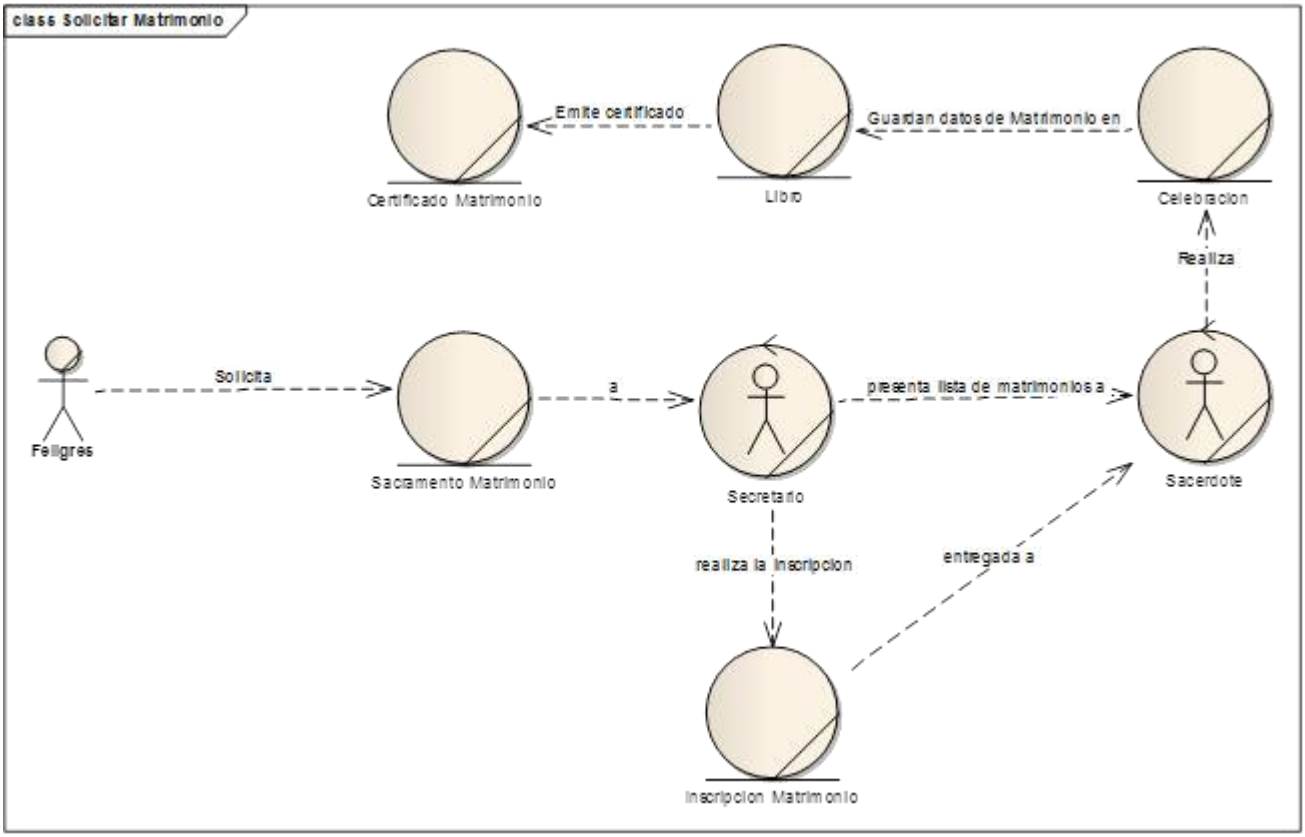


Figura Nº23. Modelo de Objeto del Negocio: Feligrés Solicita Sacramento Matrimonio

2.1.2.2.3.4 Glosario

Es un documento que define los principales términos usados en el proyecto. Permite establecer una terminología consensuada.

2.1.2.2.3.4.1 Introducción

El presente documento recoge los términos manejados durante la elaboración del proyecto de desarrollo del Sistema Informático de Gestión Sacramental. Se trata de un diccionario informal de datos y de definiciones de la nomenclatura que se maneja en la construcción del Sistema.

2.1.2.2.3.4.2 Propósito

El propósito del presente documente es definir la terminología manejada en el proyecto a desarrollar, también sirve como guía de consulta para la clarificación de los puntos conflictivos o poco esclarecidos del proyecto.

2.1.2.2.3.4.3 Alcance

El alcance del presente documento se extiende a todo el proyecto en desarrollo.

2.1.2.2.3.4.4 Organización del Glosario

El presente documento está organizado por definiciones de términos ordenados de forma ascendente según ordenación alfabética tradicional.

2.1.2.2.3.4.5 Definiciones

Administrador:

Persona con acceso, no sólo a las funcionalidades, sino a las interioridades de la BD.

Usuarios: personas que desean usar las funcionalidades del sistema.

Autorización de Bautismo:

Documento expedido por el Párroco de la Parroquia a la que pertenece un próximo nuevo miembro de la comunidad cristiana, en el que autoriza que éste reciba el Bautismo en una parroquia diferente a la correspondiente según el domicilio del catecúmeno [1].

Backup (copia de respaldo*, copia de seguridad)*:

Copia de ficheros o datos de forma que estén disponibles en caso de que un fallo produzca la pérdida de los originales. Esta sencilla acción evita numerosos, y a veces irremediables, problemas si se realiza de forma habitual y periódica.

Base de Datos:

Se encarga de almacenar información acerca de administradores, usuarios, y todo referente a las actividades de los usuarios, con el ordenador y componentes del sistema.

Bautismo:

Es el primer sacramento que se recibe en la iglesia católica. Por este sacramento somos miembros de la Iglesia e iniciamos nuestra vida cristiana [1].

Católico:

Significa "universal" La iglesia se llama católica porque está abierta a todos los hombres de cualquier región, raza o condición [1].

Catequesis:

Es la acción por la cual la Iglesia educa en la fe a sus miembros, sean estos niños, jóvenes o adultos [1].

Catequista:

Agente de pastoral con la formación necesaria para impartir la catequesis [1].

Catecúmeno:

Persona que se está instruyendo en la doctrina católica para ponerse en disposición de recibir algún sacramento [1].

Certificado de Bautismo:

Documento acreditativo de que la persona que se indica en el mismo está bautizado por la Iglesia Católica [1].

Certificado de Matrimonio:

Documento acreditativo de que la persona que se indica en el mismo está casada por

la Iglesia Católica [1].

Código Fuente:

Texto escrito en un lenguaje de programación específico y que puede ser leído por un programador.

Computadora (computador, *ordenador):*

Máquina electrónica capaz de procesar información siguiendo instrucciones almacenadas en programas. Antes que electrónicas estas máquinas fueron mecánicas o electromecánicas.

Comunión:

Sacramento de unidad en la fe, de todos los católicos. También se refiere al acto de recibir la Hostia consagrada en la celebración eucarística porque expresa la unidad de Cristo y su Iglesia [1].

Confirmación:

Sacramento por el cual se ratifica las promesas hechas en el sacramento del Bautismo [1].

Diagrama:

Representación gráfica de un conjunto de elementos, representando la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones).

Diócesis:

Es una jurisdicción territorial de la Iglesia y cuya autoridad máxima es el Obispo [1].

Interfaz de Usuario:

La interfaz de usuario (IU) es uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. La interfaz de usuario es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario. La IU es responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible.

Libro de Sacramento:

Conjunto de asientos de un sacramento encuadernados que recogen información sobre la vida partícipe de miembros de la Iglesia. Cada libro está identificado unívocamente [1].

Matrimonio:

Unión legitima del hombre y la mujer en las condiciones previstas. Por el sacramento del matrimonio, los esposos se comprometen de por vida ante Dios y la Iglesia.

Nota de Confirmación:

Documento que se envía de parroquia en la que se bautizó el catecúmeno al que se refiere la nota donde se certifica que ha sido confirmado para que se registre en su asiento bautismal [1].

Nota Marginal:

Referida a la anotación que se hace al margen de los libros de sacramentos para indicar que se ha realizado alguna modificación, cambio o corrección sobre un asiento determinado. En el caso del libro de bautismo se indicarán en forma de nota marginal que el bautizado ha recibido otros sacramentos, de forma que quede registrado en un único lugar que los ha recibido. Estos sacramentos son: comunión, confirmación, matrimonio [1].

Ocupación:

Labor que desarrolla el empleado, obrero, contratista, etc.

Operador de Base de Datos:

Persona con acceso, a las interioridades de la BD.

Paquete:

Mecanismo de propósito general para organizar elementos en grupo.

Parentesco:

La relación familiar ya sea hijo hija, esposa (o) madre padre, etc. [1].

Párroco:

Presbítero que, en nombre del obispo, se hace cargo de la parroquia asignada [1].

Parroquia:

Una determinada comunidad de feligreses, en un territorio, constituida de modo estable en una diócesis y que se encomienda a un párroco. La parroquia es la más pequeña división jurídica de la Iglesia [1].

Password (palabra de paso, *contraseña)*:

Conjunto de caracteres alfanuméricos que permite a un usuario el acceso a un determinado recurso o la utilización de un servicio dado.

Párroco:

Responsable de la Parroquia que se le confía, ejerciendo de pastor de la comunidad en que está encomendado por el Obispo diocesano [1].

Requerimientos Funcionales:

Se refiere a la funcionalidad o los servicios que se espera que el sistema provea.

Requerimientos no Funcionales:

Los requerimientos no funcionales tienen que ver con las características que de una u otra forma puedan limitar el sistema como son: el rendimiento (en tiempo y espacio), confiabilidad, interfaces, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc.

Sacramento:

Son signos visibles que fueron instituidos por Jesucristo y de la Iglesia por la cual el creyente celebra su unión con cristo. Los sacramentos son siete: Bautismo, Reconciliación, Comunión, Confirmación, Unción de los enfermos, Orden Sacerdotal, Matrimonio [1].

Sacerdote:

Se emplea para designar al presbítero, quien ha recibió el sacramento de Orden Sacerdotal. Colabora con el obispo en su acción pastoral, en la enseñanza, la

predicación del evangelio y la celebración de los sacramentos. También se les llama “presbíteros”, “curas” o “clérigos” [1].

Sistema Informático:

Conjunto de partes (hardware y software) que funcionan relacionándose entre sí con un objetivo preciso. Los usuarios son parte del sistema informático.

Sistema Operativo:

Un sistema operativo (SO) es un conjunto de programas o software destinado a permitir la comunicación del usuario con un ordenador y gestionar sus recursos de manera cómoda y eficiente. Comienza a trabajar cuando se enciende el ordenador, y gestiona el hardware de la máquina desde los niveles más básicos .Ejemplos Windows, Linux, MacOS, Solaris.

Usuario:

El *usuario* de un producto informático (bien sea hardware o software), es la persona a la que va destinada dicho producto una vez que ha superado las fases de desarrollo correspondientes. Normalmente, el software se desarrolla pensando en la comodidad del *usuario final*, y por esto se presta especial interés y esfuerzo en conseguir una interfaz de usuario lo más clara y sencilla posible.

2.1.2.2.3.4.6 Abreviaturas

2.1.2.2.3.4.6.1 BBDD, BD:

Bases de Datos, Base de Datos.

2.1.2.2.3.4.6.2 RUP:

Proceso del desarrollo del software.

2.1.2.2.3.4.6.3 SQL:

Lenguaje de Consulta estructurado.

2.1.2.2.3.4.6.4 TCP:

Protocolo de control de transferencia.

2.1.2.2.3.4.6.5 URL:

Uniform Resource Locator (Localizados Uniformes de Recursos).

2.1.2.2.3.4.6.6 Web:

Sitio de interés. Fuente de Hipertexto.

2.1.2.2.3.5 Modelo de Casos de Uso

El modelo de Casos de Uso presenta las funciones del sistema y los actores que hacen uso de ellas. Se representa mediante Diagramas de Casos de Uso.

2.1.2.2.3.5.1 Introducción

El presente documento es un artefacto de la disciplina Requisitos en la metodología RUP la cual estamos implementando.

2.1.2.2.3.5.2 Propósito

Comprender la estructura y la dinámica del sistema desarrollado.

Identificar el nivel de complejidad del sistema.

Identificar posibles mejoras.

2.1.2.2.3.5.3 Alcance

Identificar y definir procesos del sistema según los objetivos de la organización.

Definir un Caso de Uso para cada proceso del sistema (el diagrama de caso de uso nos detalla el contexto y los límites de la organización.

Diagramas de Casos de Uso.

2.1.2.2.3.5.4 Diagramas de Casos de Uso

2.1.2.2.3.5.4.1 Casos de Usos del Sistema General

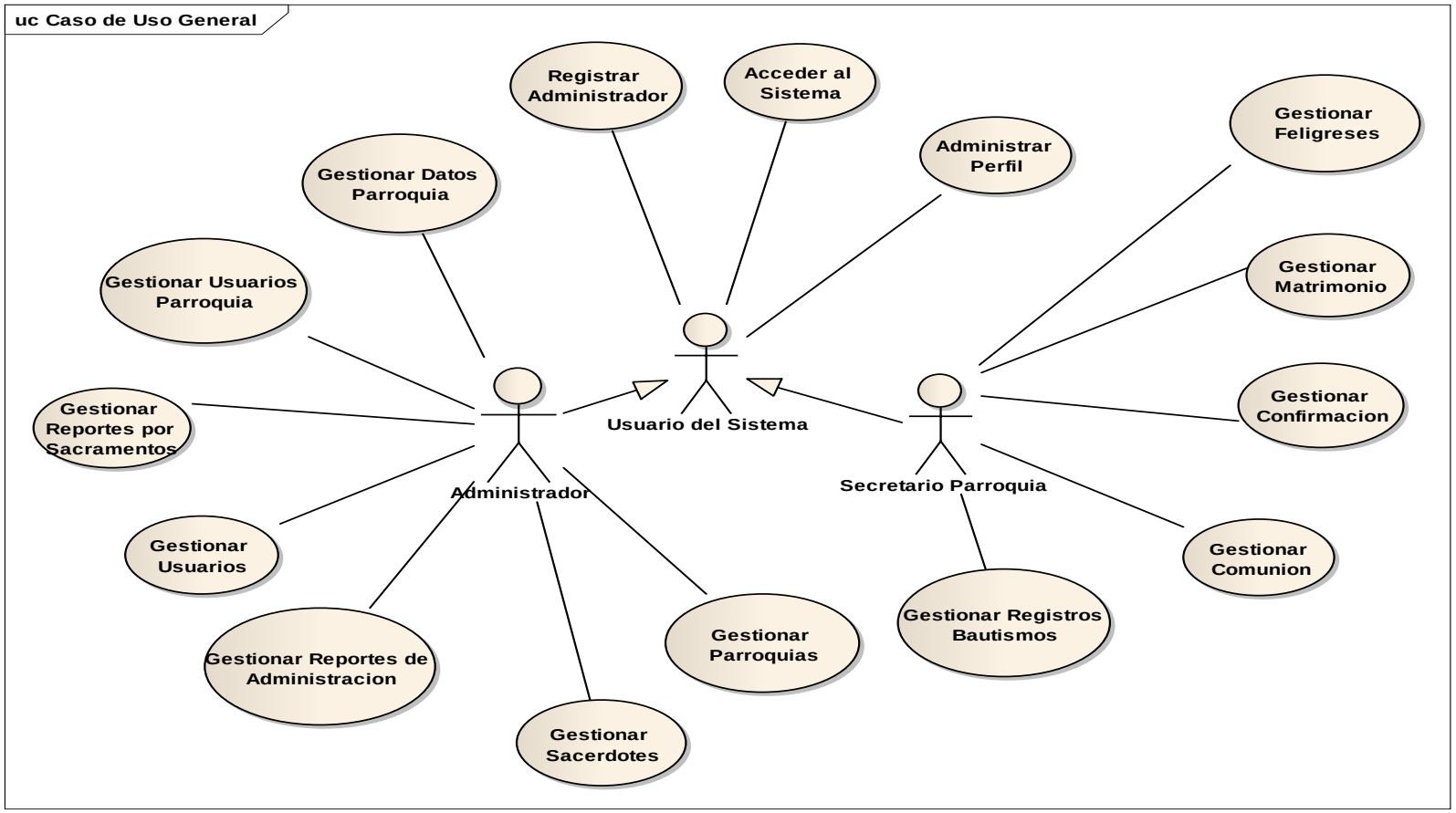


Figura Nº24. Casos de Usos del Sistema General

2.1.2.2.3.5.4.1.1 Modelo Casos de Uso Actores del Sistema

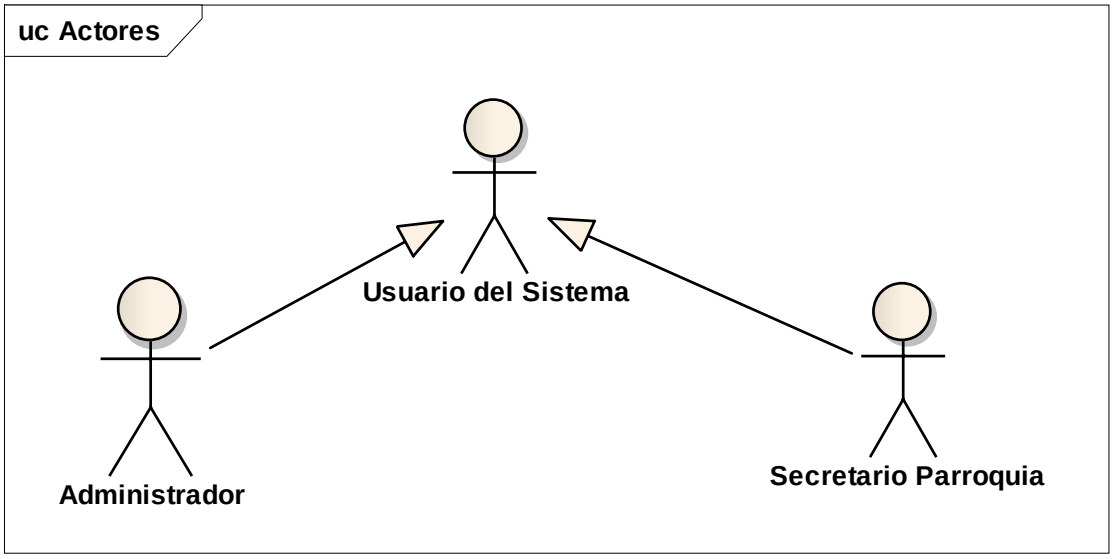


Figura Nº25. Modelo Casos de Usos Actores del Sistema

2.1.2.2.3.5.4.1.2 Modelo Casos de Uso Acceder al Sistema

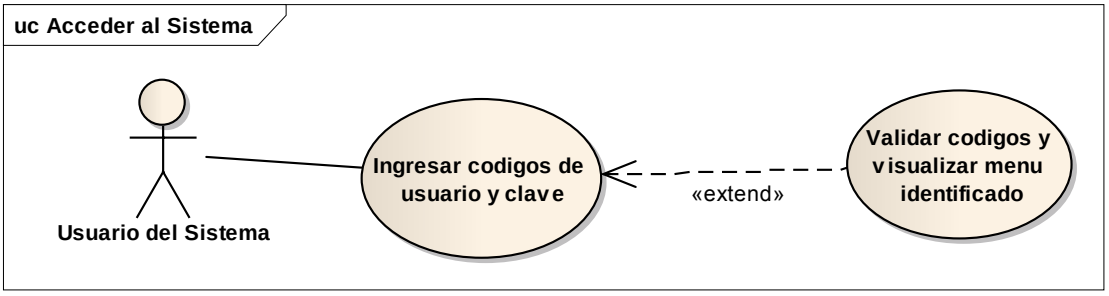


Figura Nº26. Modelo Casos de Usos Acceder al Sistema

2.1.2.2.3.5.4.1.3 Modelo Casos de Uso Gestión Administrador

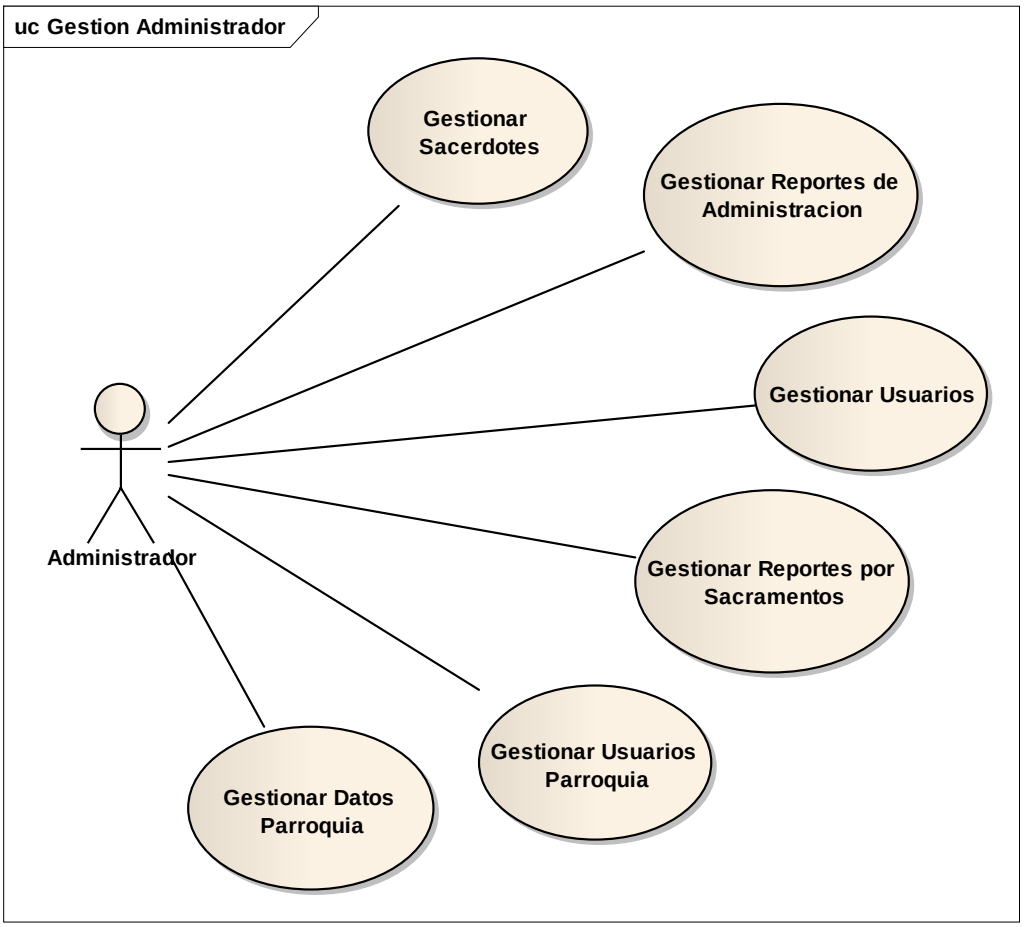


Figura Nº27. Modelo Casos de Usos Gestión Administrador

2.1.2.2.3.5.4.1.4 Modelo Casos de Uso Gestión Secretario

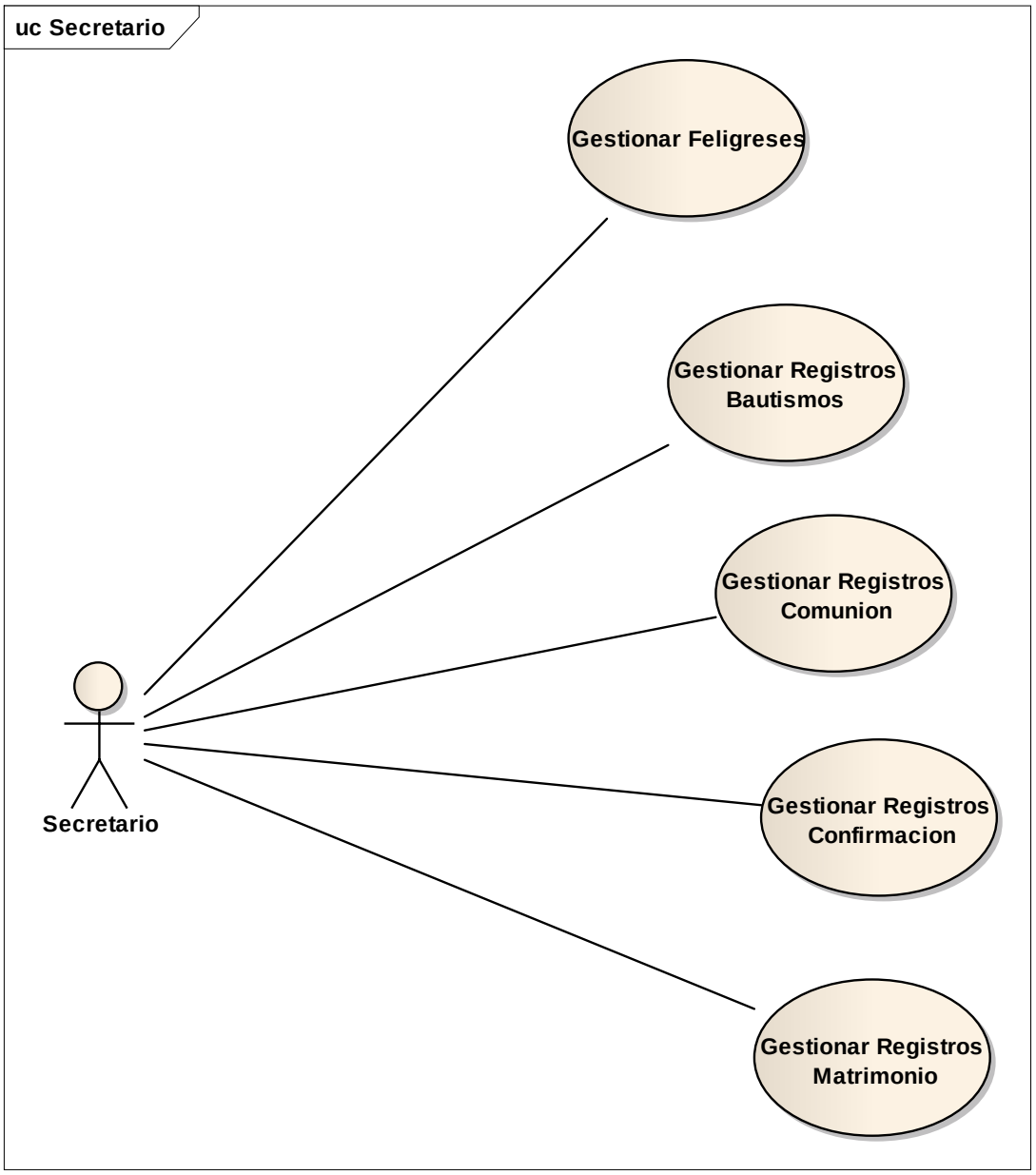


Figura Nº28. Modelo Casos de Uso Gestión Secretario

2.1.2.2.3.5.4.1.5 Modelo Casos de Uso Gestionar Usuarios

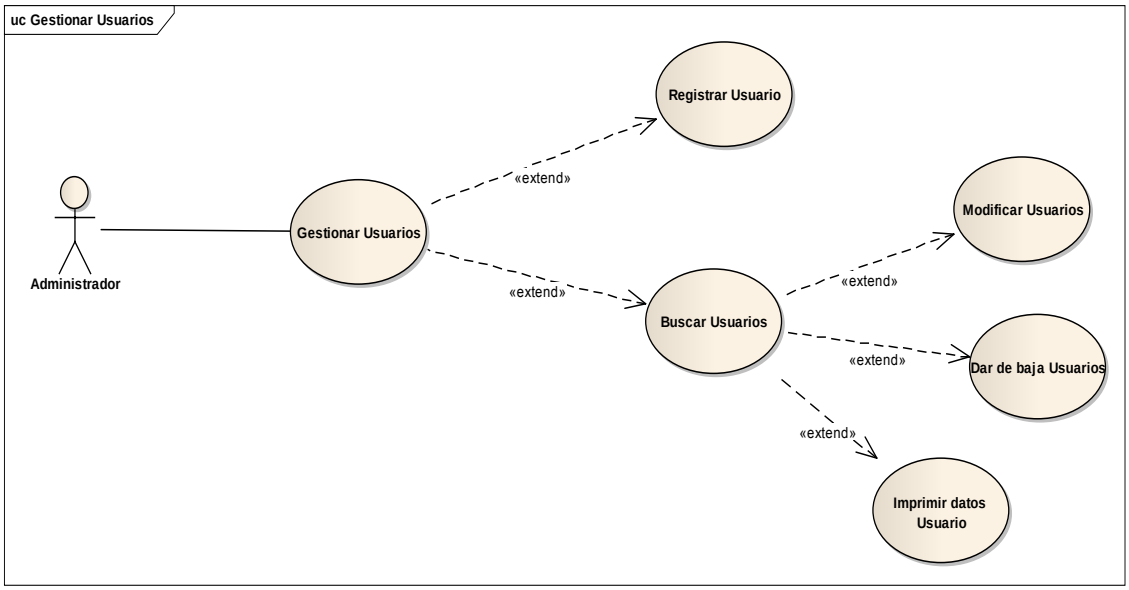


Figura Nº29. Modelo Casos de Uso Gestionar Usuario

2.1.2.2.3.5.4.1.6 Modelo Casos de Uso Gestionar Roles

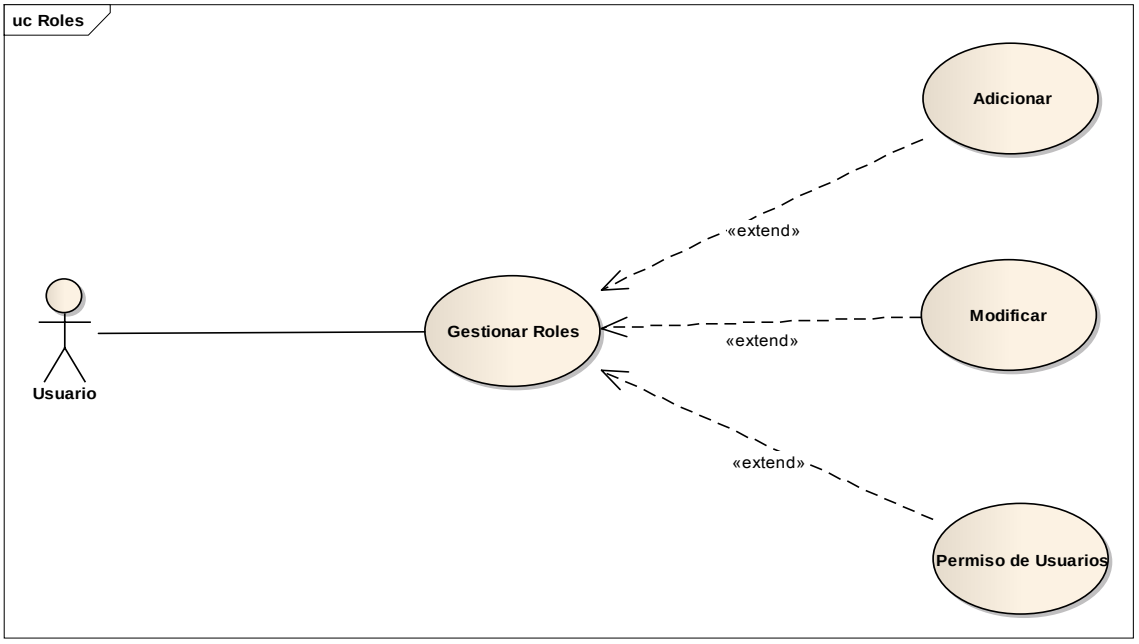


Figura Nº30. Modelo Casos de Uso Gestionar Roles

2.1.2.2.3.5.4.1.7 Modelo Casos de Uso Gestionar Sacerdotes

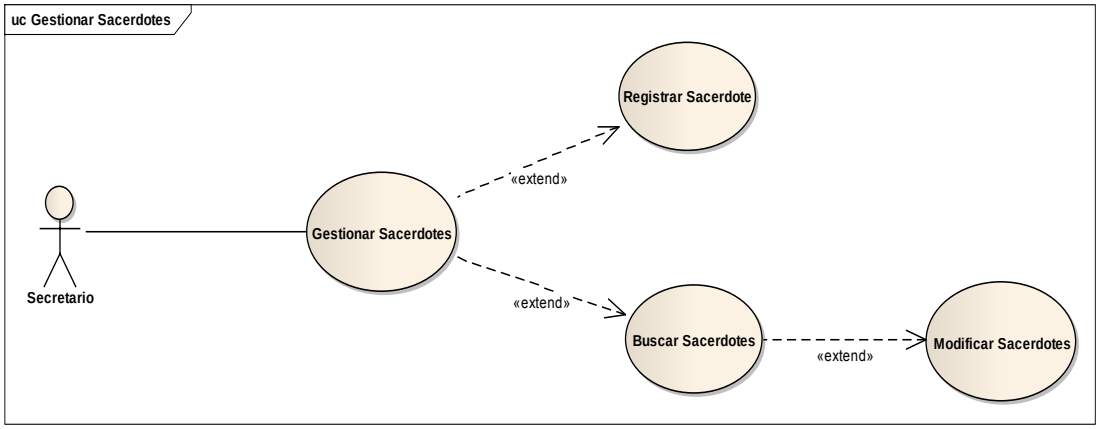


Figura Nº31. Modelo Casos de Uso Gestionar Sacerdotes

2.1.2.2.3.5.4.1.8 Modelo de Casos de Uso Administrar Backup

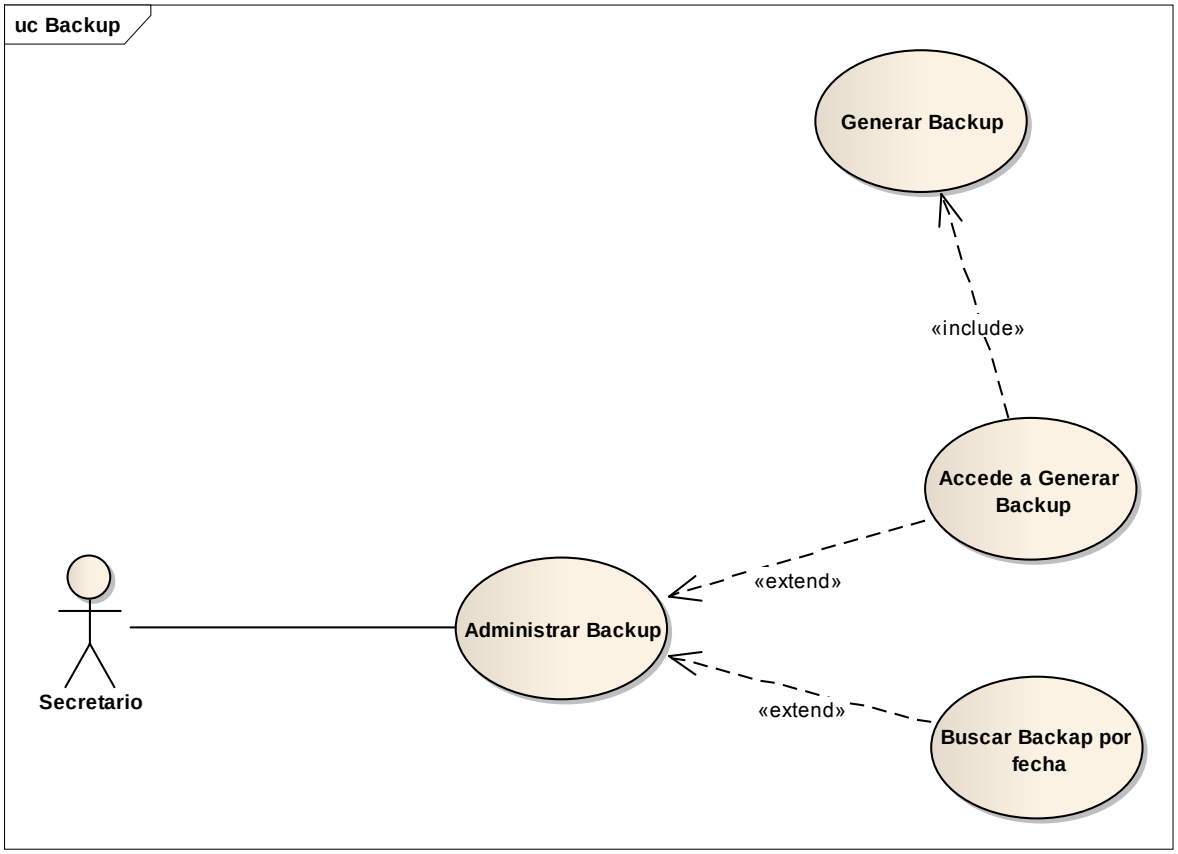


Figura Nº32. Modelo Casos de Uso Gestionar Backup

2.1.2.2.3.5.4.1.9 Modelo de Casos de Uso Gestionar Feligreses

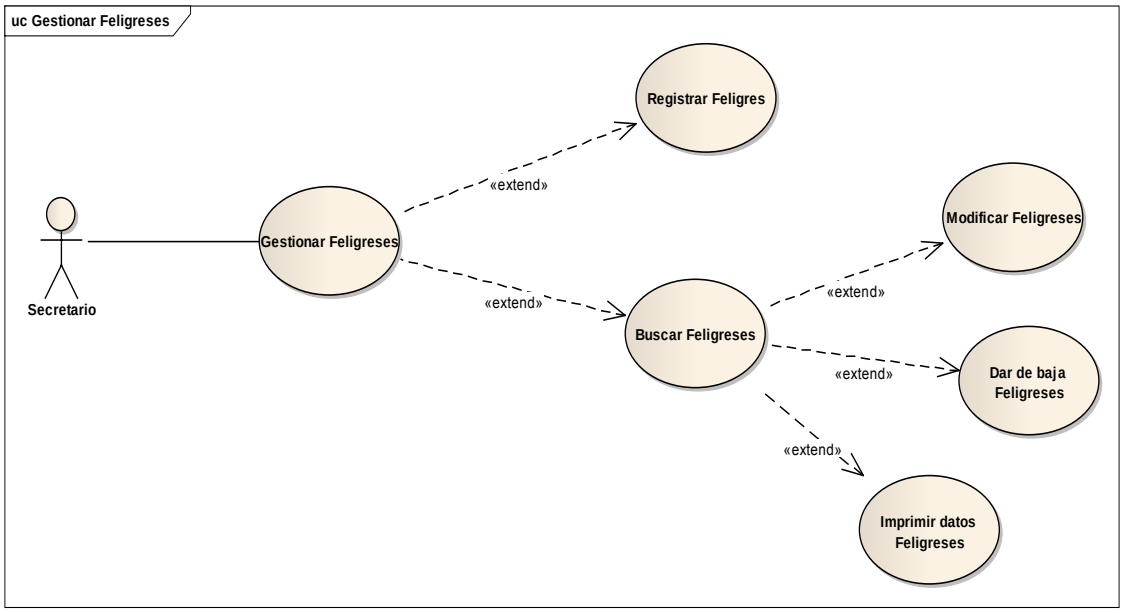


Figura Nº33. Modelo Casos de Uso Gestionar Feligreses

2.1.2.2.3.5.4.1.10 Modelo de Casos de Uso Gestionar Libros Sacramentales

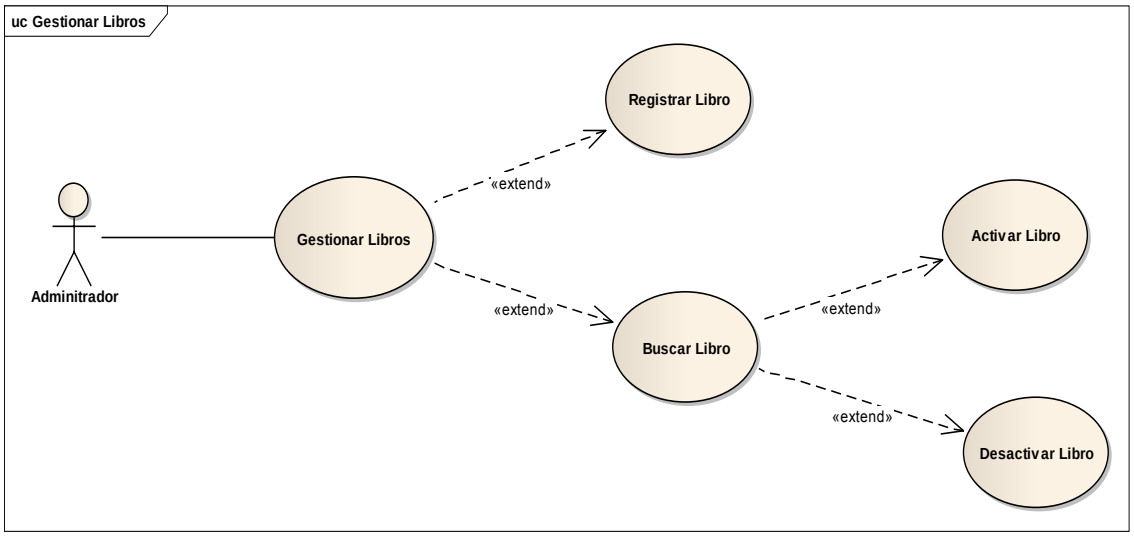


Figura Nº34. Modelo Casos de Uso Gestionar Libros Sacramentales

2.1.2.2.3.5.4.1.11 Modelo de Casos de Uso Gestionar Parroquias

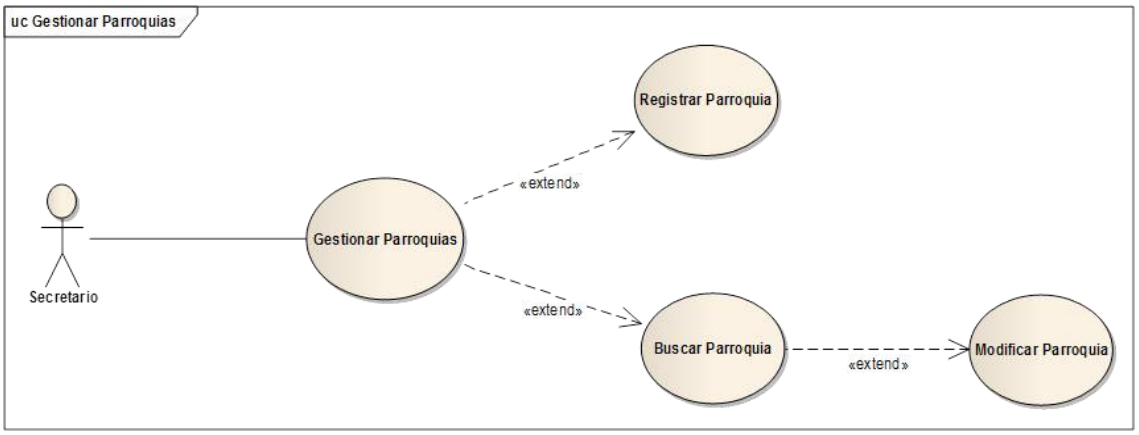


Figura Nº35. Modelo Casos de Uso Gestionar Parroquias

2.1.2.2.3.5.4.1.12 Modelo de Casos de Uso Gestionar Bautismo

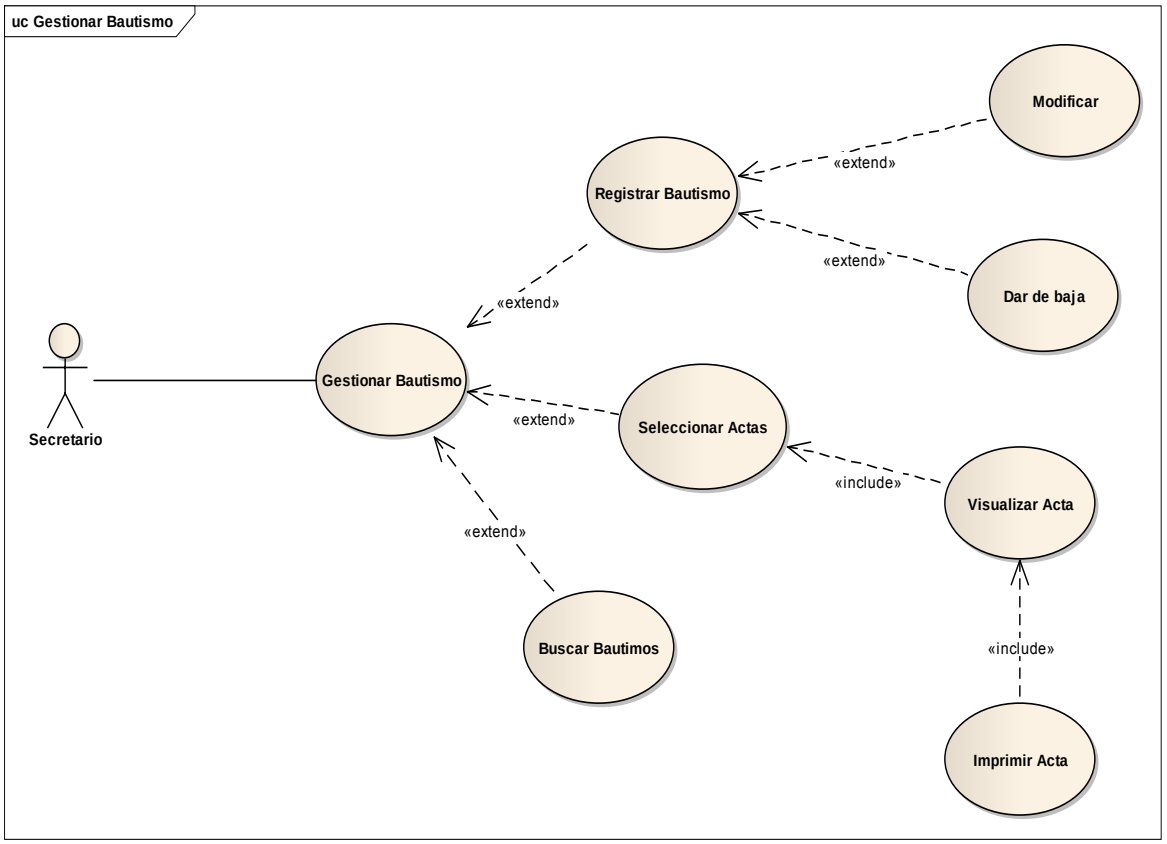


Figura Nº36. Modelo Casos de Uso Gestionar Bautismo

2.1.2.2.3.5.4.1.13 Modelo de Casos de Uso Emitir Certificado de Bautismo

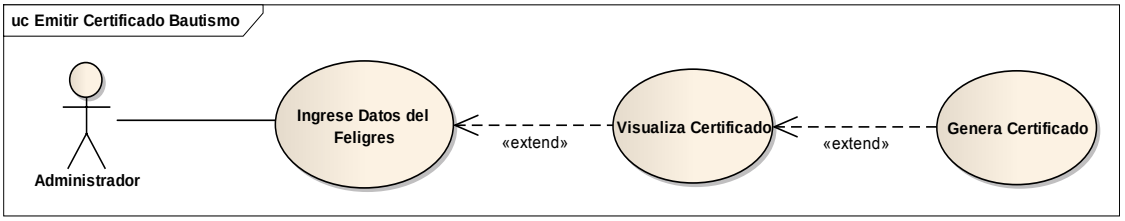


Figura Nº37. Modelo Casos de Uso Emitir Certificado de Bautismo

2.1.2.2.3.5.4.1.14 Modelo de Casos de Uso Gestionar Comunión

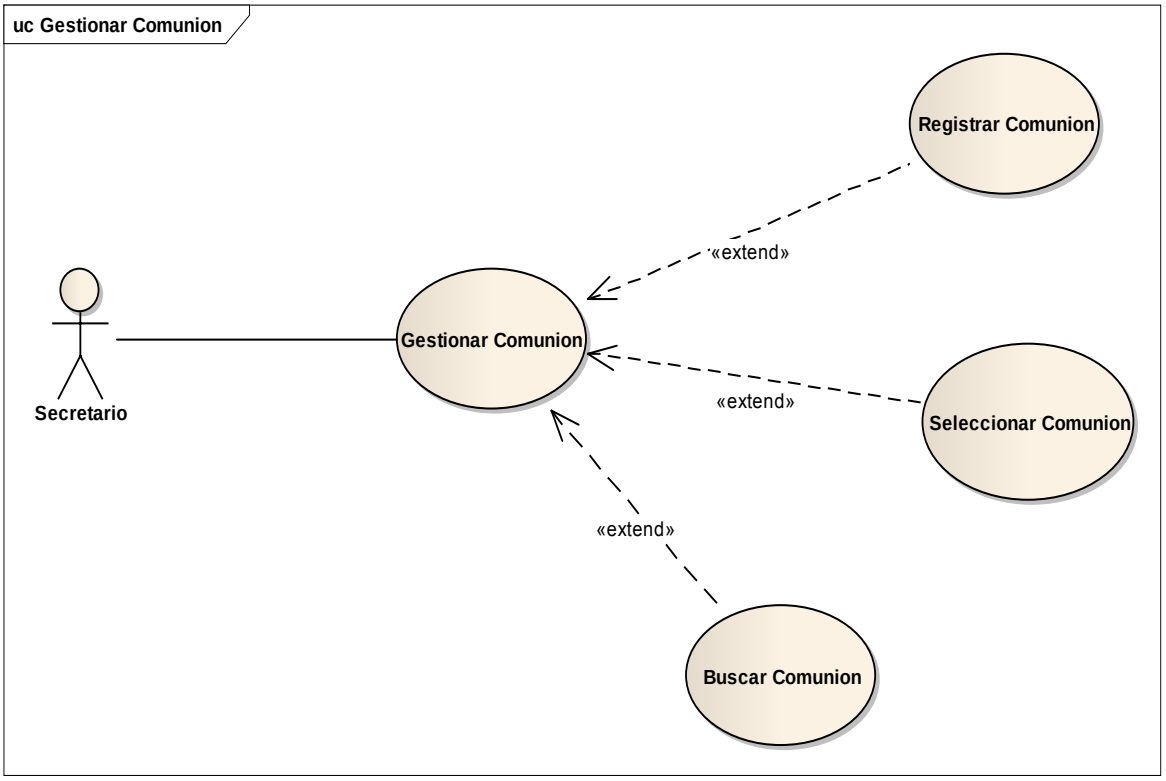


Figura Nº38. Modelo Casos de Uso Gestionar Comunión

2.1.2.2.3.5.4.1.15 Modelo de Casos de Uso Gestionar Confirmación

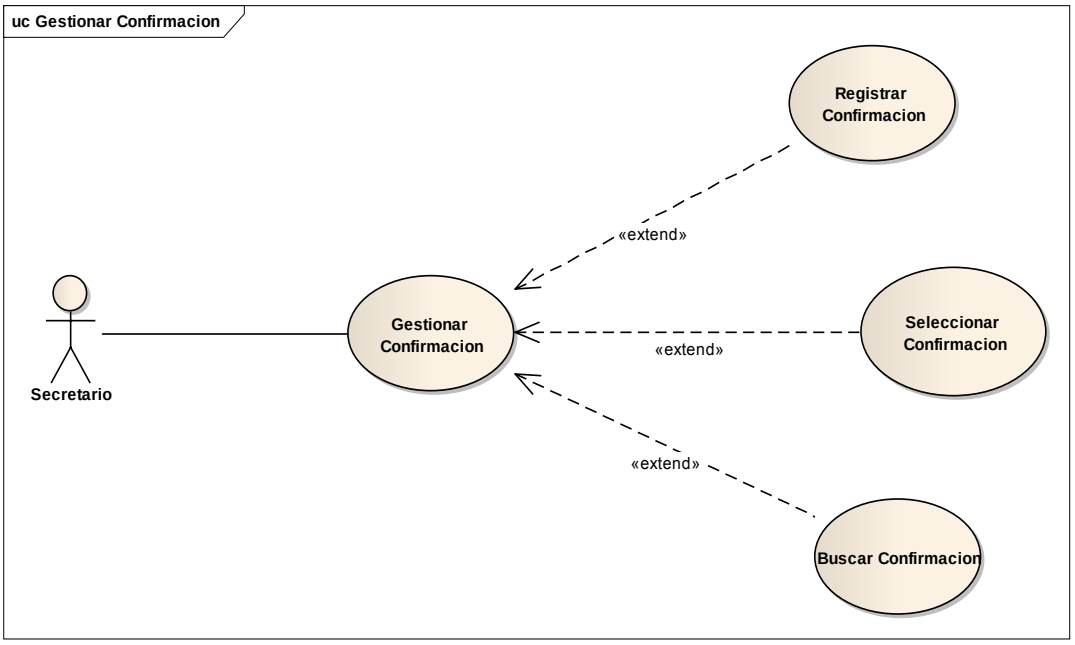


Figura Nº39. Modelo Casos de Uso Gestionar Confirmación

2.1.2.2.3.5.4.1.16 Modelo de Casos de Uso Gestionar Matrimonio

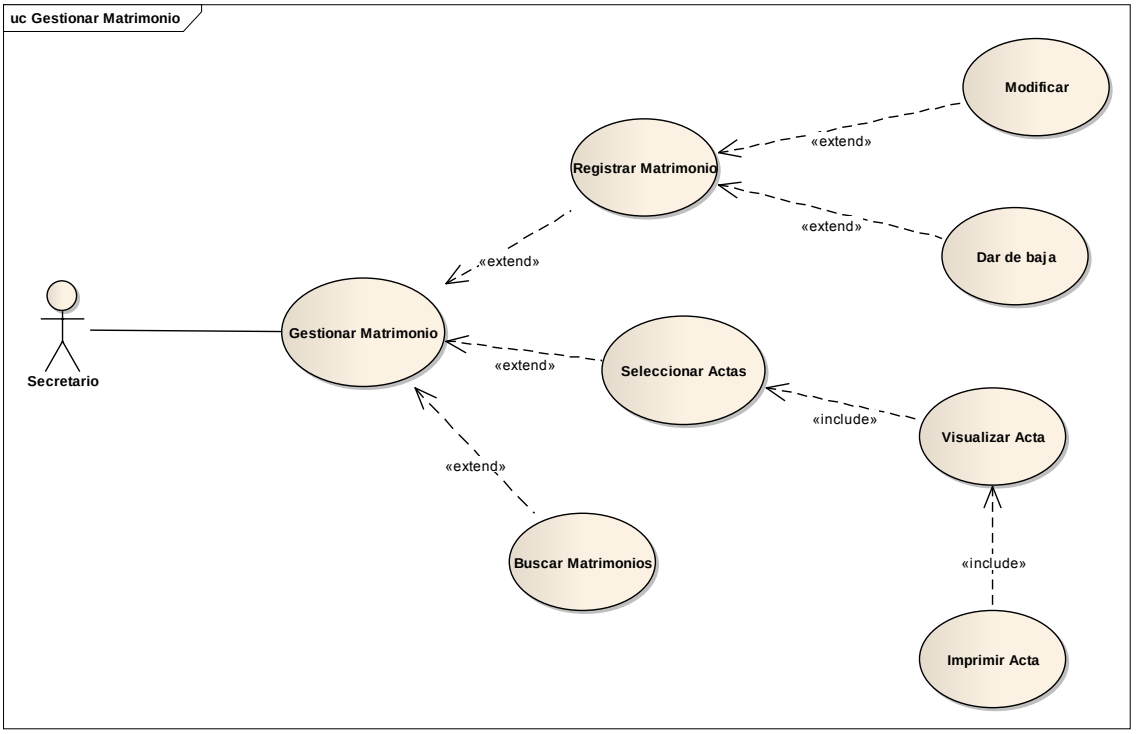


Figura Nº40. Modelo Casos de Uso Gestionar Matrimonio

2.1.2.2.3.5.4.1.17 Modelo de Casos de Uso Emitir Certificado de Matrimonio

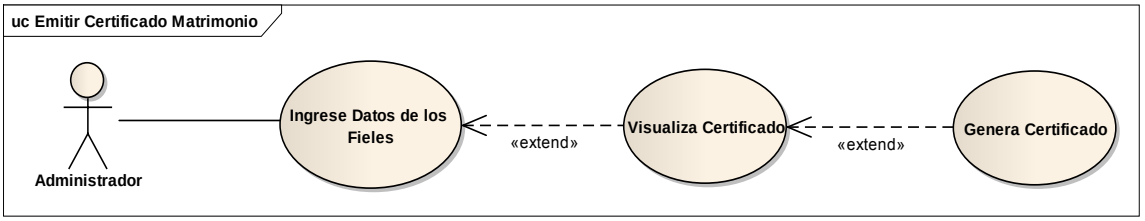


Figura Nº41. Modelo Casos de Uso Emitir Certificado de Matrimonio

2.1.2.2.3.5.4.1.18 Modelo de Casos de Uso Gestionar Reportes

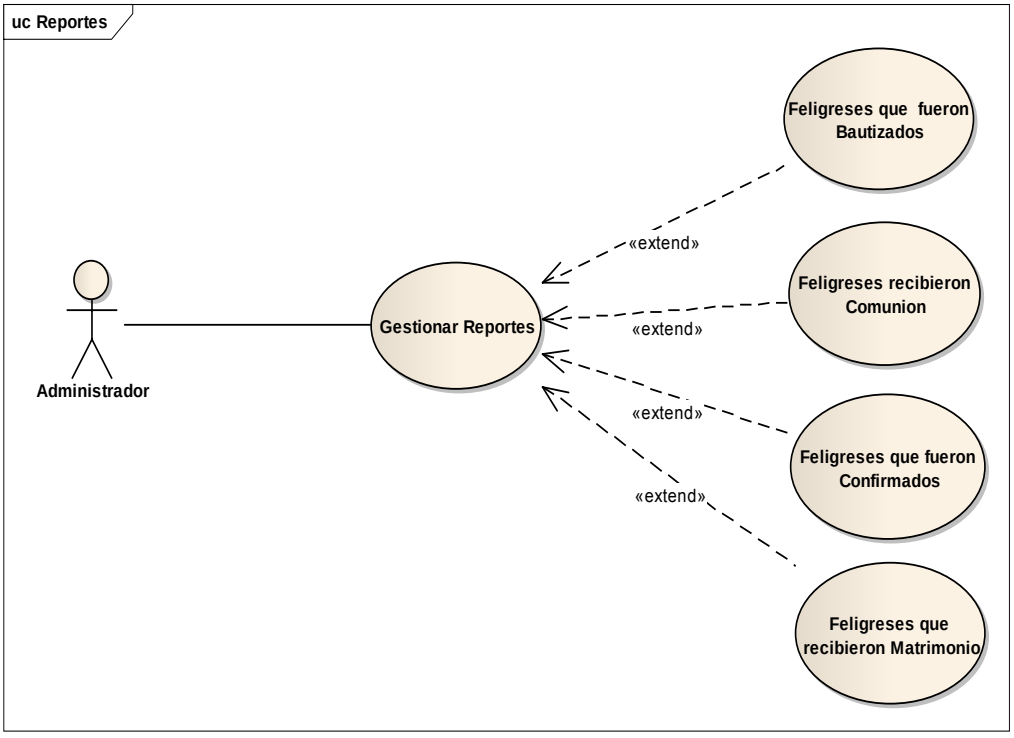


Figura Nº42. Modelo Casos de Uso Gestionar Reportes

2.1.2.2.3.5.4.1.19 Modelo de Casos de Uso Reportes Bautismos

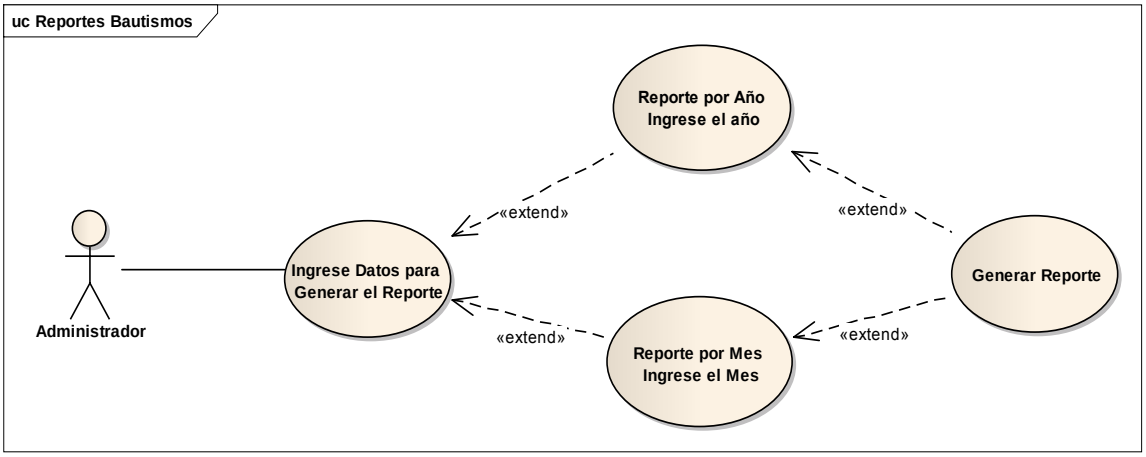


Figura Nº43. Modelo Casos de Uso Reportes Bautismos

2.1.2.2.3.5.4.1.20 Modelo de Casos de Uso Reportes Comunión

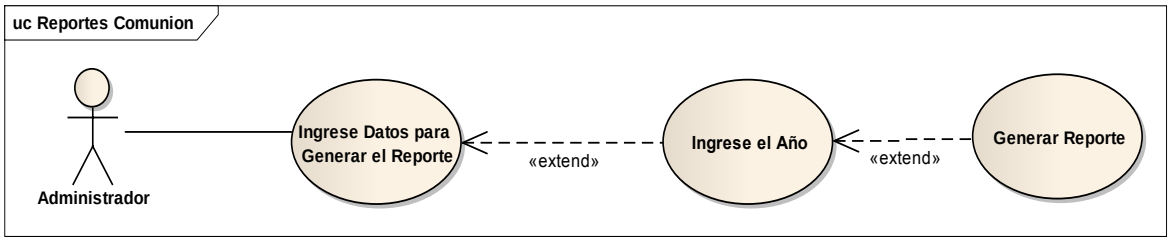


Figura Nº44. Modelo Casos de Uso Reportes Comunión

2.1.2.2.3.5.4.1.21 Modelo de Casos de Uso Reportes Confirmación

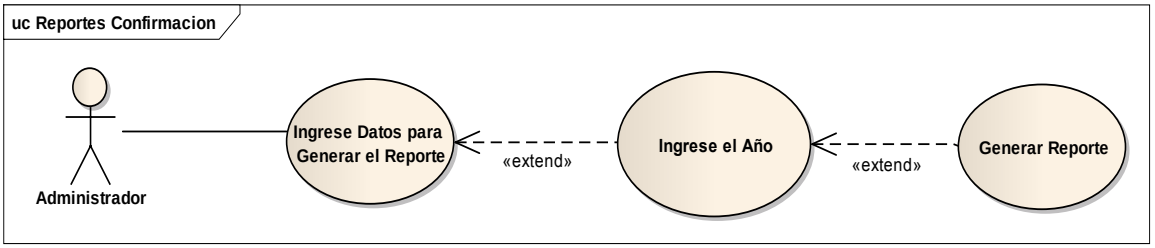


Figura Nº45. Modelo Casos de Uso Reportes Confirmación

2.1.2.2.3.5.4.1.22 Modelo de Casos de Uso Reportes Matrimonio

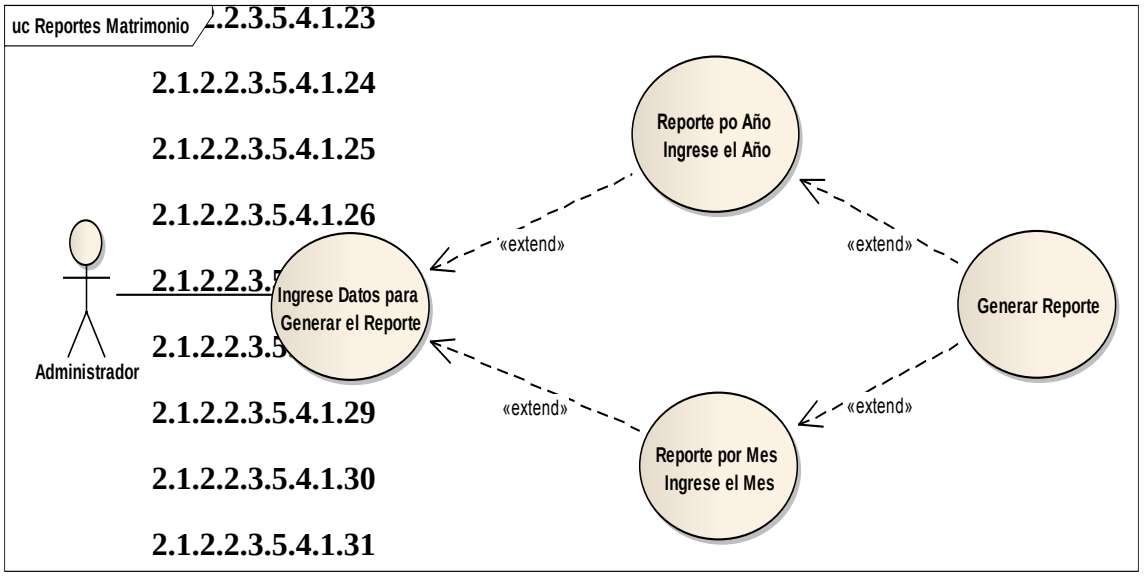


Figura Nº46. Modelo Casos de Uso Reportes Matrimonio

2.1.2.2.3.6 Visión

Este documento define la visión del producto desde la perspectiva del cliente, especificando las necesidades y características del producto. Constituye una base de acuerdo en cuanto a los requisitos del sistema.

2.1.2.2.3.6.1 Introducción

2.1.2.2.3.6.1.1 Propósito

El propósito de éste documento es recoger, definir y analizar las necesidades más importantes y las características del desarrollo del Sistema Informático para Mejoramiento de Afiliación, con el objetivo de llevar a cabo un producto que tome en cuenta las estrategias necesarias para poder brindar información valedera para el Usuario final.

2.1.2.2.3.6.1.2 Alcance

El presente documento se ocupa de reunir todas las necesidades del Usuario, para así poder diseñar un Sistema que satisfaga a las mismas como ser:

Brindar un medio de consultas de información acerca de los datos de los Feligreses que pertenecen a la Parroquia San Martin de Porres así como también los datos de los sacramentos que recibieron.

2.1.2.2.3.6.1.3 Definiciones, Acrónimos y Abreviaciones

RUP: Son las siglas de Rational Unified Process. Se trata de una metodología para describir el proceso de desarrollo de software.

UML: Son las siglas de Lenguaje Unificado de Modelación. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

COCOMO: Acrónimo en inglés de Modelo Constructivo de Costes**.** Es un modelo que permite realizar estimaciones y planificaciones de proyectos de sistemas de información.

LAN: Las siglas de Red de Área Local.

SQA: Acrónimo en inglés de Aseguramiento de la Calidad del Software.

2.1.2.2.3.6.2 Posicionamiento

2.1.2.2.3.6.2.1 Oportunidad de Negocio

Actualmente toda el registro que se tiene en la parroquia sobre los feligreses es almacenado en libros esto crea dificultades cuando tenemos que buscar información sobre los datos de los sacramentos que recibió algún feligrés y también se corre con el peligro de perder esta información, por esto es muy importante que los datos estén bien almacenados de ahí que se tiene la necesidad de implantar un sistema.

Los beneficios de la utilización de nuevas tecnologías de comunicación como ser el internet nos aseguran la integridad, veracidad y la organización de nuestra información.

Tanto el inadecuado proceso de organización de la información y registro manual de los datos de los feligreses, fueron motivos suficientes para elaborar el proyecto que permita el Mejorar la eficiencia en el cumplimiento de sacramentos en la Parroquia San Martin de Porres.

2.1.2.2.3.6.2.2 Sentencia que define el problema

El problema de	Inadecuado registro de datos de los sacramentos recibidos por un feligrés en la parroquia.
afecta a	Parroquia San Martin y Feligreses de la Parroquia.
El impacto asociado es	Insuficiencia de mecanismos que proporcionen información de los registros de los Feligreses.
Una solución adecuada sería	Diseñar un Sistema informático destinado a la Registro de los sacramentos recibidos por un feligrés en la parroquia.

Tabla Nº6. Sentencia que define el problema

2.1.2.2.3.6.2.3 Sentencia que define la posición del producto

Para	La Parroquia San Martin de Porres de Tarija.
Quienes	Feligrés de la Parroquia, Secretario de la Parroquia y Párroco.
El nombre del producto	**"SIGES" Sistema Informático de Gestión Sacramental.**
Que	Contribuir a mejorar la calidad de servicios a los feligreses de la parroquia San Martin de Porres de Tarija.
Nuestro producto	Permitir gestionar los sacramentos que se reciben en la parroquia mediante una interfaz gráfica sencilla y amigable, proporcionar un acceso rápido y actualizado a la base de datos.

Tabla Nº7. Sentencia que define la posición del producto

2.1.2.2.3.6.3 Descripción de los Stakeholders (Participantes del proyecto) y Usuarios

Para proveer de una forma efectiva productos y servicios que se ajusten a las necesidades de los usuarios, es necesario identificar e involucrar a todos los participantes en el proyecto como parte del proceso de modelado de requerimientos. También es necesario identificar a los usuarios del sistema y asegurarse de que el conjunto de participantes en el proyecto los representa adecuadamente. Esta sección muestra un perfil de los participantes y de los usuarios involucrados en el proyecto, así como los problemas más importantes que éstos perciben para enfocar la solución propuesta hacia ellos. No describe sus requisitos específicos ya que éstos se capturan mediante otro artefacto. En lugar de esto proporciona la justificación de por qué estos requisitos son necesarios.

2.1.2.2.3.6.3.1 Perfil de los Participantes

2.1.2.2.3.6.3.1.1 Universidad Autónoma Juan Misael Saracho

Representante	Universidad Autónoma Juan Misael Saracho.
Descripción	Entidad que asesora el proyecto.
Tipo	Proveedor.
Responsabilidades	Seguimiento del desarrollo del proyecto. Aprueba requisitos y funcionalidades.
Grado de participación	Velar que el proyecto sea ejecutado.

Tabla Nº8. Universidad Autónoma Juan Misael Saracho

2.1.2.2.3.6.3.1.2 Jefe del Proyecto

Representante	Jefe del Proyecto
Descripción	Grupo desarrollador del software.
Tipo	Desarrolladores.
Responsabilidades	Provee los equipos para el desarrollo tanto del análisis, diseño y programación del software. Diseña las posibles interfaces finales a ser implementadas en el sistema. Programa el software de acuerdo a los requisitos del sistema. Ejecuta y realiza reiteradas pruebas al sistema en su posible versión final.
Grado de participación	Encargados del análisis, diseño y desarrollo del sistema.

Tabla Nº9. Jefe de Proyecto

2.1.2.2.3.6.3.1.3 Parroquia San Martin de Porres

Representante	P. Jorge Machicao
Descripción	Párroco de la Parroquia.
Tipo	Administrador.
Responsabilidades	Encargado de la administración de la Parroquia, de realizar la celebración de los sacramentos y de proporcionar la información requerida por los feligreses.
Grado de participación	Velar por que se cumplan requisitos y funcionalidades, y que el proyecto sea ejecutado.

Tabla Nº10. Parroquia San Martin de Porres.

2.1.2.2.3.6.3.1.4 Docente

Representante	Docente
Descripción	Ente regulador universitario.
Tipo	Guía de desarrollo.
Responsabilidades	Realiza un control paso a paso del desarrollo del proyecto. Aprueba los distintos puntos tratados en el proyecto. Analiza los distintos documentos presentados acerca del producto.
Grado de participación	Guiar y evaluar el análisis, diseño y desarrollo del sistema.

Tabla Nº11. Docente

2.1.2.2.3.6.3.2 Perfil de Usuario

2.1.2.2.3.6.3.2.1 Administrador

Representante	Administrador
Descripción	Administrador de los recursos del sistema y del módulo Administrar.
Tipo	Usuario.
Responsabilidades	Responsable del control de roles y acceso, usuarios del sistema; además es el responsable de resguardar y restaurar la base de datos del sistema.
Grado de participación	Usar el Sistema, buscar debilidades del mismo.

Tabla Nº12. Administrador

2.1.2.2.3.6.3.2.2 Secretario

Representante	Secretario.
Descripción	Usuario independiente y directo, con ciertos atributos-permisos de ejecución de la aplicación.
Tipo	Secretario.
Responsabilidades	Hacer uso adecuado del sistema velar por la integridad del mismo, respetando las funcionalidades del mismo.
Grado de participación	Usar el Sistema, buscar debilidades del mismo.

Tabla Nº13. Secretario

2.1.2.2.3.6.3.2.3 Usuario Registrado

Representante	Operador.
Descripción	Usuario independiente y directo, con ciertos atributos-permisos de ejecución de la aplicación.
Tipo	Usuario causal.
Responsabilidades	Hacer uso adecuado del sistema velar por la integridad del mismo, respetando las funcionalidades del mismo.
Grado de participación	Usar el Sistema, buscar debilidades del mismo.

Tabla Nº14. Usuario Registrado

2.1.2.2.3.6.3.3 Entorno del Usuario

El secretario de la parroquia San Martin de Porres debe acceder al Sitio Web del Sistema donde podrá hacer Uso de Información del mismo, en el cual el Usuario debe estar registrado, para poder obtener algunas características adicionales-atributos-permisos. Si es un Administrador debe acceder al Programa instalado donde debe estar registrado como administrador para poder realizar la gestión de todo el Sistema.

2.1.2.2.3.6.4 Descripción Global del Producto

Debido a la creciente demanda de las nuevas tecnologías de información y comunicación y el impacto sociocultural que causan en nuestro entorno, elaboramos el proyecto “SIGES” Sistema Informático de Gestión Sacramental.

Los beneficios de la utilización de nuevas tecnologías de comunicación como ser el Internet nos aseguran la integridad, veracidad y la organización de nuestra información.

2.1.2.2.3.7 Especificaciones de Casos de Uso

Para los casos de uso que lo requieran (cuya funcionalidad no sea evidente o que no baste con una simple descripción narrativa) se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen: precondiciones, post-condiciones, flujo de eventos, requisitos no-funcionales asociados. También, para casos de uso cuyo flujo de eventos sea complejo podrá adjuntarse una representación gráfica mediante un Diagrama de Actividad.

2.1.2.2.3.7.1 Introducción

Las Especificaciones de los casos de Uso es una descripción detallada de los casos de uso del sistema.

2.1.2.2.3.7.2 Propósito

Interpretar y describir los casos de uso.

2.1.2.2.3.7.3 Alcance

Describe los procesos internos de los casos de uso.

Detalla los flujos de los casos de uso según lo establecido por la organización.

2.1.2.2.3.7.4 Especificación de los Casos de Uso

2.1.2.2.3.7.4.1 Especificación de Caso de Uso Acceder al Sistema

<table>
<tr><td>Caso de uso:</td><td colspan="2">Ingresar al sistema.</td></tr>
<tr><td colspan="3">Descripción:
Permite ingresar al sistema, este caso tiene como función controlar el acceso y al mismo tiempo recuperar los permisos correspondientes al momento que el usuario introduzca su usuario y clave en el sistema.</td></tr>
<tr><td colspan="3">Actores:
Administrador, Secretario de la Parroquia.</td></tr>
<tr><td colspan="3">Precondiciones:
Actor no tiene privilegios para realizar ninguna acción.</td></tr>
<tr><td>Flujo Normal:
1. El actor introduce datos de usuario y clave en el sistema.
2. El actor presiona botón Ingresar.
3. El sistema valida sus datos.
4. El actor espera su validación.
5. Si los datos son correctos muestra la Pantalla Principal de Administración.</td><td colspan="2">Flujo Alternativo:
3. Si el actor no llena el formulario de acceso, se muestra un mensaje donde se informa que los campos son requeridos.
3. Con más de 3 intentos fallidos el sistema deshabilita la opción de Ingresar.
5. Si el usuario no existe se muestra mensaje de error y vuelve a mostrar la pantalla índex.</td></tr>
<tr><td colspan="3">Post condiciones: Ninguno</td></tr>
</table>

Tabla Nº15. Especificación de Caso de Uso Acceder al Sistema

2.1.2.2.3.7.4.1.1 Especificación Caso de Uso Administrar Rol

<table>
<tr><td>Caso de uso:</td><td colspan="2">Administrar Rol.</td></tr>
<tr><td colspan="3">Descripción: Permite Listar, Adicionar, Modificar, Eliminar, Cambiar Estado, Asignar Funcionalidad.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El sistema muestra Listado de Roles registrados en el sistema.
2. El actor realiza la Búsqueda de Rol en el Listado.
3. El sistema muestra el(los) Roles que cumple(n) con los Criterios de Búsqueda.
4. El actor selecciona una fila del Listado y hace Clic derecho.
5. El actor selecciona la opción que requiere.
6. El sistema verifica que se haya seleccionado una fila del listado.
7. El sistema solicita la confirmación de la selección de la fila.</td><td>Flujo Alternativo:
6. El actor no ha seleccionado ninguna fila del Listado y el sistema muestra un mensaje de advertencia “Debe Seleccionar un Rol”.
7. El actor decide no confirmar selección y el sistema no realiza ninguna operación.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº16. Especificación Caso de Uso Administrar Rol

2.1.2.2.3.7.4.1.2 Especificación Caso de Uso Adicionar Rol

<table>
<tr><td>Caso de uso:</td><td colspan="2">Adicionar Rol.</td></tr>
<tr><td colspan="3">Descripción: Permite registrar nuevo Rol en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona botón Adicionar.
2. El sistema le muestra formulario.
3. El actor introduce datos del Rol en el formulario.
4. Selecciona botón Guardar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema guarda datos del formulario.
7. El sistema regresa a la pantalla de Administrar Rol.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Rol.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº17. Especificación Caso de Uso Adicionar Rol

2.1.2.2.3.7.4.1.3 Especificación Caso de Uso Modificar Rol

Caso de uso:	Modificar Rol.
Descripción: Permite modificar datos de un Rol en el sistema.	
Actores: Administrador.	
Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Rol.	
Flujo Normal: 1. El actor presiona en el menú la opción Modificar. 2. El sistema le muestra formulario con datos del Rol. 3. El actor realiza los cambios de datos del Rol en el formulario. 4. Selecciona botón Modificar. 5. Se valida datos del formulario que son obligatorios estén introducidos. 6. El sistema actualiza los datos del formulario. 7. El sistema regresa a la pantalla de Administrar Rol.	**Flujo Alternativo:** 4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Rol. 5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.
Pues condiciones: Ninguno	

Tabla Nº18. Especificación Caso de Uso Modificar Rol

2.1.2.2.3.7.4.1.4 Especificación Caso de Uso Eliminar Rol

Caso de uso:	Eliminar Rol.
Descripción: Permite eliminar del sistema un Rol.	
Actores: Administrador.	
Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Rol.	
Flujo Normal: 1. El actor presiona en el menú la opción Eliminar. 2. El sistema muestra el mensaje de confirmación. 3. Por Si, El sistema realiza la eliminación del Rol. 4. El sistema regresa a la pantalla de Administrar Rol.	**Flujo Alternativo:** 3. Por No, El sistema no realiza ninguna operación.
Pues condiciones: Eliminación realizada con éxito.	

Tabla Nº19. Especificación Caso de Uso Eliminar Rol

2.1.2.2.3.7.4.1.5 Especificación Caso de Uso Cambiar Estado Rol

<table>
<tr><td>Caso de uso:</td><td colspan="2">Cambiar Estado Rol.</td></tr>
<tr><td colspan="3">Descripción: Permite cambiar estado de un Rol en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Rol.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presión en el menú la opción Cambiar Estado.
2. El sistema muestra el mensaje de confirmación.
3. Por si, El sistema realiza el cambio de estado del Rol.
4. El sistema regresa a la pantalla de Administrar Rol.</td><td>Flujo Alternativo:
3. Por No, El sistema no realiza ninguna operación.</td></tr>
<tr><td colspan="3">Pues condiciones: Cambio de Estado realizado con éxito.</td></tr>
</table>

Tabla Nº20. Especificación Caso de Uso Cambiar Estado Rol

2.1.2.2.3.7.4.1.6 Especificación Caso de Uso Asignar Funcionalidad

<table>
<tr><td>Caso de uso:</td><td colspan="2">Asignar Funcionalidad.</td></tr>
<tr><td colspan="3">Descripción: Permite Listar, Adicionar, Modificar, Eliminar, Cambiar Estado.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El sistema muestra Listado de Funcionalidades Asignadas y no Asignadas al Rol registrados en el sistema.
2. El actor selecciona una o varias filas del Listado y hace Clic en adicionar y eliminar funcionalidad.
3. Selecciona botón Asignar.
4. El sistema guarda datos del formulario.
5. El sistema regresa a la pantalla de Administrar Rol.</td><td>Flujo Alternativo:
2. El actor no ha seleccionado ninguna fila del Listado y el sistema muestra un mensaje de advertencia “Por Favor Seleccione una Funcionalidad para Adicionar” o Por Favor Seleccione una Funcionalidad para Eliminar.
3. Selecciona botón Cancelar, se regresa a la pantalla Administrar Rol.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº21. Especificación Caso de Uso Asignar Funcionalidad

2.1.2.2.3.7.4.1.7 Especificación Caso de Uso Administrar Usuario

<table>
<tr><td>Caso de uso:</td><td colspan="2">Administrar Usuario.</td></tr>
<tr><td colspan="3">Descripción: Permite Listar, Adicionar, Modificar, Eliminar, Cambiar Estado, Asignar Usuario y Clave de Usuario y Lista Personas Que no Son Usuarios..</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El sistema muestra Listado de Usuarios registrados en el sistema.
2. El actor realiza la Búsqueda de Rol en el Listado.
3. El sistema muestra el(los) Usuario(s) que cumple(n) con los Criterios de Búsqueda.
4. El actor selecciona una fila del Listado y hace Clic derecho.
5. El actor selecciona la opción que requiere.
6. El sistema verifica que se haya seleccionado una fila del listado.
7. El sistema solicita la confirmación de la selección de la fila.</td><td>Flujo Alternativo:
6. El actor no ha seleccionado ninguna fila del Listado y el sistema muestra un mensaje de advertencia “Debe Seleccionar un Usuario”.
7. El actor decide no confirmar selección y el sistema no realiza ninguna operación.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº22. Especificación Caso de Uso Administrar Usuario

2.1.2.2.3.7.4.1.8 Especificación Caso de Uso Adicionar Usuario

<table>
<tr><td>Caso de uso:</td><td colspan="2">Adicionar Usuario.</td></tr>
<tr><td colspan="3">Descripción: Permite registrar nuevo Usuario en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona botón Adicionar.
2. El sistema le muestra formulario.
3. El actor introduce datos del Usuario (que es una Persona) en el formulario.
4. Selecciona botón Guardar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema guarda datos del formulario.
7. El sistema regresa a la pantalla de Administrar Usuario.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Usuario.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº23. Especificación Caso de Uso Adicionar Usuario

2.1.2.2.3.7.4.1.9 Especificación Caso de Uso Modificar Usuario

<table>
<tr><td>Caso de uso:</td><td colspan="2">Modificar Usuario.</td></tr>
<tr><td colspan="3">Descripción: Permite modificar datos de un Usuario en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Usuario.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona en el menú la opción Modificar.
2. El sistema le muestra formulario con datos del Usuario.
3. El actor realiza los cambios de datos del Usuario en el formulario.
4. Selecciona botón Modificar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema actualiza los datos del formulario.
7. El sistema regresa a la pantalla de Administrar Usuario.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Usuario.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº24. Especificación Caso de Uso Modificar Usuario

2.1.2.2.3.7.4.1.10 Especificación Caso de Uso Eliminar Usuario

<table>
<tr><td>Caso de uso:</td><td colspan="2">Eliminar Usuario.</td></tr>
<tr><td colspan="3">Descripción: Permite eliminar del sistema un Usuario.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Usuario.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona en el menú la opción Eliminar.
2. El sistema muestra el mensaje de confirmación.
3. Por Si, El sistema realiza la eliminación del Usuario.
4. El sistema regresa a la pantalla de Administrar Usuario.</td><td>Flujo Alternativo:
3. Por No, El sistema no realiza ninguna operación.</td></tr>
<tr><td colspan="3">Pues condiciones: Eliminación realizada con éxito.</td></tr>
</table>

Tabla Nº25. Especificación Caso de Uso Eliminar Usuario

2.1.2.2.3.7.4.1.11 Especificación Caso de Uso Cambiar Estado Usuario

Caso de uso:	Cambiar Estado Usuario.
Descripción: Permite cambiar estado de un Usuario en el sistema.	
Actores: Administrador.	
Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Usuario.	
Flujo Normal: 1. El actor presión en el menú la opción Cambiar Estado. 2. El sistema muestra el mensaje de confirmación. 3. Por si, El sistema realiza el cambio de estado del Usuario. 4. El sistema regresa a la pantalla de Administrar Usuario.	**Flujo Alternativo:** 3. Por No, El sistema no realiza ninguna operación.
Pues condiciones: Cambio de Estado realizado con éxito.	

Tabla Nº26. Especificación Caso de Uso Cambiar Estado Usuario

2.1.2.2.3.7.4.1.12 Especificación Caso de Uso Asignar Usuario y Clave

<table>
<tr><td>Caso de uso:</td><td colspan="2">Asignar Usuario y Clave.</td></tr>
<tr><td colspan="3">Descripción: Permite registrar un Usuario.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona botón Asignar Usuario y Clave.
2. El sistema le muestra formulario.
3. El actor introduce datos del Usuario en el formulario.
4. Selecciona botón Guardar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema guarda datos del formulario.
1. El sistema regresa a la pantalla de Administrar Usuario.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Usuario.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno.</td></tr>
</table>

Tabla Nº27. Especificación Caso de Uso Asignar Usuario y Clave

2.1.2.2.3.7.4.1.13 Especificación de Casos de Uso Administrar Backup

<table>
<tr><td>Caso de uso:</td><td colspan="2">Administrar Backup.</td></tr>
<tr><td colspan="3">Descripción: Permite Listar, Resguardar, Restaurar, Eliminar, Cambiar Estado y Generar Reporte PDF de Backup.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El sistema muestra Listado de Backup’s registrados en el sistema.
2. El actor realiza la Búsqueda de Backup en el Listado.
3. El sistema muestra (los) Backup(s) que cumple(n) con los Criterios de Búsqueda.
4. El actor selecciona una fila del Listado y hace Clic derecho.
5. El actor selecciona la opción que requiere.
6. El sistema verifica que se halla seleccionado una fila del listado.
7. El sistema solicita la confirmación de la selección de la fila.</td><td>Flujo Alternativo:
6. El actor no ha seleccionado ninguna fila del Listado y el sistema muestra un mensaje de advertencia “Debe Seleccionar un Backup”.
7. El actor decide no confirmar selección y el sistema no realiza ninguna operación.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº28. Especificación Caso de Uso Administrar Backup

2.1.2.2.3.7.4.1.14 Especificación Caso de Uso Resguardar Backup

<table>
<tr><td>Caso de uso:</td><td colspan="2">Resguardar Backup.</td></tr>
<tr><td colspan="3">Descripción: Permite resguardar un nueva Backup en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona botón Resguardar.
2. El sistema le muestra formulario.
3. El actor introduce datos del Backup en el formulario.
4. Selecciona botón Resguardar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema guarda datos del formulario.
7. El sistema regresa a la pantalla de Administrar Backup.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Backup.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº29. Especificación Caso de Uso Resguardar Backup

2.1.2.2.3.7.4.1.15 Especificación Caso de Uso Restaurar Backup

<table>
<tr><td>Caso de uso:</td><td colspan="2">Restaurar Backup.</td></tr>
<tr><td colspan="3">Descripción: Permite restaurar un Backup en el sistema.</td></tr>
<tr><td colspan="3">Actores: Administrador.</td></tr>
<tr><td colspan="3">Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Backup.</td></tr>
<tr><td colspan="2">Flujo Normal:
1. El actor presiona en el menú la opción Restaurar.
2. El sistema le muestra formulario con datos del Backup.
3. El actor realiza la introducción de datos del Backup en el formulario.
4. Selecciona botón Restaurar.
5. Se valida datos del formulario que son obligatorios estén introducidos.
6. El sistema guarda datos del formulario.
7. El sistema regresa a la pantalla de Administrar Backup.</td><td>Flujo Alternativo:
4. Selecciona botón Cancelar, se regresa a la pantalla Administrar Backup.
5. Se muestra un mensaje donde se informa que los campos son requeridos y se vuelve al paso 3 para que el actor pueda ingresar la información que falta.</td></tr>
<tr><td colspan="3">Pues condiciones: Ninguno</td></tr>
</table>

Tabla Nº30. Especificación Caso de Uso Restaurar Backup

2.1.2.2.3.7.4.1.16 Especificación Caso de Uso Eliminar Backup

Caso de uso:	Eliminar Backup.
Descripción: Permite eliminar del sistema una Backup.	
Actores: Administrador.	
Precondiciones: Actor haber iniciado sesión; Estar registrado en el sistema el Backup.	
Flujo Normal: 1. El actor presiona en el menú la opción Eliminar. 2. El sistema muestra el mensaje de confirmación. 3. Por Si, El sistema realiza la eliminación del Backup. 4. El sistema regresa a la pantalla de Administrar Backup.	**Flujo Alternativo:** 3. Por No, El sistema no realiza ninguna operación.
Pues condiciones: Eliminación realizada con éxito.	

Tabla Nº31. Especificación Caso de Uso Eliminar Backup

2.1.2.2.3.7.4.1.17 Especificación Caso de Uso Generar Reporte Lista Backup

Caso de uso:	Generar Reporte Lista Backup.
Descripción: Permite visualizar reporte.	
Actores: Administrador.	
Precondiciones: Actor haber iniciado sesión.	
Flujo Normal: ***Figura Nº1.*** El actor presiona en el menú la opción Reporte. ***Figura Nº2.*** El sistema muestra el Reporte.	**Flujo Alternativo:**
Pues condiciones:	

Tabla Nº32. Especificación Caso de Uso Generar Reporte Lista Backup

2.1.2.2.3.8 Especificaciones Adicionales

Este documento capturará todos los requisitos que no han sido incluidos como parte de los casos de uso y se refieren requisitos no-funcionales globales. Dichos requisitos incluyen: requisitos legales o normas, aplicación de estándares, requisitos de calidad del producto, tales como: confiabilidad, desempeño, etc., u otros requisitos de ambiente, tales como: sistema operativo, requisitos de compatibilidad, etc.

2.1.2.2.3.9 Prototipos de Interfaces de Usuario

2.1.2.2.3.9.1 Introducción

Se trata de prototipos que permiten al usuario hacerse una idea más o menos precisa de las interfaces que proveerá el sistema y así, conseguir retroalimentación de su parte respecto a los requisitos del sistema. Estos prototipos se realizarán como: dibujos a mano en papel, dibujos con alguna herramienta gráfica o prototipos ejecutables interactivos, siguiendo ese orden de acuerdo al avance del proyecto. Sólo los de este último tipo serán entregados al final de la fase de Elaboración, los otros serán desechados. Asimismo, este artefacto, será desechado en la fase de Construcción en la medida que el resultado de las iteraciones vayan desarrollando el producto final.

2.1.2.2.3.9.2 Propósito

Presentar los prototipos de pantallas para que el usuario tenga una idea de la interfaz que se presentaran en el Sistema.

2.1.2.2.3.9.3 Alcance

Mostrar los Prototipos de Pantallas, sujeto a modificaciones a lo largo del desarrollo del Sistema.

2.1.2.2.3.9.1 Diseño de Pantallas

2.1.2.2.3.9.1.1 Pantalla Acceder al Sistema

Figura Nº47. Pantalla Acceder al Sistema

Para acceder al sistema el usuario debe introducir su: nombre de usuario y clave.

El sistema validara los datos ingresados y desplegará la Menú Principal.

2.1.2.2.3.9.1.2 Menú Principal

Figura Nº48. Pantalla Principal

Esta es la pantalla menú principal en el cual se pueden observar los módulos con los que cuenta el Sistema: Libros, Sacerdotes, Feligreses, Parroquias, Bautismos, Comuniones, Confirmaciones, Matrimonios, Usuarios, Reportes y Resguardos.

2.1.2.2.3.9.1.3 Pantalla Gestionar Usuarios

Figura Nº49. Pantalla Gestionar Usuarios

En esta pantalla se lista a los usuarios del sistema.

2.1.2.2.3.9.1.4 Pantalla Registrar Nuevo Usuario

Figura Nº50. Pantalla Registrar Nuevo Usuario

En esta pantalla se realiza el registro de un nuevo usuario del Sistema.

2.1.2.2.3.9.1.5 Pantalla Modificar Datos Usuario

Figura Nº51. Pantalla Modificar Datos Usuario

En esta pantalla se modifican los datos de algún usuario del Sistema.

2.1.2.2.3.9.1.6 Pantalla Dar de baja Usuario

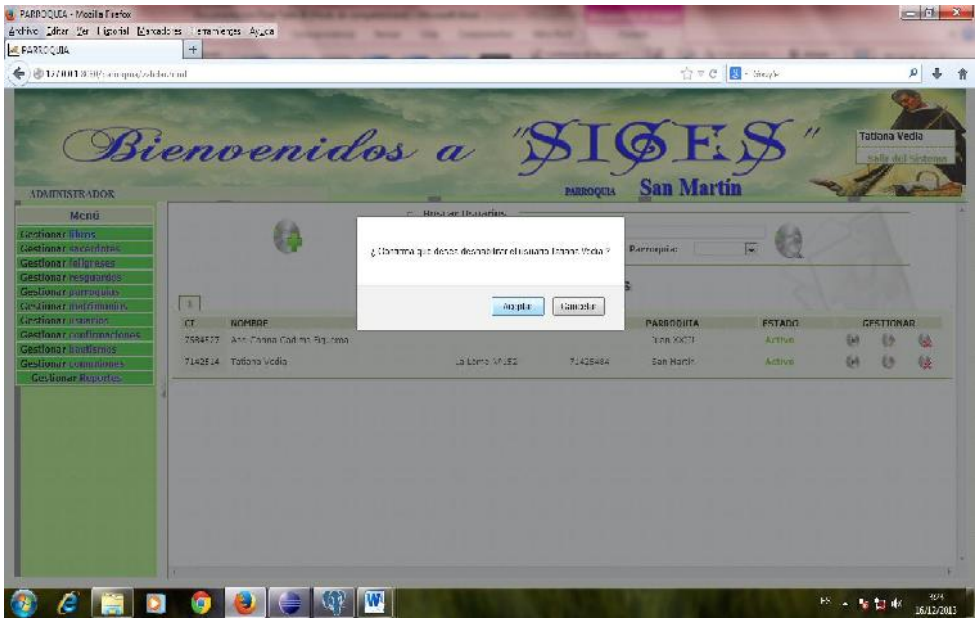


Figura Nº52. Pantalla Dar de baja Usuario

En esta pantalla se muestra la opción deshabilitar a un Usuario del Sistema.

2.1.2.2.3.9.1.7 Pantalla Buscar Usuarios

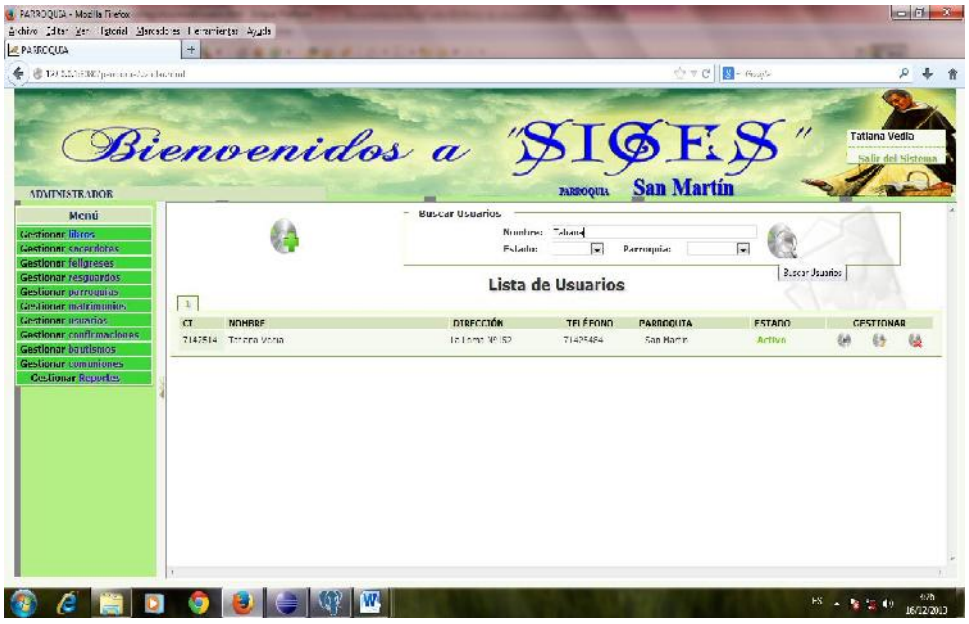


Figura Nº53. Pantalla Buscar Usuarios

En esta pantalla se muestra la opción buscar Usuarios del Sistema.

2.1.2.2.3.9.1.8 Pantalla Resguardo Base de Datos

Figura Nº54. Pantalla Resguardo Base de Datos

En esta pantalla se lista los resguardos de la base de datos.

2.1.2.2.3.9.1.9 Pantalla Generar Resguardo Base de Datos

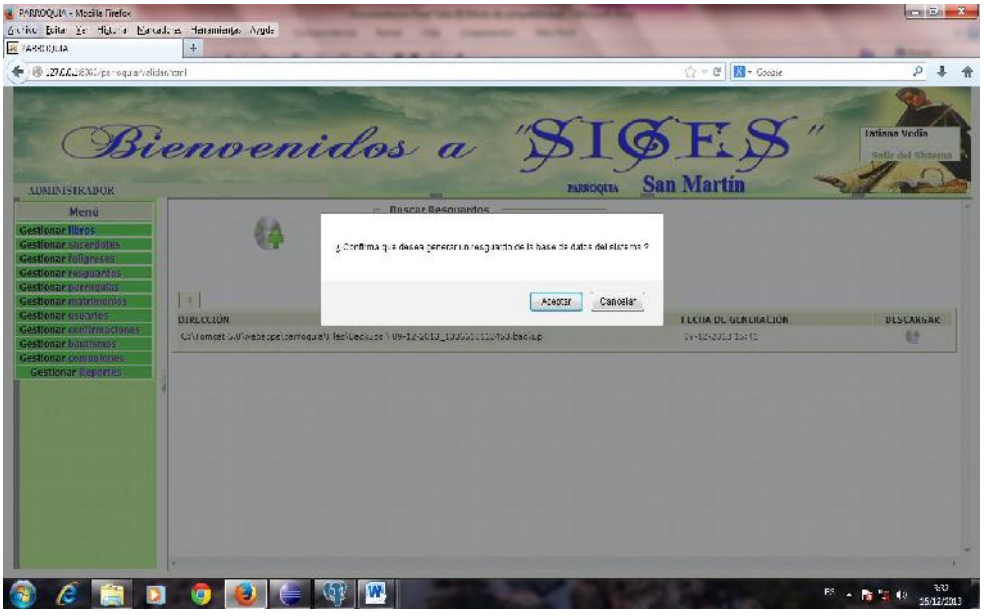


Figura Nº55. Pantalla Generar Resguardo Base de Datos

En esta pantalla se muestra como se genera un resguardo de la base de datos.

2.1.2.2.3.9.1.10 Pantalla Buscar Resguardos

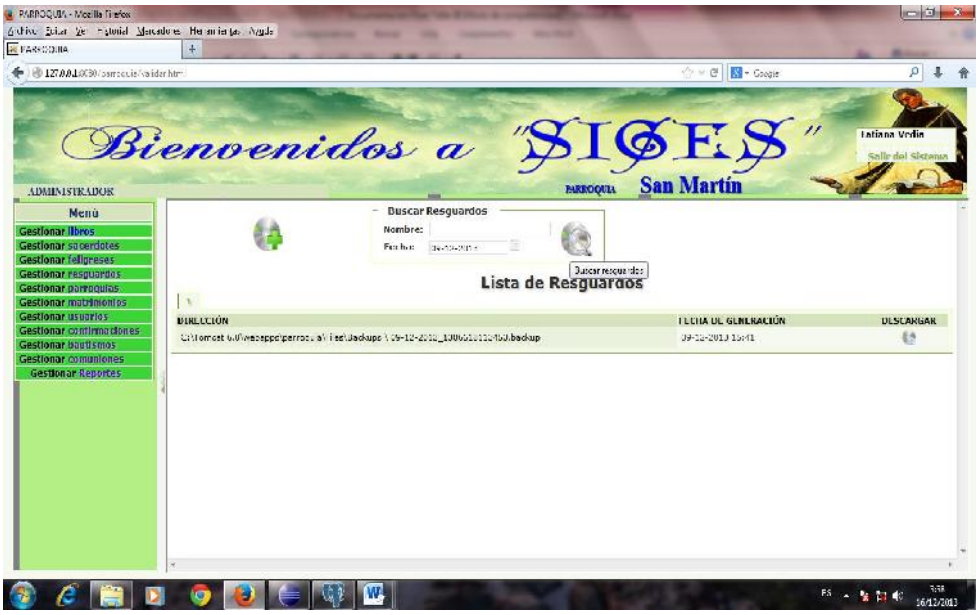


Figura Nº56. Pantalla Buscar Resguardos

En esta pantalla se muestra la opción buscar resguardos de la base de datos.

2.1.2.2.3.9.1.11 Pantalla Gestionar Parroquias

Figura Nº57. Pantalla Gestionar Parroquia

En esta pantalla se lista a las parroquias con sus respectivos barrios.

2.1.2.2.3.9.1.12 Pantalla Registrar Parroquia

Figura Nº58. Pantalla Registrar Parroquia

En esta pantalla se realiza el registro de una nueva parroquia.

2.1.2.2.3.9.1.13 Pantalla Modificar datos de la Parroquia

Figura Nº59. Pantalla Modificar datos de la Parroquia

2.1.2.2.3.9.1.14 Pantalla Buscar Parroquia

Figura Nº60. Pantalla Buscar Parroquia

En esta pantalla se muestra la opción buscar Parroquia.

2.1.2.2.3.9.1.15 Pantalla Gestionar Sacerdotes

Figura Nº61. Pantalla Gestionar Sacerdotes

En esta pantalla se lista a los Sacerdotes.

2.1.2.2.3.9.1.16 Pantalla Registrar nuevo Sacerdote

Figura Nº62. Pantalla Registrar nuevo Sacerdote

En esta pantalla se realiza el registro de un nuevo Sacerdote.

2.1.2.2.3.9.1.17 Pantalla Modificar Sacerdotes

Figura Nº63. Pantalla Modificar Sacerdotes

En esta pantalla se modifican los datos de algún sacerdote.

2.1.2.2.3.9.1.18 Pantalla Gestionar Feligreses

Figura Nº64. Pantalla Gestionar Feligreses

En esta pantalla se muestra lista de feligreses.

2.1.2.2.3.9.1.19 Pantalla Registrar nuevo Feligrés

Figura Nº65. Pantalla Gestionar Feligrés

En esta pantalla se realiza el registro de un nuevo feligrés.

2.1.2.2.3.9.1.20 Pantalla Modificar Feligrés

Figura Nº66. Pantalla Modificar Feligrés

En esta pantalla se modifican los datos de algún feligrés.

2.1.2.2.3.9.1.21 Pantalla Reporte Feligrés

Figura Nº67. Pantalla Reporte Feligrés

En esta pantalla se visualizan los datos de los sacramentos recibidos por un feligrés.

2.1.2.2.3.9.1.22 Pantalla Gestionar Libros

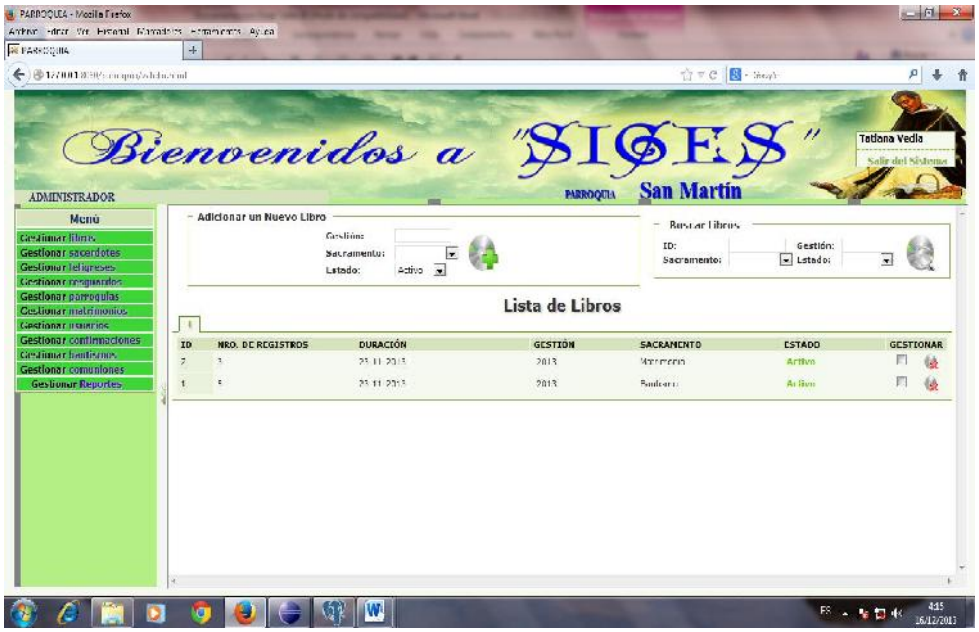


Figura Nº68. Pantalla Gestionar Libros

En esta pantalla se lista a los libros existentes de cada sacramento.

2.1.2.2.3.9.1.23 Pantalla Registrar Libro

Figura Nº69. Pantalla Registrar Libro

En esta pantalla se realiza el registro de un nuevo libro.

2.1.2.2.3.9.1.24 Pantalla Gestionar Bautismos

Figura Nº70. Pantalla Gestionar Bautismos

En esta pantalla se lista las actas del Sacramento Bautismo

2.1.2.2.3.9.1.25 Pantalla Registrar Nuevo Bautismo

Figura Nº71. Pantalla Registrar nuevo Bautismo

En esta pantalla se realiza el registro de una nueva inscripción a sacramento Bautismo

2.1.2.2.3.9.1.26 Pantalla Buscar Registros Bautismo

Figura Nº72. Pantalla Buscar Registros Bautismo

En esta pantalla se muestra la opción buscar Registros de Bautismos

2.1.2.2.3.9.1.27 Pantalla Bautismos Celebrados

Figura Nº73. Pantalla Bautismos Celebrados

En esta pantalla se muestra los inscritos para el bautismo, se selecciona los celebrados

2.1.2.2.3.9.1.28 Pantalla Certificado de Bautismo

Figura Nº74. Pantalla Certificado de Bautismo

En esta pantalla se muestra el certificado de Bautismo emitido por la parroquia

2.1.2.2.3.9.1.29 Pantalla Gestionar Comunión

Figura Nº75. Pantalla Gestionar Comunión

En esta pantalla se lista a los registros de Sacramento comunión.

2.1.2.2.3.9.1.30 Pantalla Registrar nueva Comunión

Figura Nº76. Pantalla Registrar inscripción Comunión

En esta pantalla se realiza la inscripción de un feligrés a Sacramento Comunión.

2.1.2.2.3.9.1.31 Pantalla Registros Confirmación

Figura Nº77. Pantalla Registros Confirmación

En esta pantalla se lista a los registros de Sacramento confirmación.

2.1.2.2.3.9.1.32 Pantalla Registrar nueva Confirmación

Figura Nº78. Pantalla Registrar nueva Confirmación

En esta pantalla se realiza la inscripción de un feligrés a Sacramento Confirmación.

2.1.2.2.3.9.1.33 Pantalla Gestionar Matrimonios

Figura Nº79. Pantalla Gestionar Matrimonios

En esta pantalla se lista las actas del Sacramento Matrimonio.

2.1.2.2.3.9.1.34 Pantalla Registrar nuevo Matrimonio

Figura Nº80. Pantalla Registrar nuevo Matrimonio

En esta pantalla se realiza el registro de una nueva inscripción a Matrimonio.

2.1.2.2.3.9.1.35 Pantalla Buscar Matrimonio

Figura Nº81. Pantalla Buscar Matrimonio

En esta pantalla se muestra la opción buscar Registros de Matrimonio.

2.1.2.2.3.9.1.36 Pantalla Matrimonios Celebrados

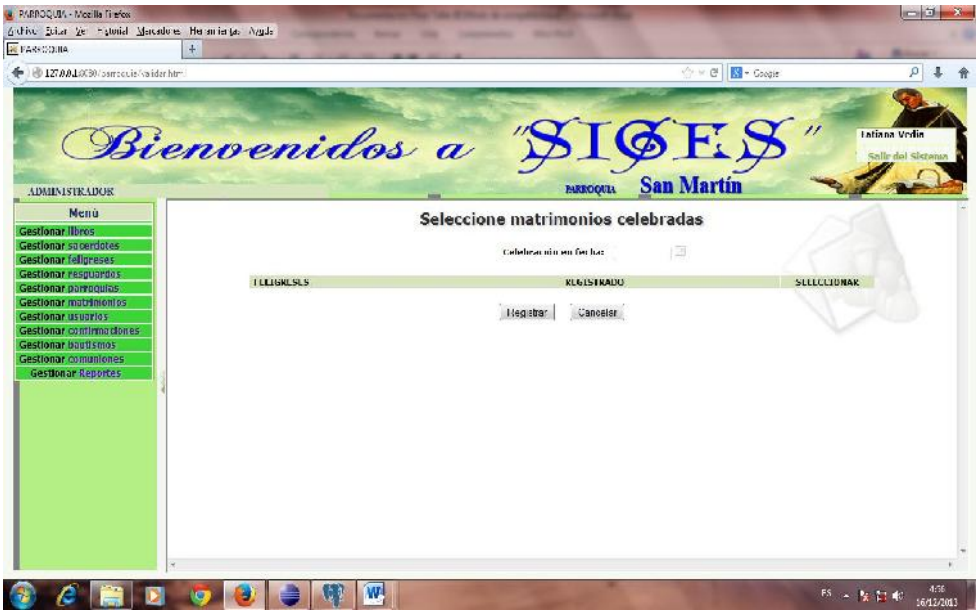


Figura Nº82. Pantalla Matrimonios Celebrados

En esta pantalla se muestra los inscritos para matrimonio, se selecciona los celebrados

2.1.2.2.3.9.1.37 Pantalla Gestionar Reportes

Figura Nº83. Pantalla Gestionar Reportes

En esta pantalla se visualiza el tipo de reportes a seleccionar por sacramento

2.1.2.2.3.9.1.38 Pantalla Reportes Bautismos

Figura Nº84. Pantalla Reportes Bautismos

2.1.2.2.3.9.1.39 Pantalla Reportes Comunión

Figura Nº85. Pantalla Reportes Comunión

2.1.2.2.3.9.1.40 Pantalla Reportes Confirmaciones

Figura Nº86. Pantalla Reportes Confirmaciones

2.1.2.2.3.9.1.41 Pantalla Reportes Matrimonios

Figura Nº87. Pantalla Reportes Confirmaciones

2.1.2.2.3.10 Modelo de Análisis y Diseño

Este modelo establece la realización de los casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño (incluyendo una orientación hacia el entorno de implementación), de acuerdo al avance del proyecto.

2.1.2.2.3.10.1 Modelado de Diagrama de Actividades

2.1.2.2.3.10.1.1 Introducción

El diagrama de Actividades es un artefacto de la disciplina Análisis de Sistemas en la metodología RUP la cual estamos implementando.

Los Diagramas de Actividades se Utilizan para modelar aspectos dinámicos de un Sistema, esto implica modelar los pasos secuenciales de un proceso.

2.1.2.2.3.10.1.2 Propósito

Comprender la estructura y la dinámica del sistema deseado.

Identificar posibles mejoras en el Sistema.

2.1.2.2.3.10.1.3 Alcance

Describir los procesos del sistema y los clientes.

Identificar y definir los procesos de los casos de uso según los objetivos de la organización.

Definir un diagrama de actividad para cada caso de uso del sistema.

2.1.2.2.3.10.1.4 Diagrama de Actividades

2.1.2.2.3.10.1.4.1 Diagrama de Actividad: Caso de Uso Acceder al Sistema

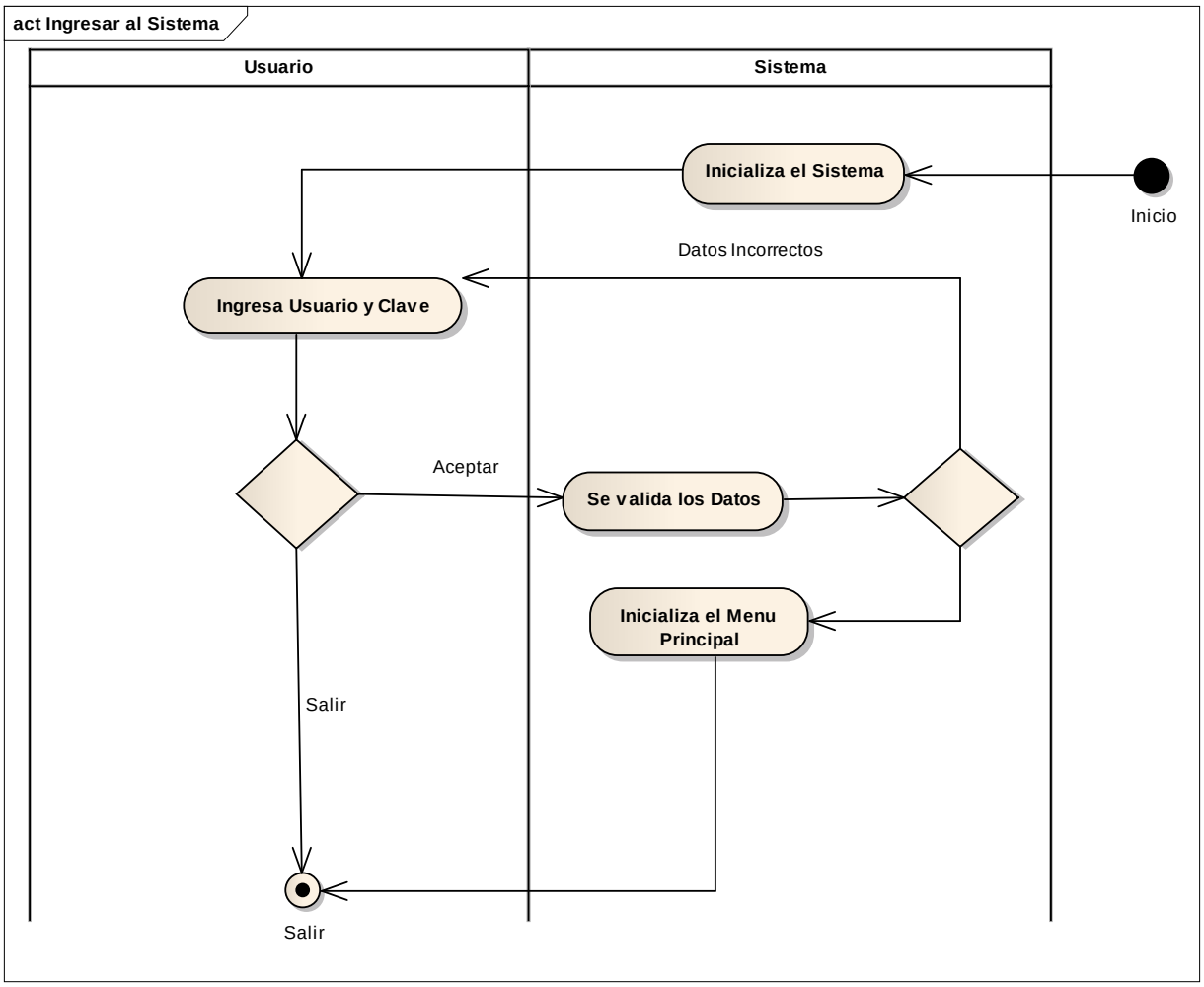


Figura Nº88. Diagrama de actividad: CU Acceder al Sistema

2.1.2.2.3.10.1.4.2 Diagrama de Actividad: Caso de Uso Gestionar Usuarios

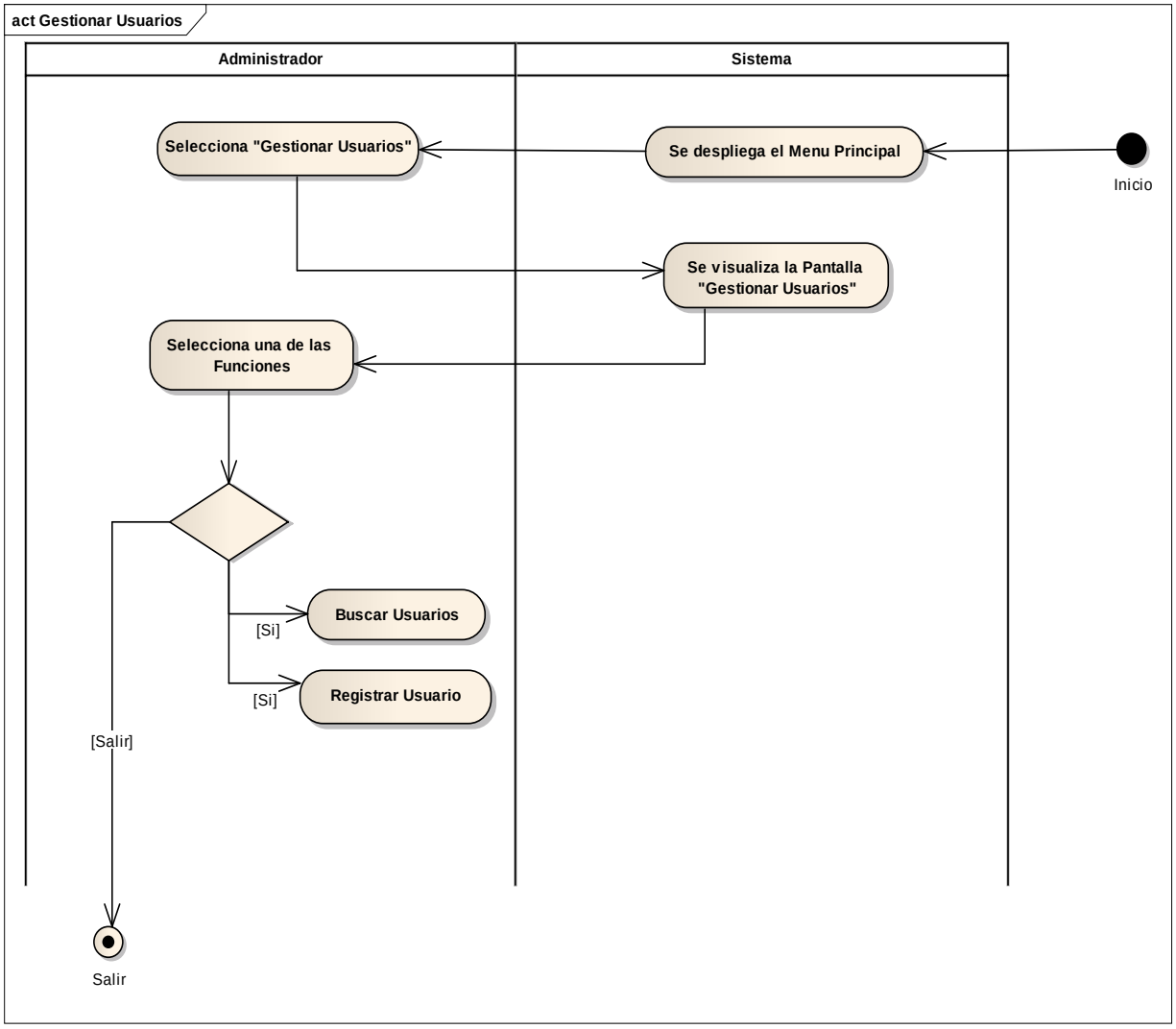


Figura Nº89. Diagrama de actividad: CU Gestionar Usuario

2.1.2.2.3.10.1.4.3 Diagrama de Actividad: Caso de Uso Registrar Usuario

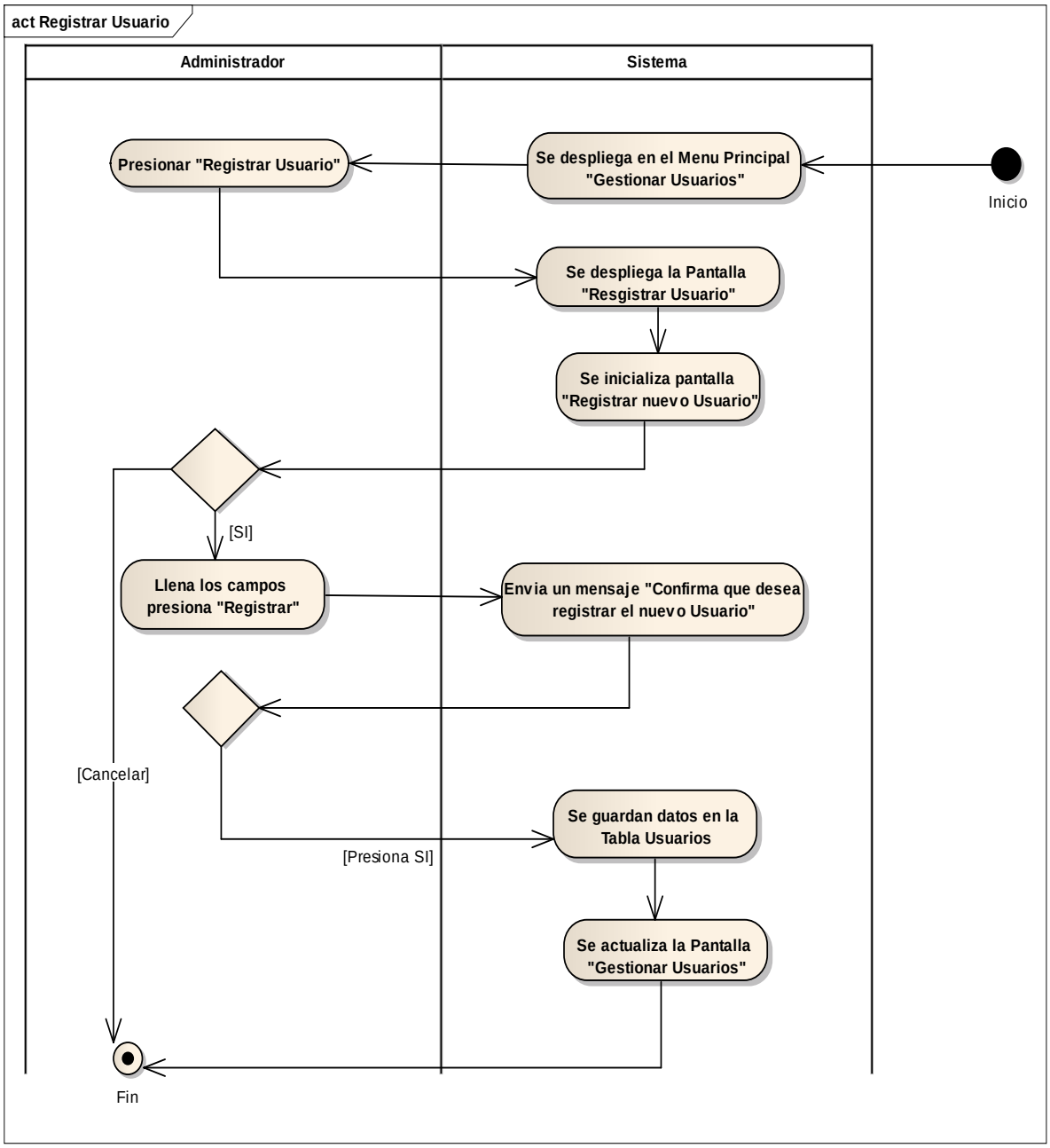


Figura Nº90. Diagrama de actividad: CU Registrar Usuario

2.1.2.2.3.10.1.4.4 Diagrama de Actividad: Caso de Uso Modificar Usuario

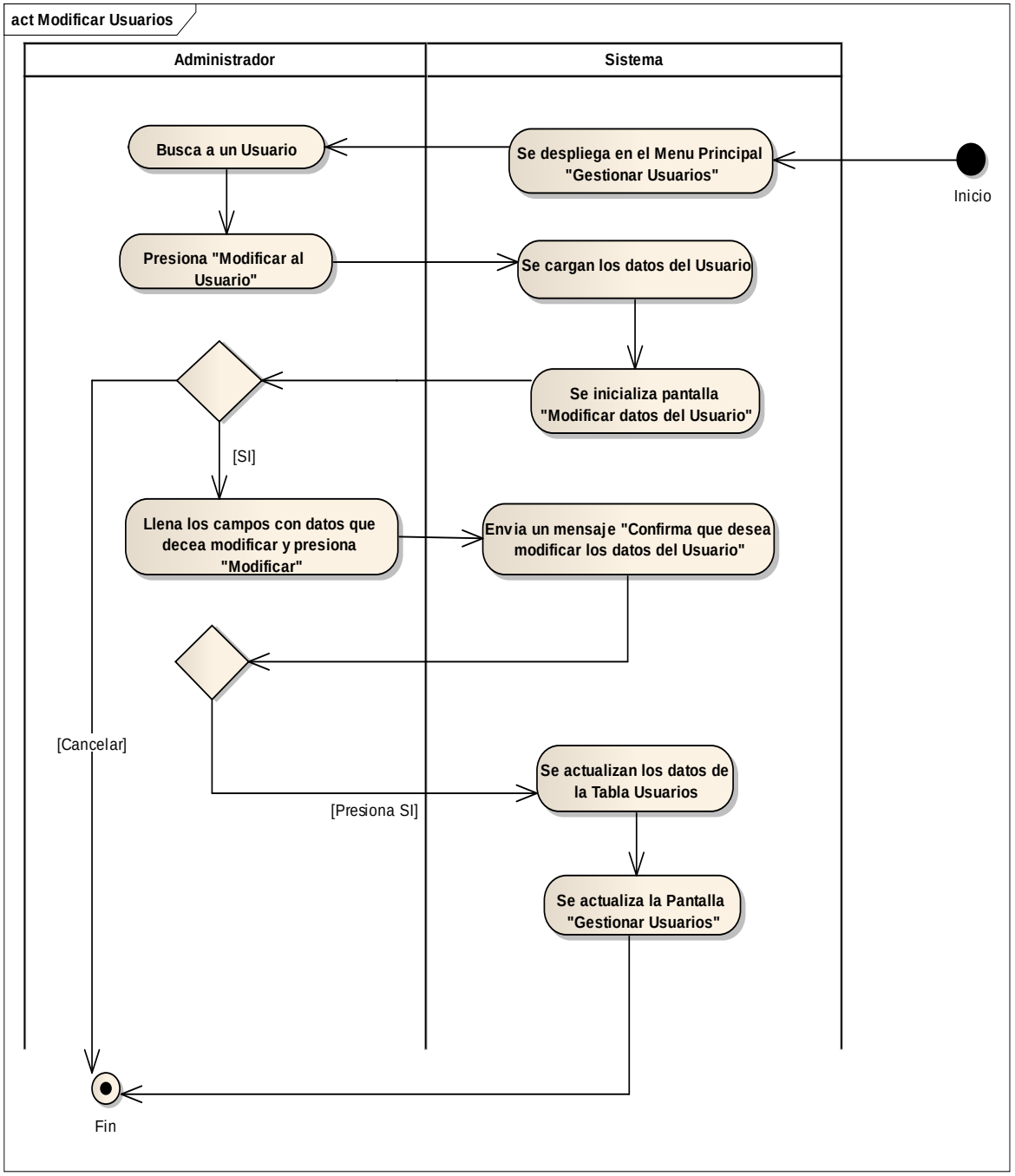


Figura Nº91. Diagrama de actividad: CU Modificar Usuario

2.1.2.2.3.10.1.4.5 Diagrama de Actividad: Caso de Uso Dar de baja Usuarios

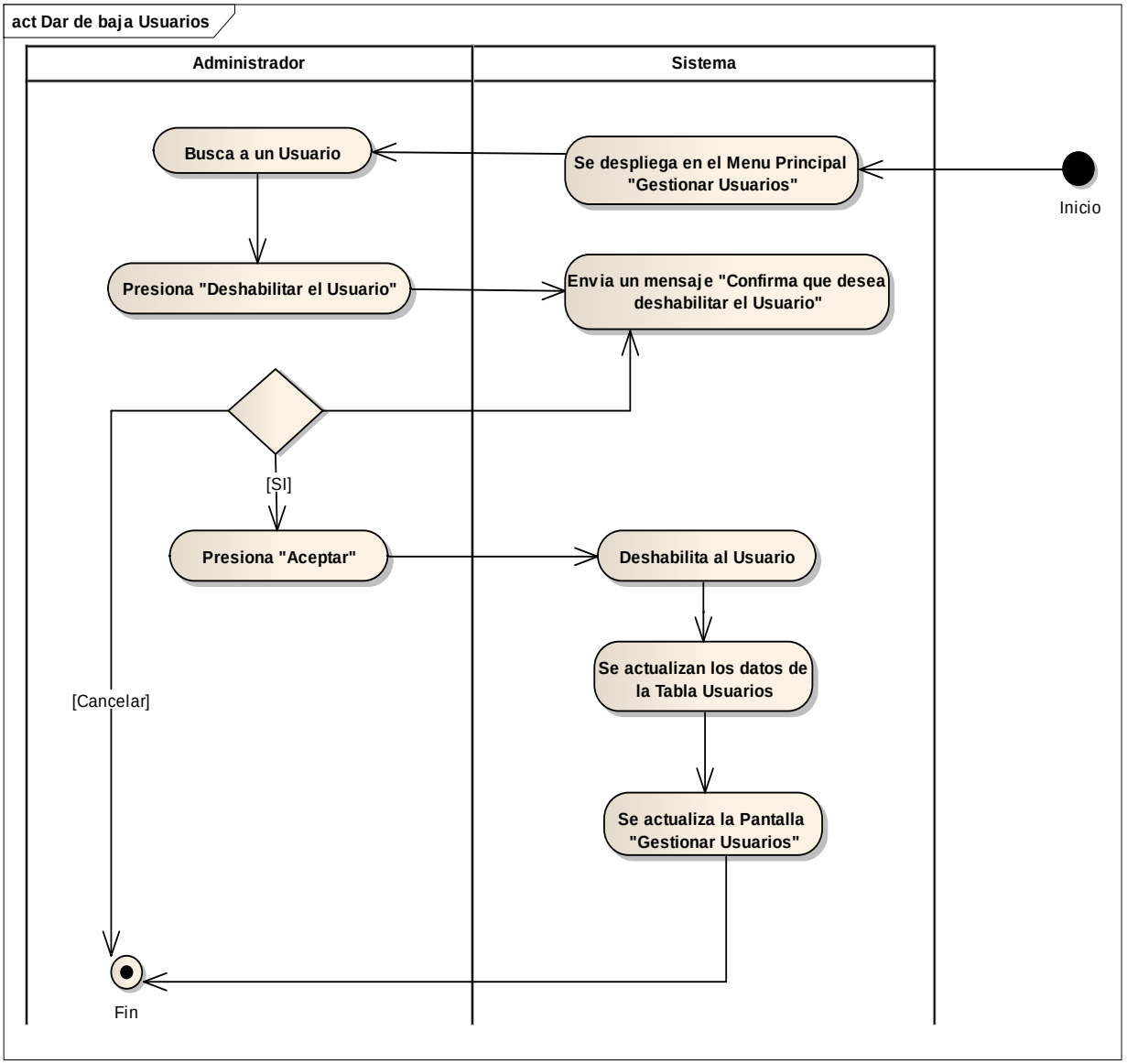


Figura Nº92. Diagrama de actividad: CU Dar de baja Usuarios

2.1.2.2.3.10.1.4.6 Diagrama de Actividad: Caso de Uso Imprimir datos Usuarios

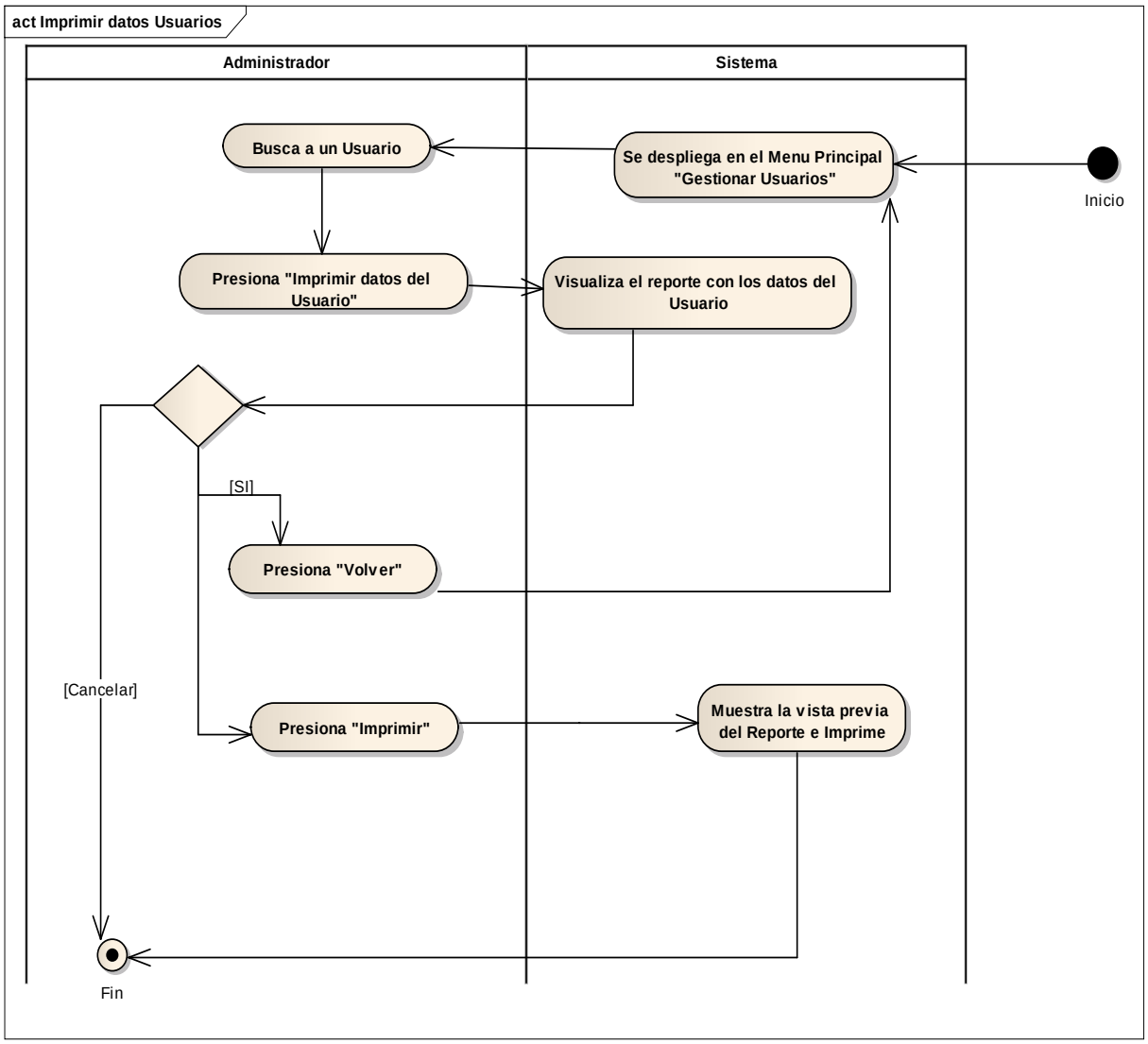


Figura Nº93. Diagrama de actividad: CU Imprimir datos Usuarios

2.1.2.2.3.10.1.4.7 Diagrama de Actividad: Caso de Uso Gestionar Feligreses

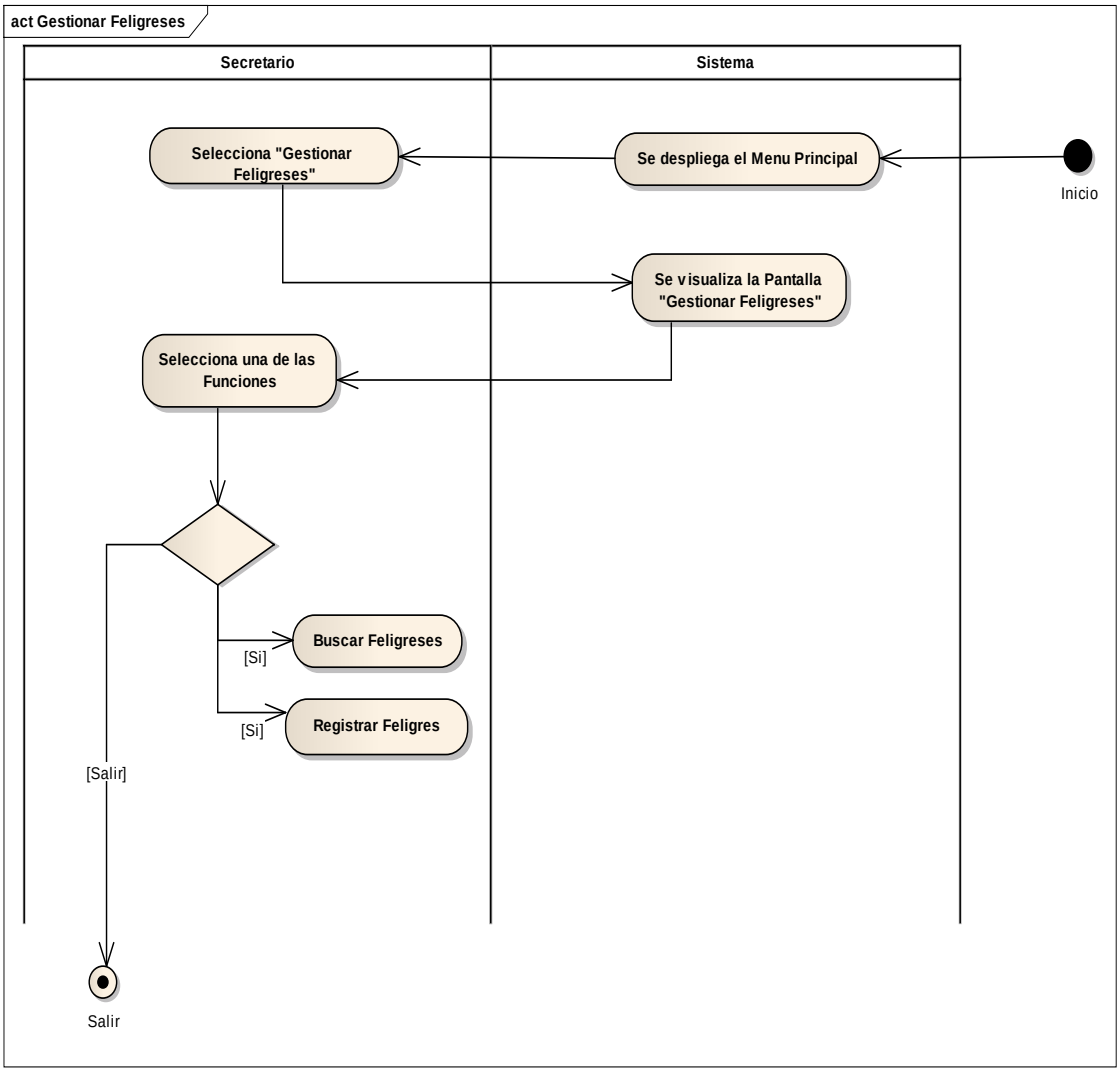


Figura Nº94. Diagrama de actividad: CU Gestionar Feligreses

2.1.2.2.3.10.1.4.8 Diagrama de Actividad: Caso de Uso Registrar Feligreses

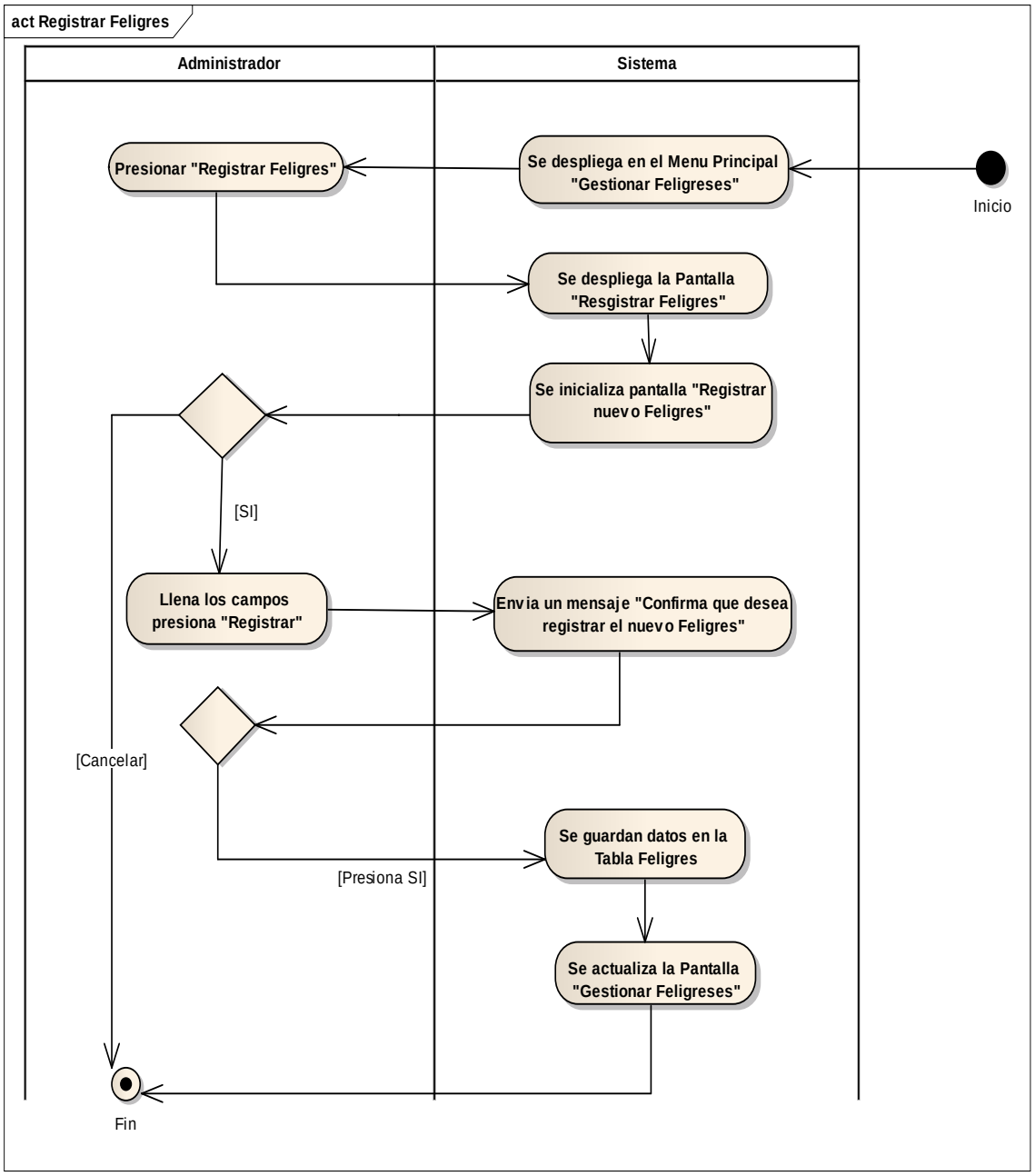


Figura Nº95. Diagrama de actividad: CU Registrar Feligreses

2.1.2.2.3.10.1.4.9 Diagrama de Actividad: Caso de Uso Modificar Feligreses

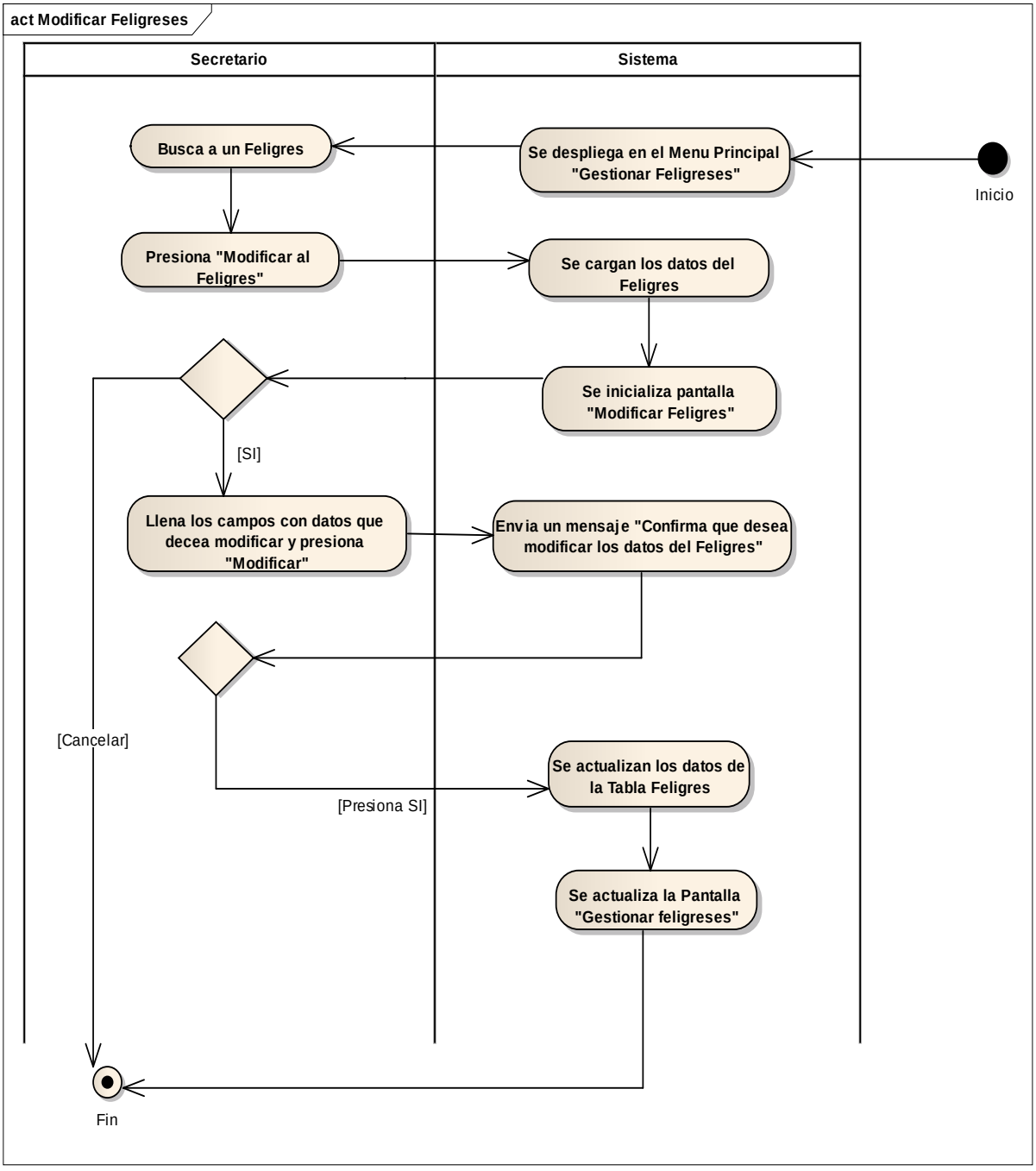


Figura Nº96. Diagrama de actividad: CU Modificar Feligreses

2.1.2.2.3.10.1.4.10 Diagrama de Actividad: Caso de Uso Imprimir datos Feligrés

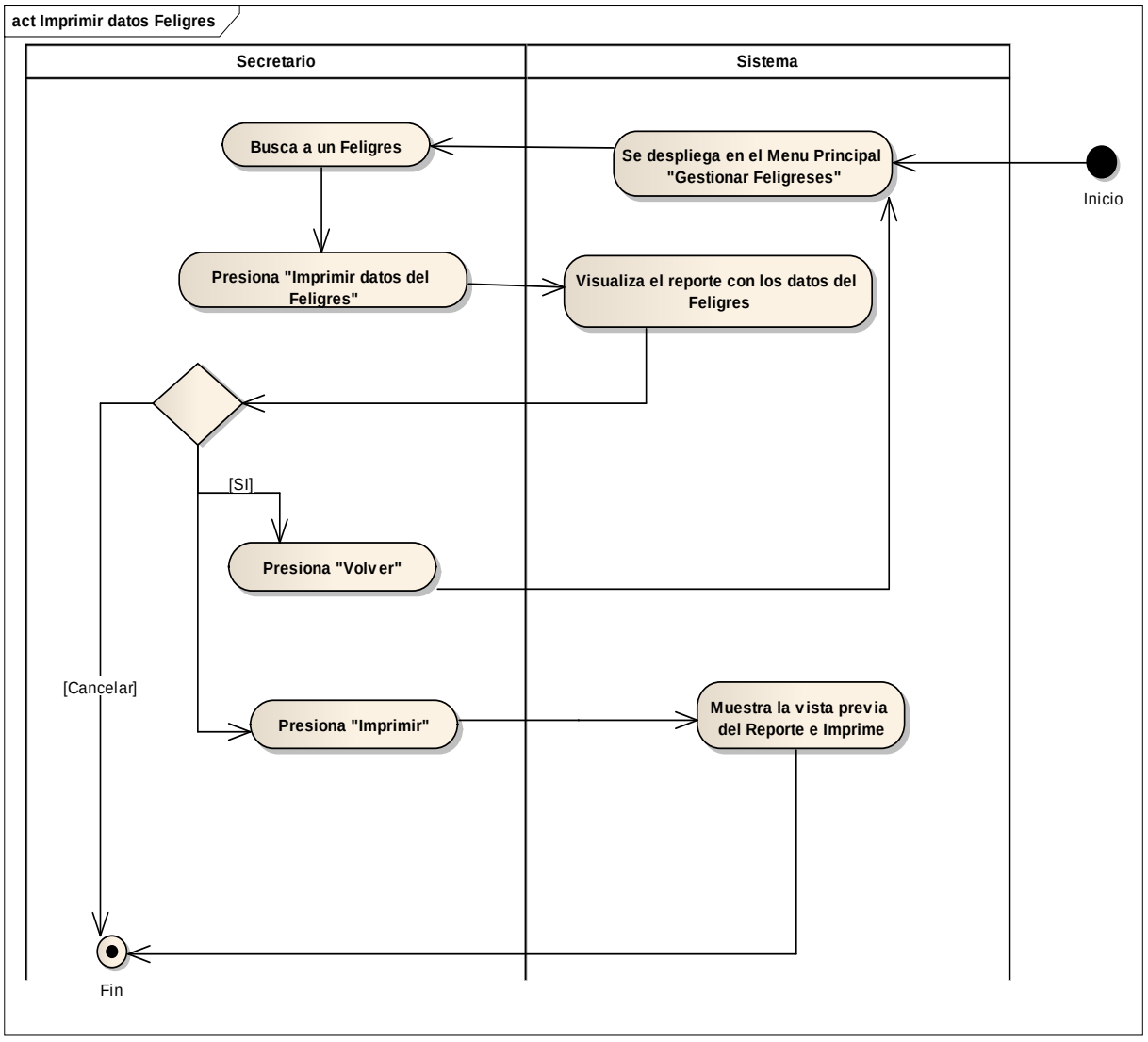


Figura Nº97. Diagrama de actividad: CU Imprimir datos Feligrés

2.1.2.2.3.10.1.4.11 Diagrama de Actividad: Caso de Uso Gestionar Libros

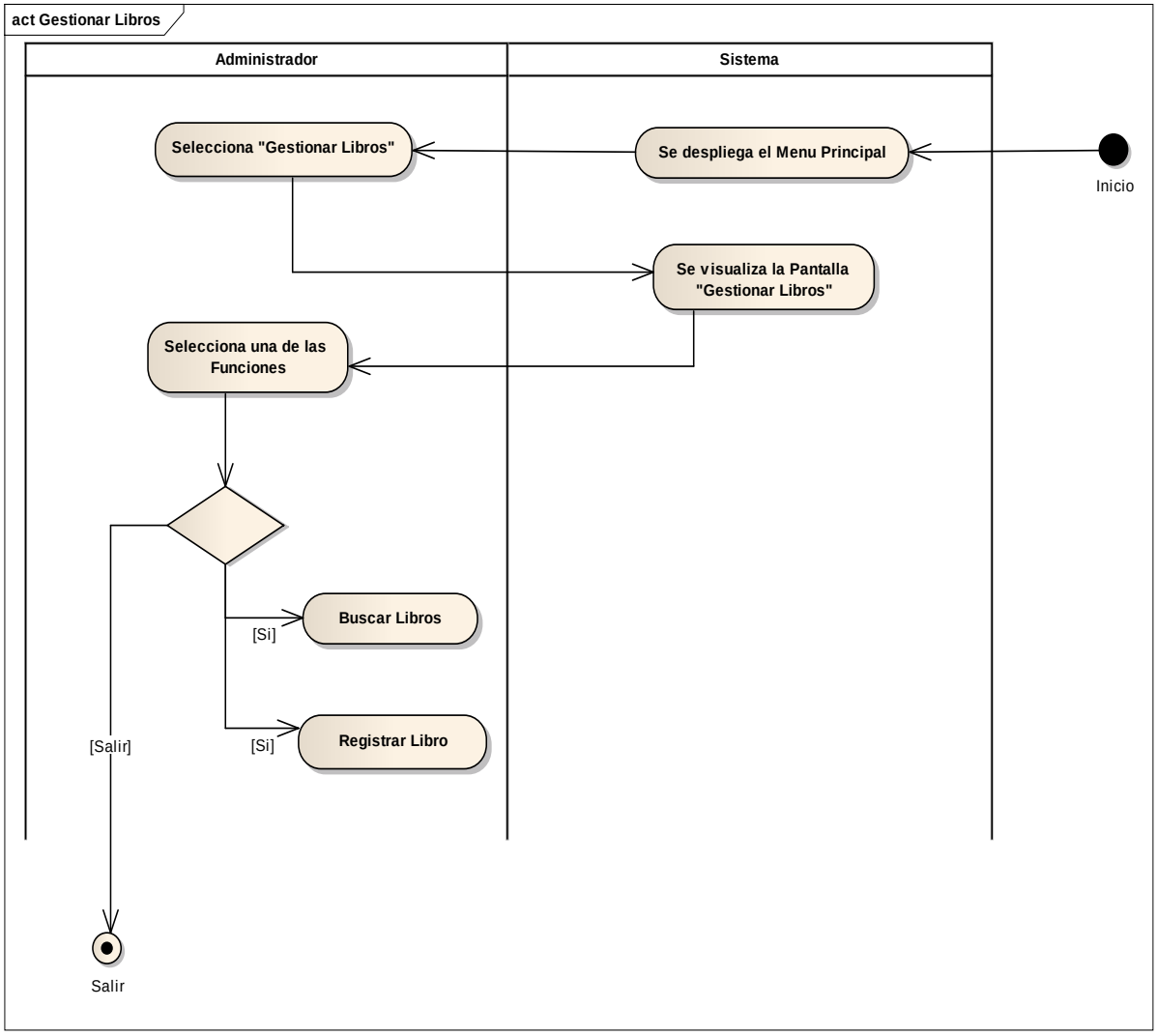


Figura Nº98. Diagrama de actividad: CU Gestionar Libros

2.1.2.2.3.10.1.4.12 Diagrama de Actividad: Caso de Uso Registrar Libro

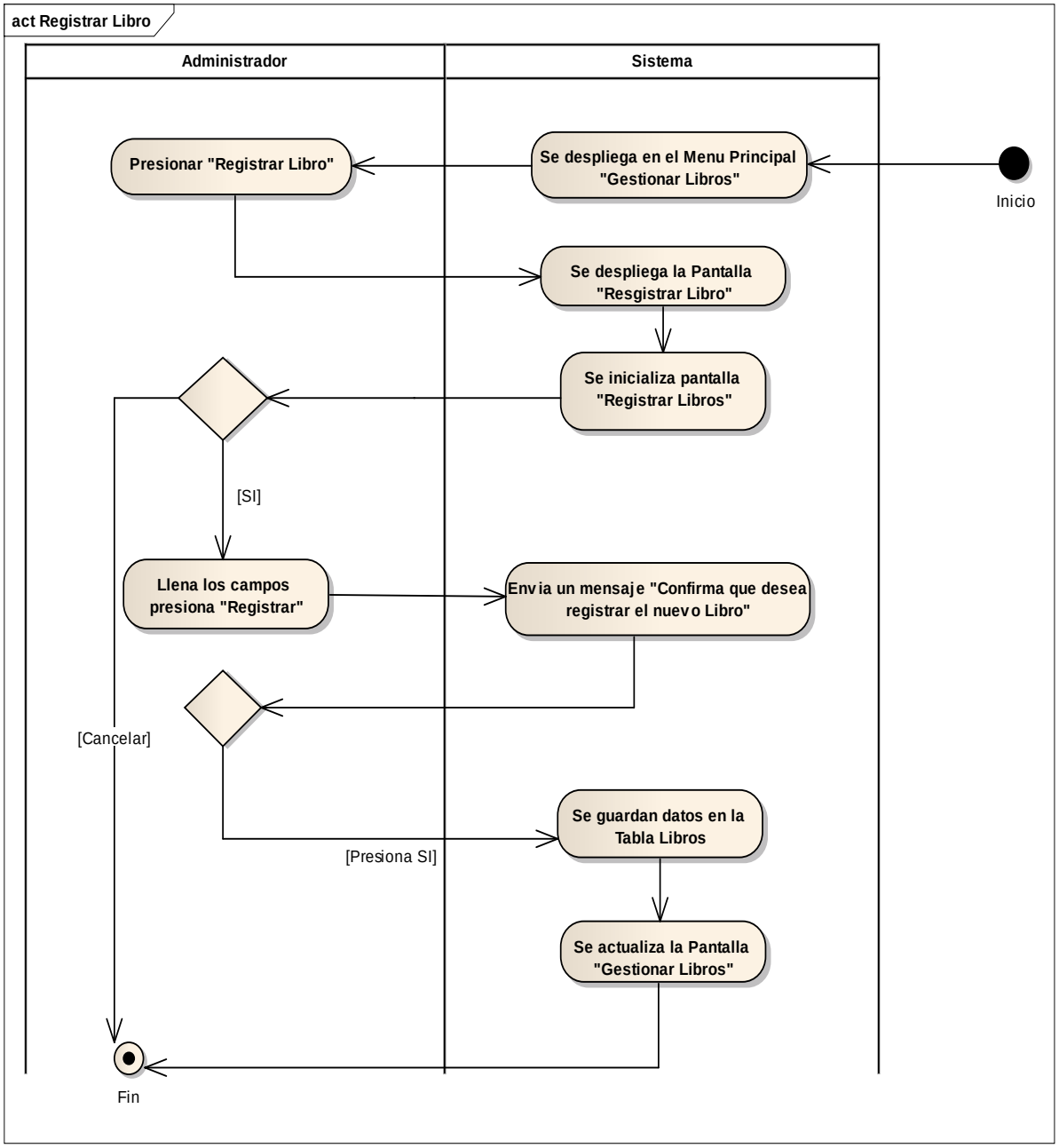


Figura Nº99. Diagrama de actividad: CU Registrar Libro

2.1.2.2.3.10.1.4.13 Diagrama de Actividad: Caso de Uso Modificar Libro

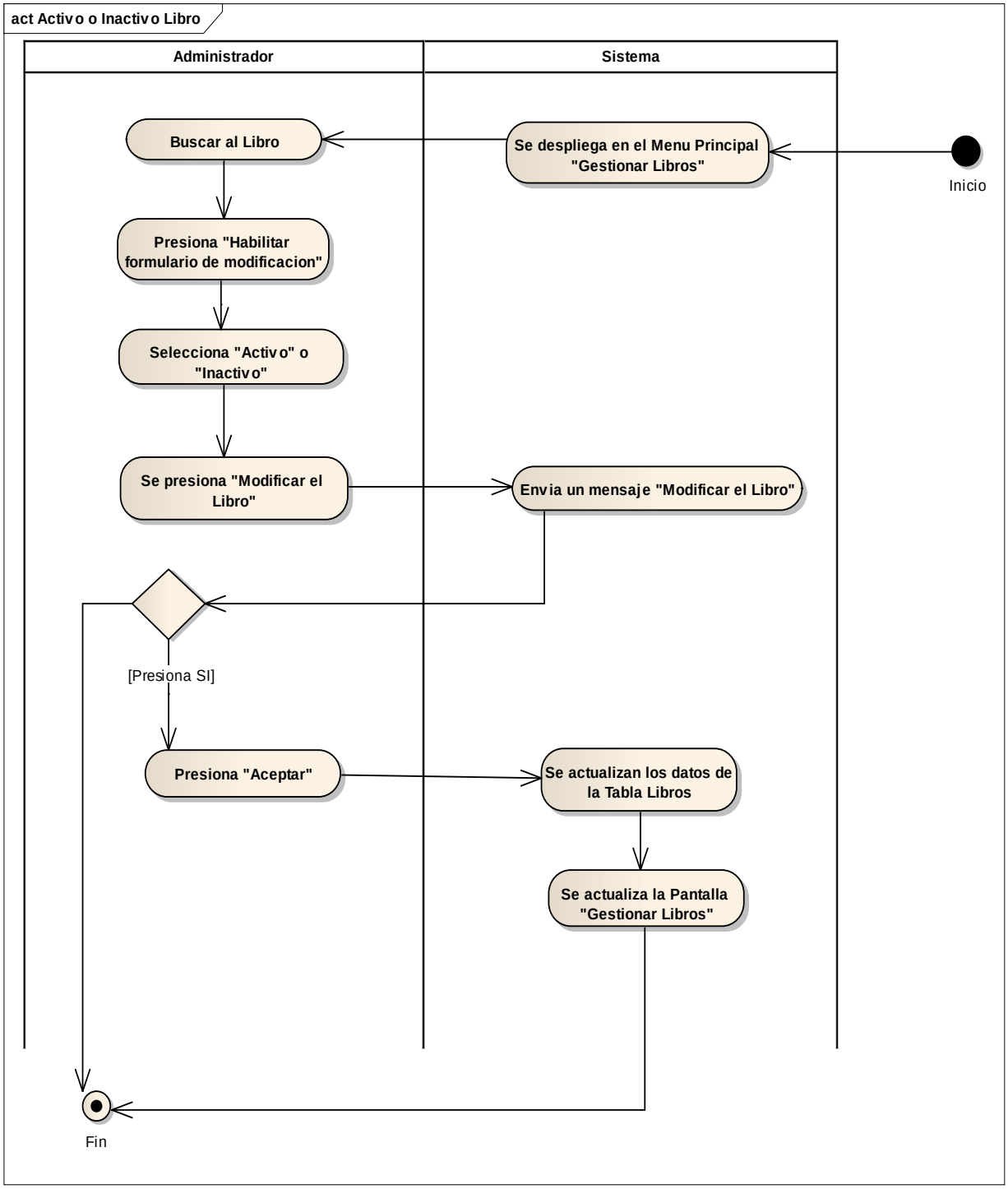


Figura Nº100. Diagrama de actividad: CU Modificar Libro

2.1.2.2.3.10.1.4.14 Diagrama de Actividad: Caso de Uso Gestionar Parroquia

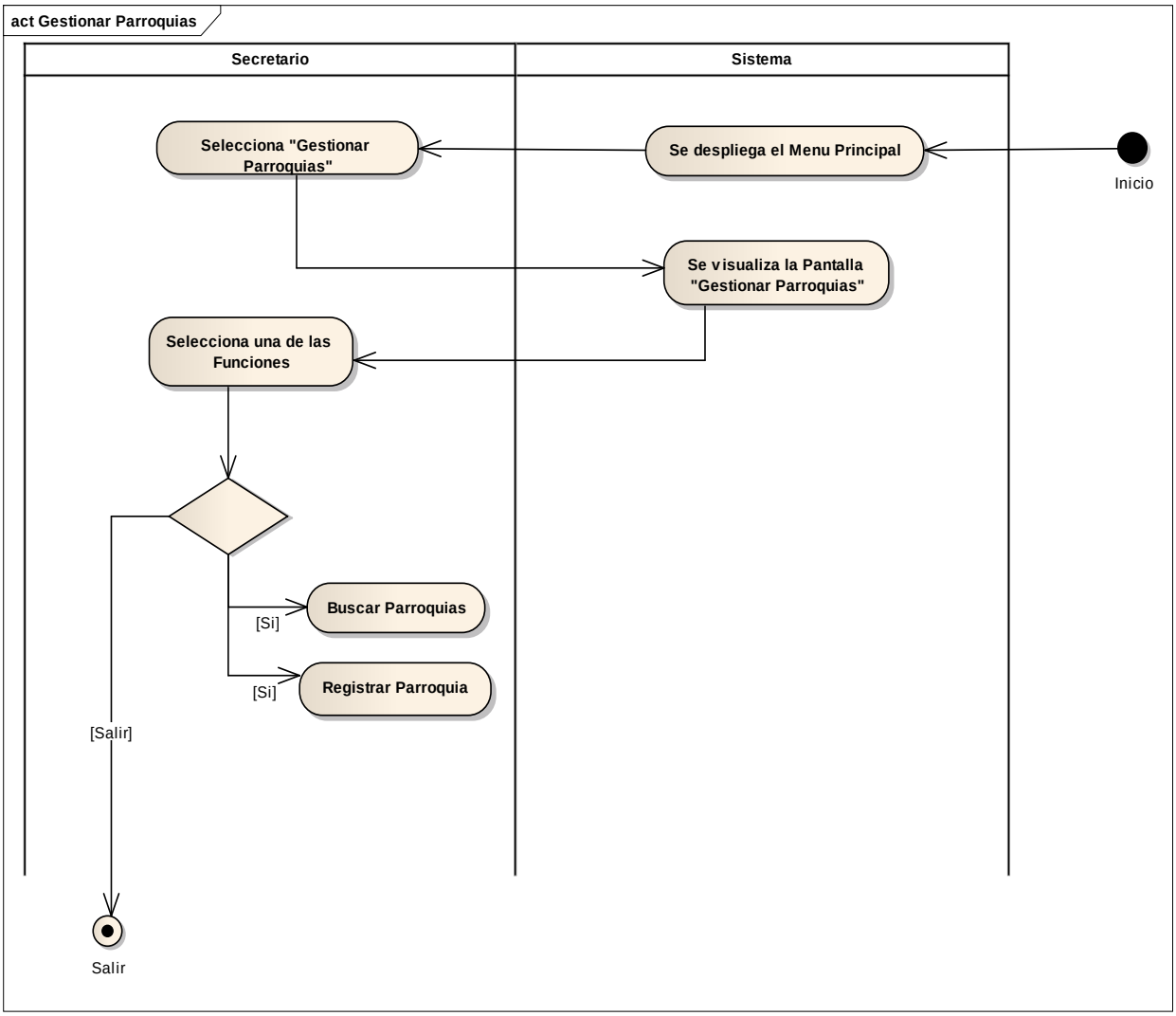


Figura Nº101. Diagrama de actividad: CU Gestionar Parroquia

2.1.2.2.3.10.1.4.15 Diagrama de Actividad: Caso de Uso Registrar Parroquia

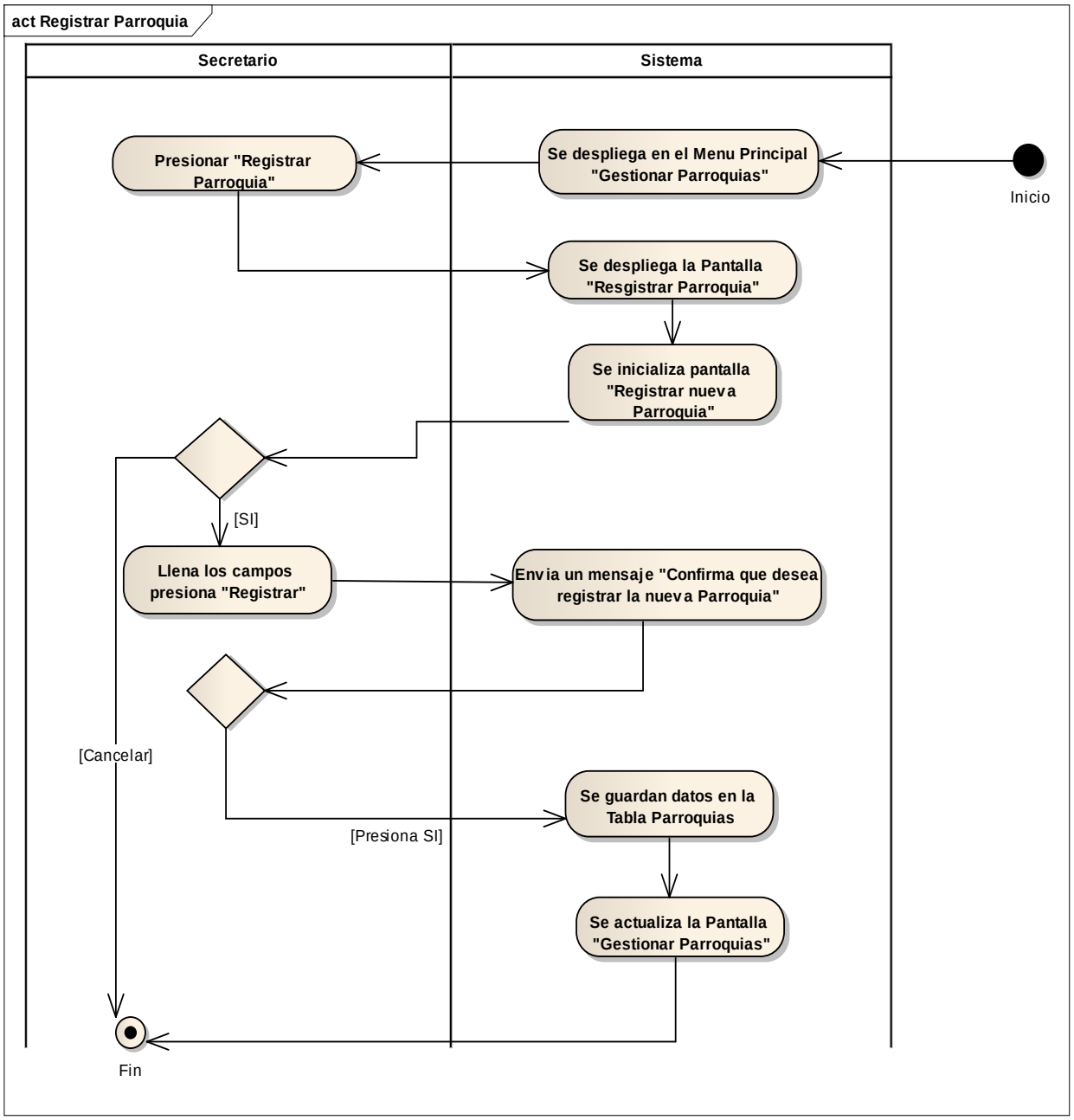


Figura Nº102. Diagrama de actividad: CU Registrar Parroquia

2.1.2.2.3.10.1.4.16 Diagrama de Actividad: Caso de Uso Modificar Parroquia

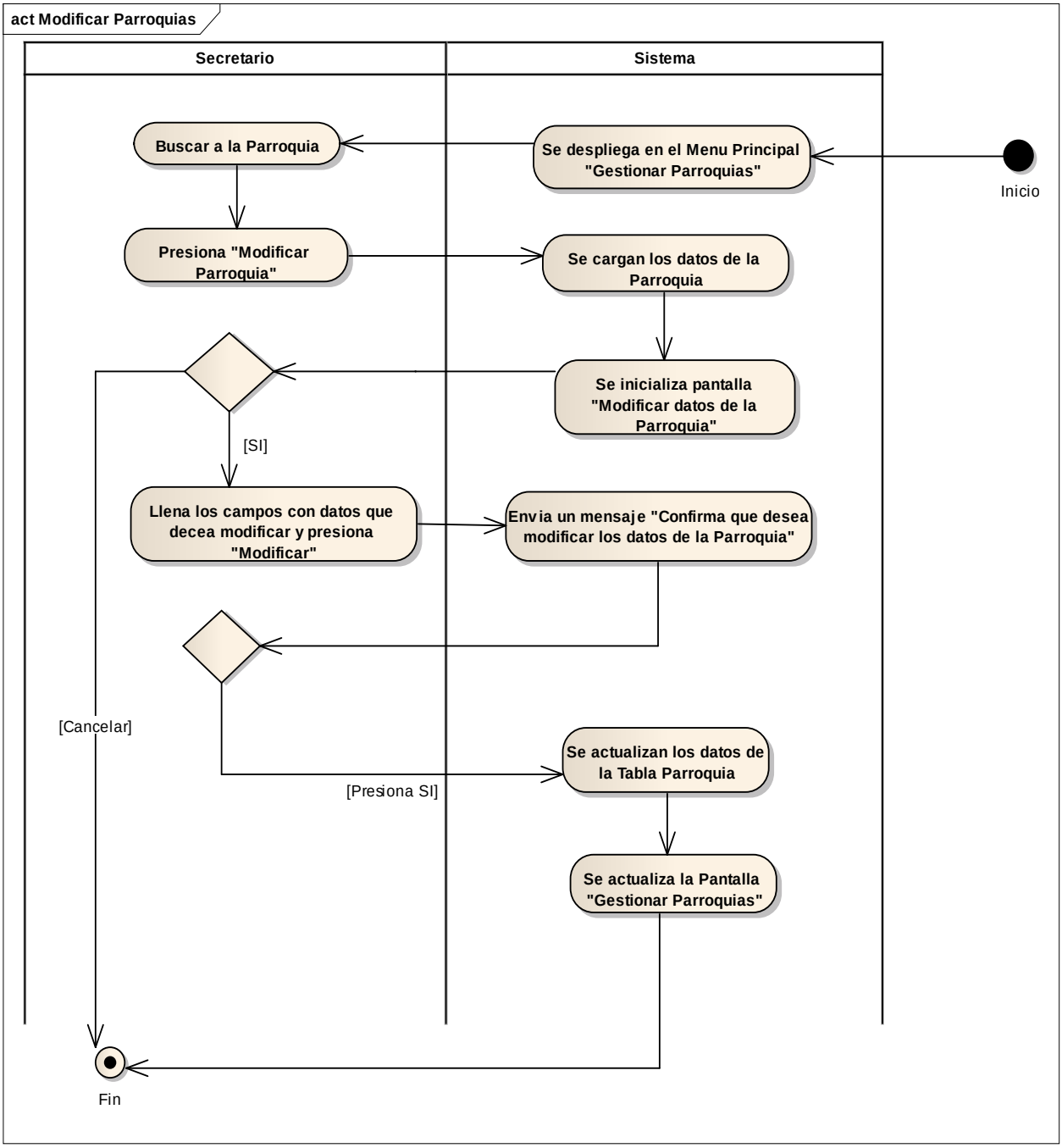


Figura Nº103. Diagrama de actividad: CU Modificar Parroquia

2.1.2.2.3.10.1.4.17 Diagrama de Actividad: Caso de Uso Gestionar Sacerdotes

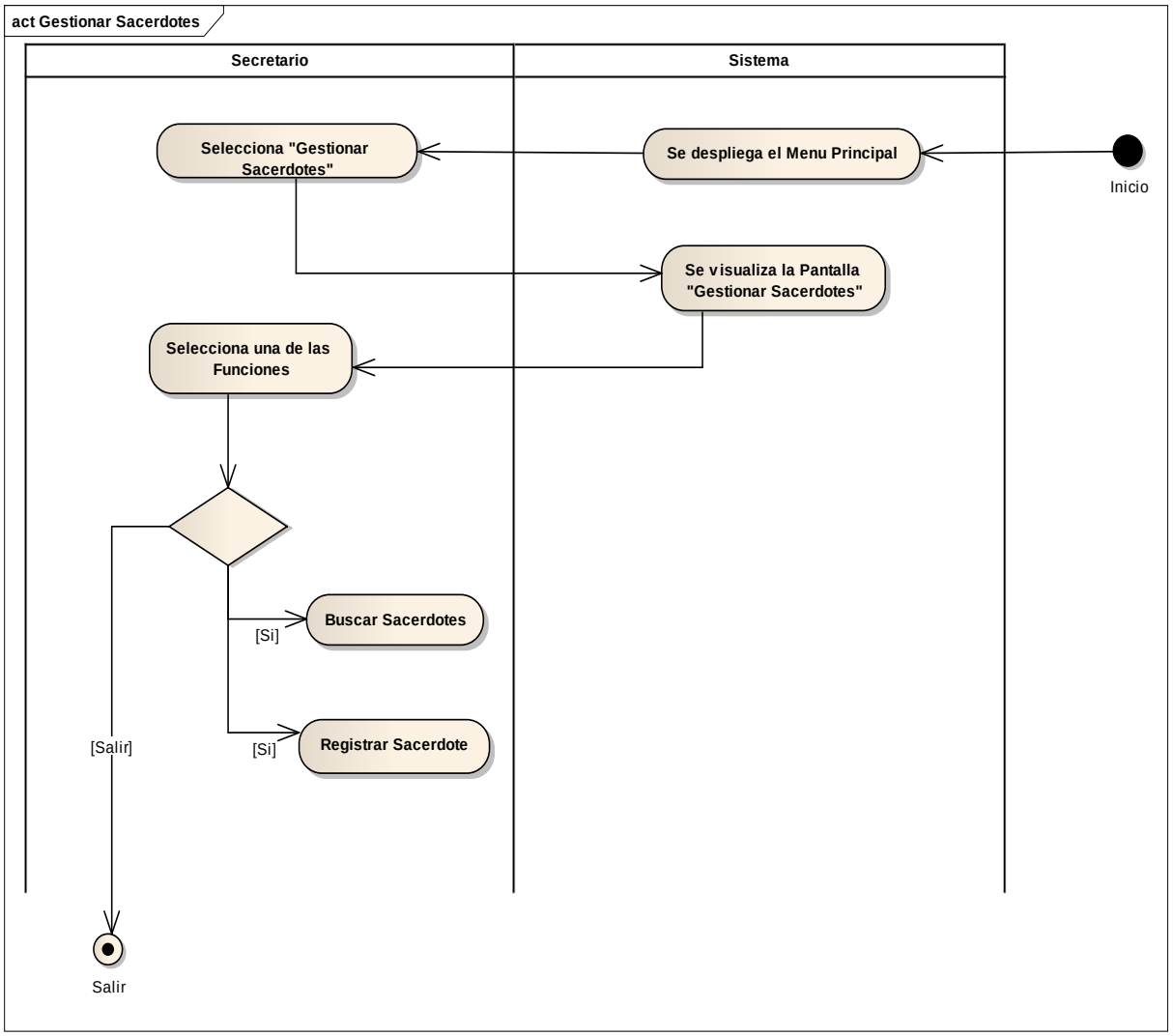


Figura Nº104. Diagrama de actividad: CU Gestionar Sacerdotes

2.1.2.2.3.10.1.4.18 Diagrama de Actividad: Caso de Uso Registrar Sacerdote

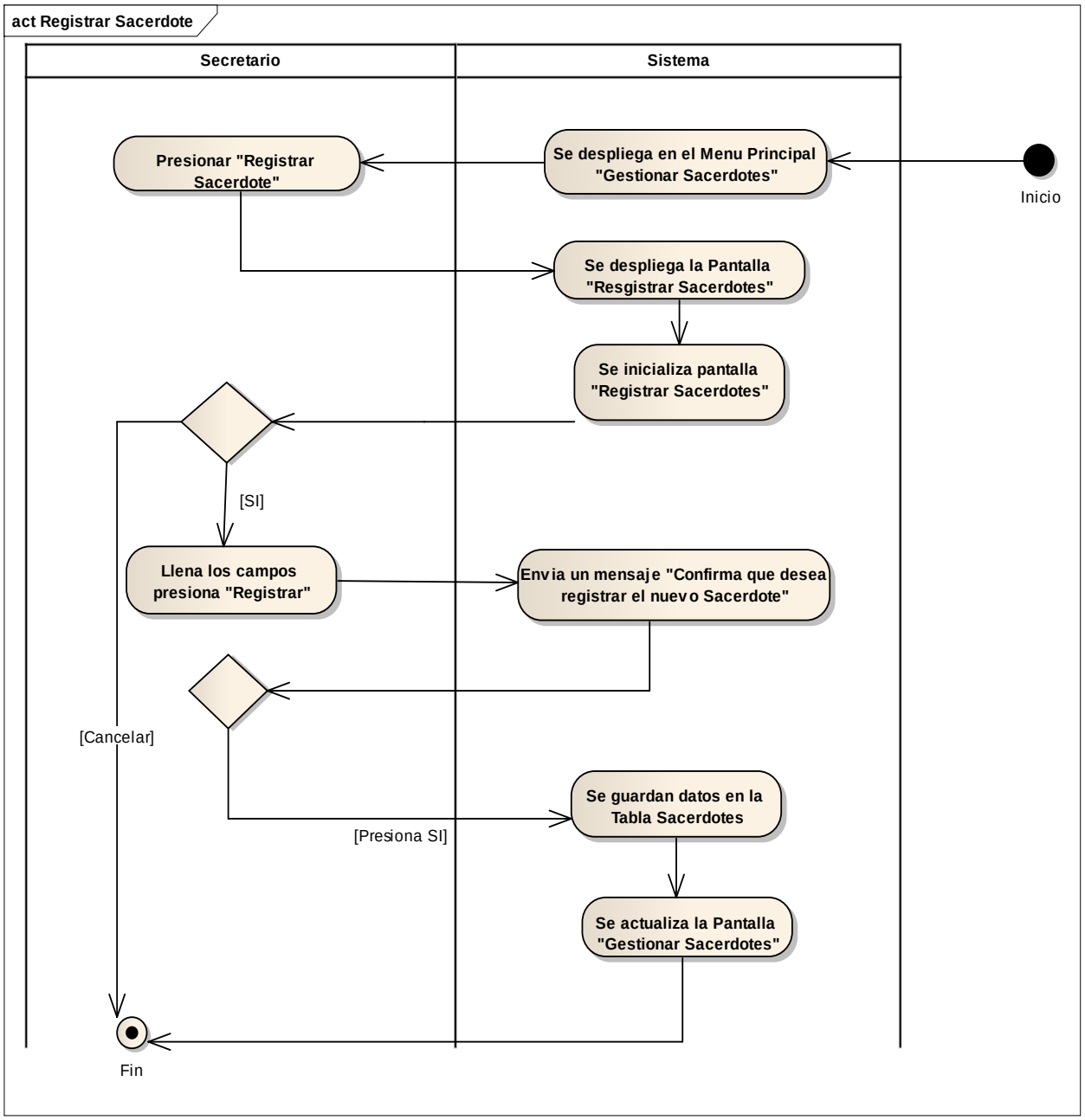


Figura N°105. Diagrama de actividad: CU Registrar Sacerdote

2.1.2.2.3.10.1.4.19 Diagrama de Actividad: Caso de Uso Modificar Sacerdote

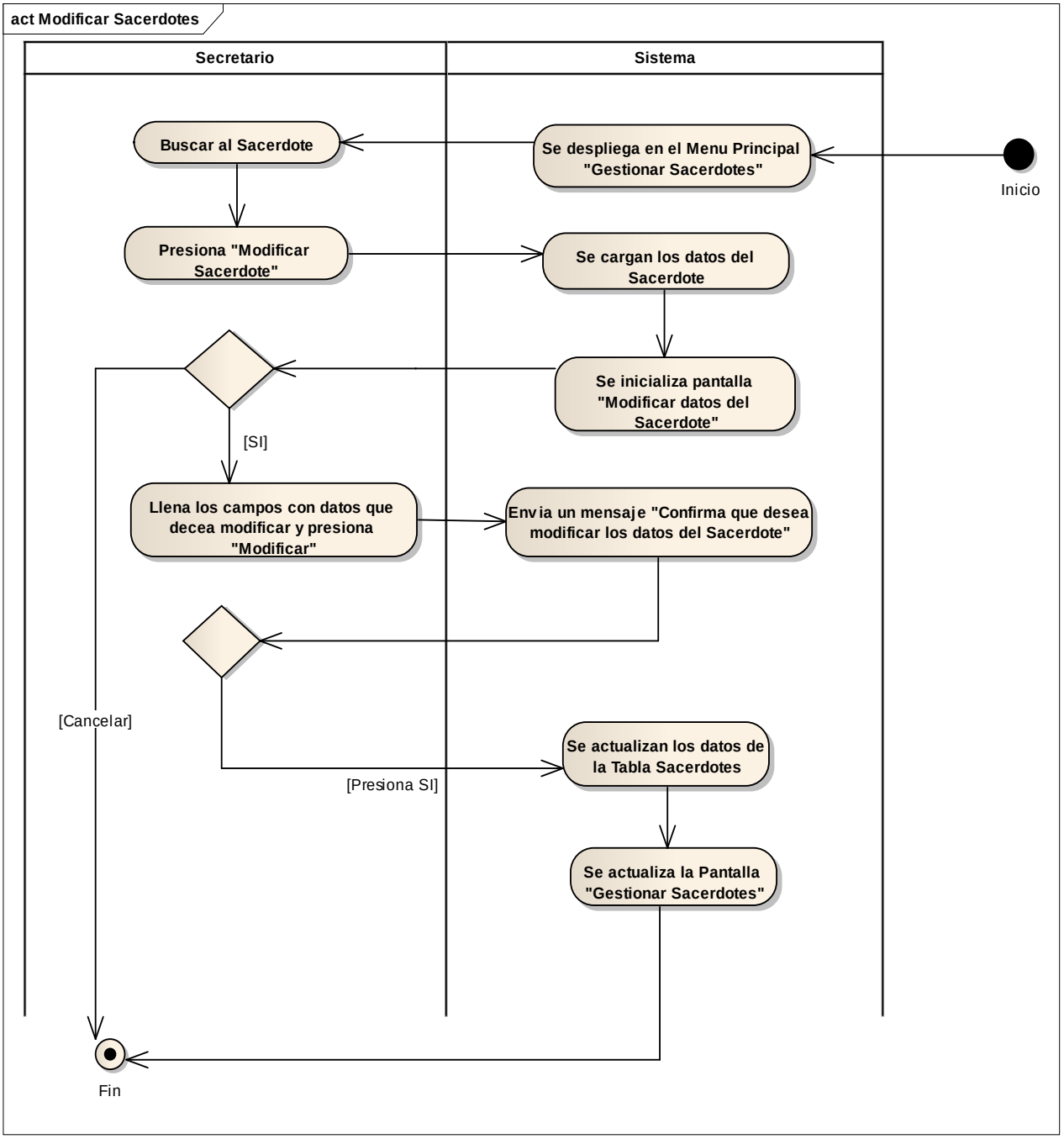


Figura Nº106. Diagrama de actividad: CU Modificar Sacerdote

2.1.2.2.3.10.1.4.20 Diagrama de Actividad: Caso de Uso Gestionar Bautismo

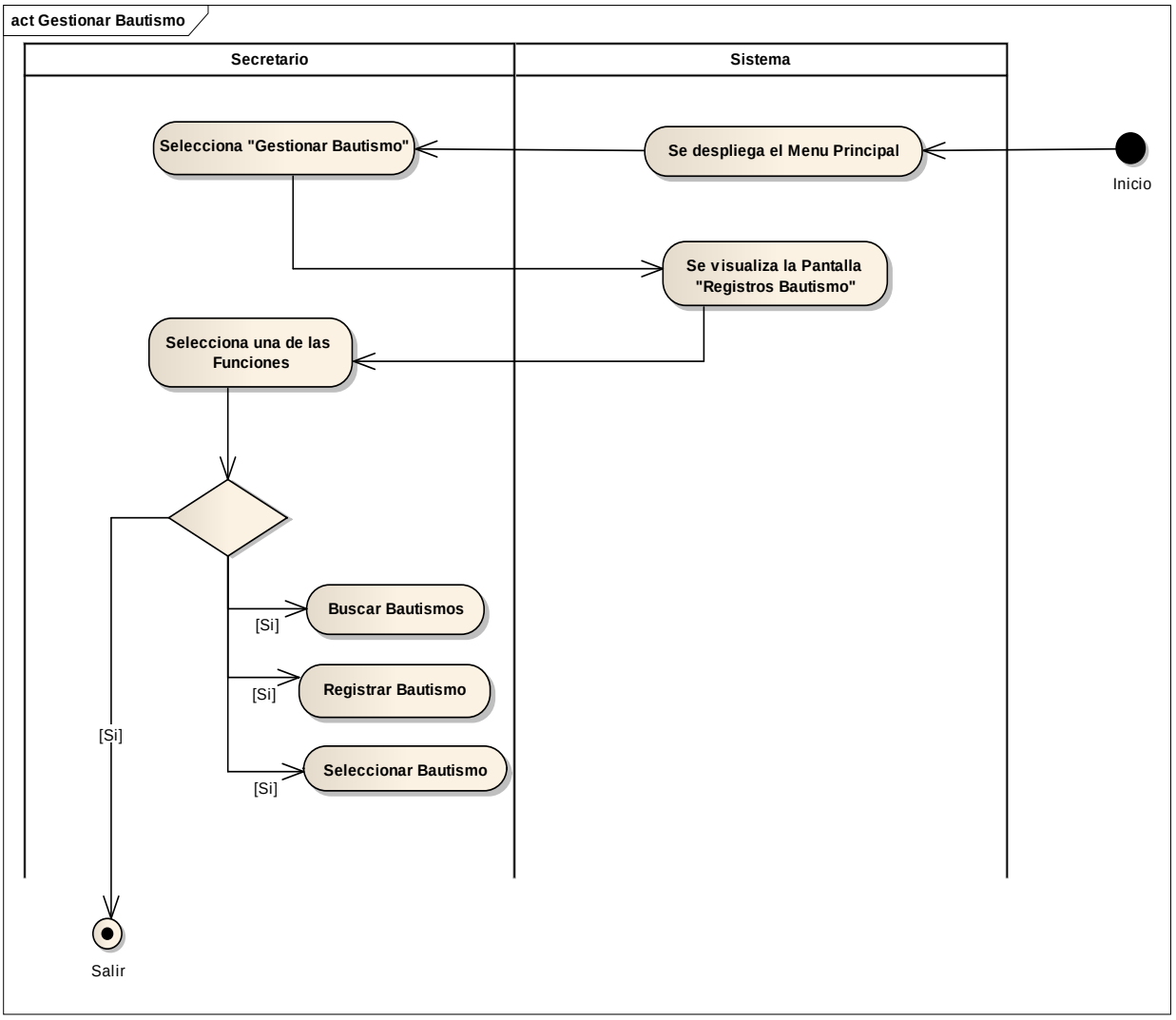


Figura Nº107. Diagrama de actividad: CU Gestionar Bautismo

2.1.2.2.3.10.1.4.21 Diagrama de Actividad: Registrar Bautismo

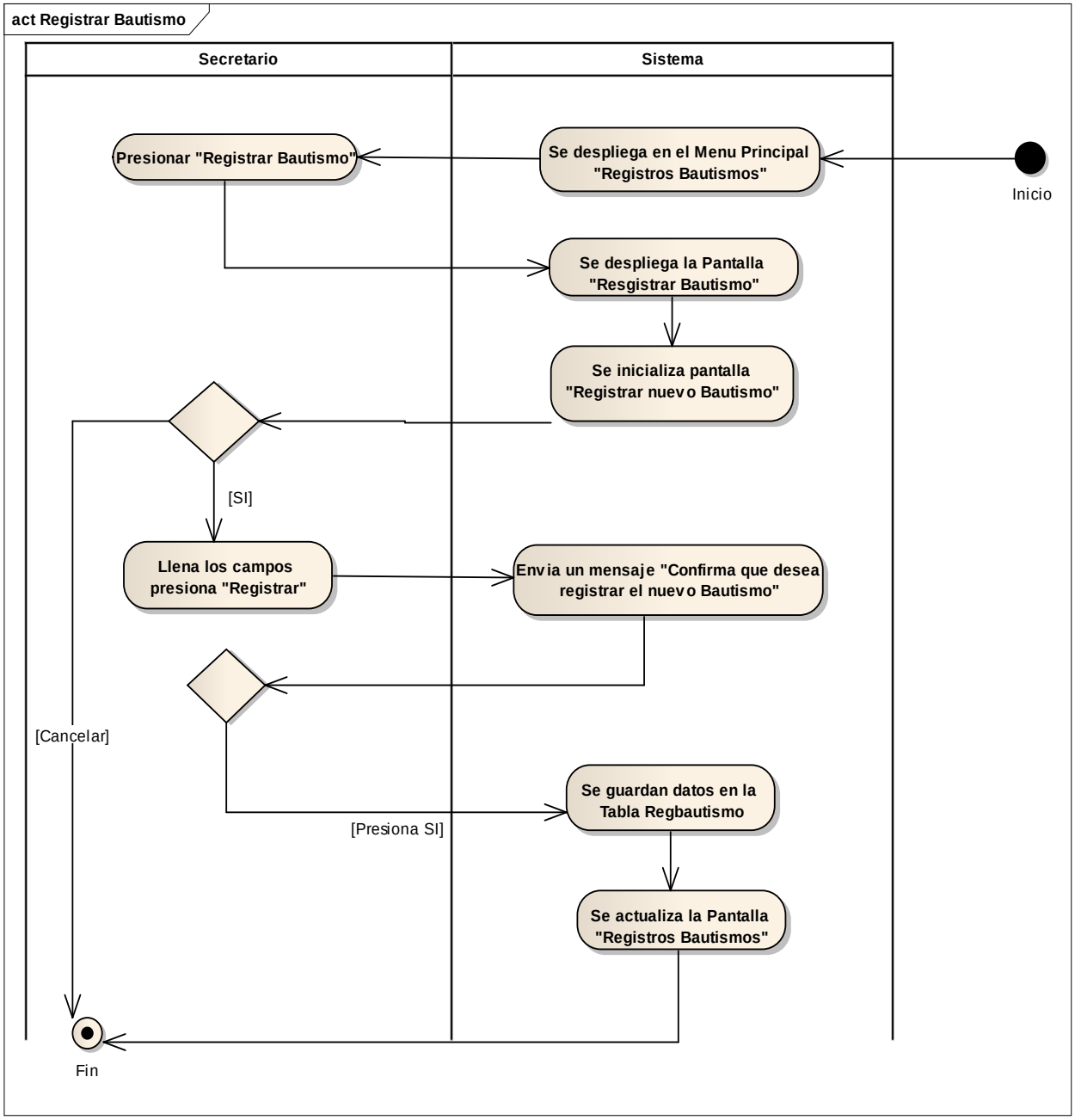


Figura Nº108. Diagrama de actividad: CU Registrar Bautismo

2.1.2.2.3.10.1.4.22 Diagrama de Actividad: Caso de Uso Gestionar Comunión

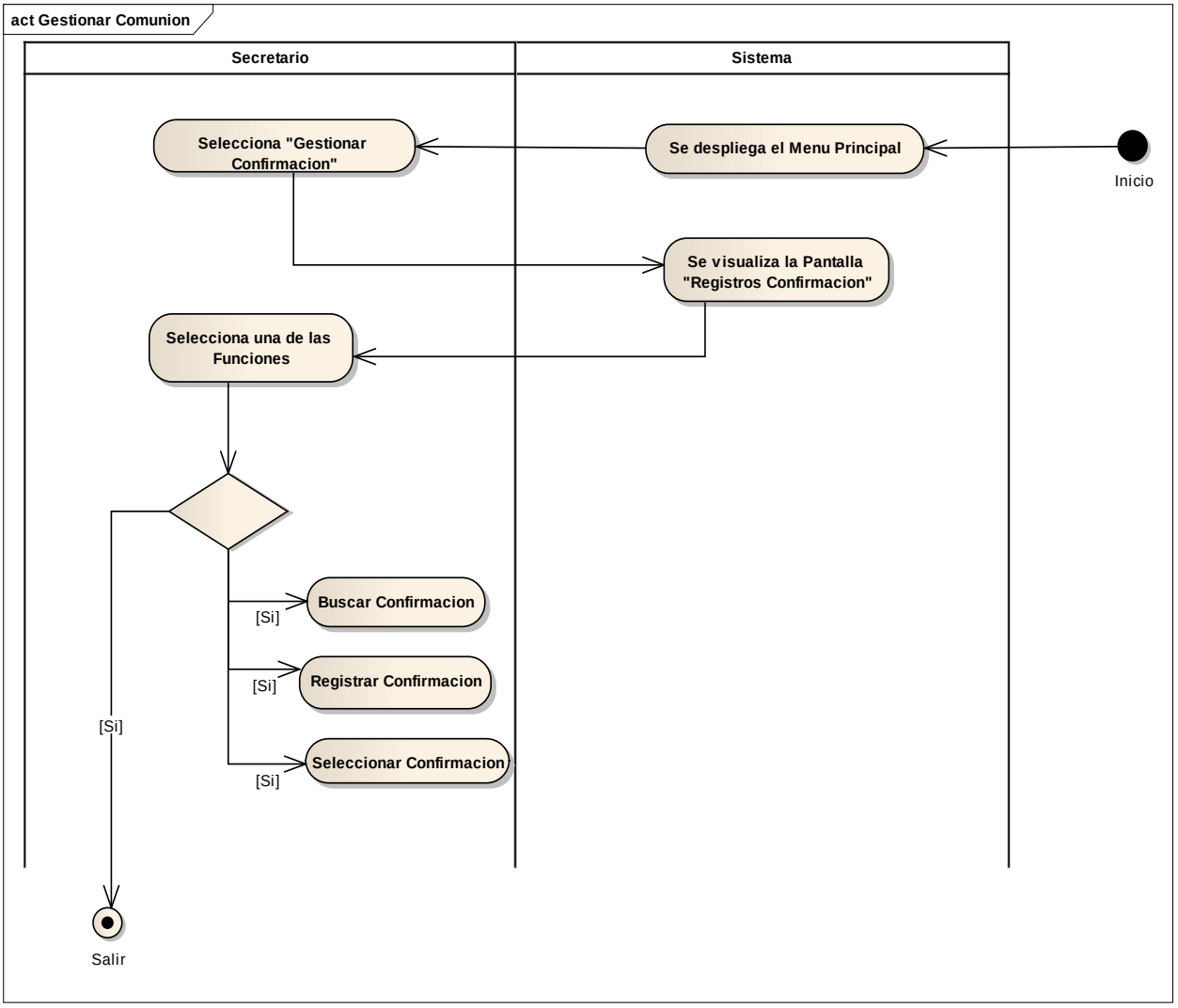


Figura Nº109. Diagrama de actividad: CU Gestionar Comunión

2.1.2.2.3.10.1.4.23 Diagrama de Actividad: Caso de Uso Registrar Comunión

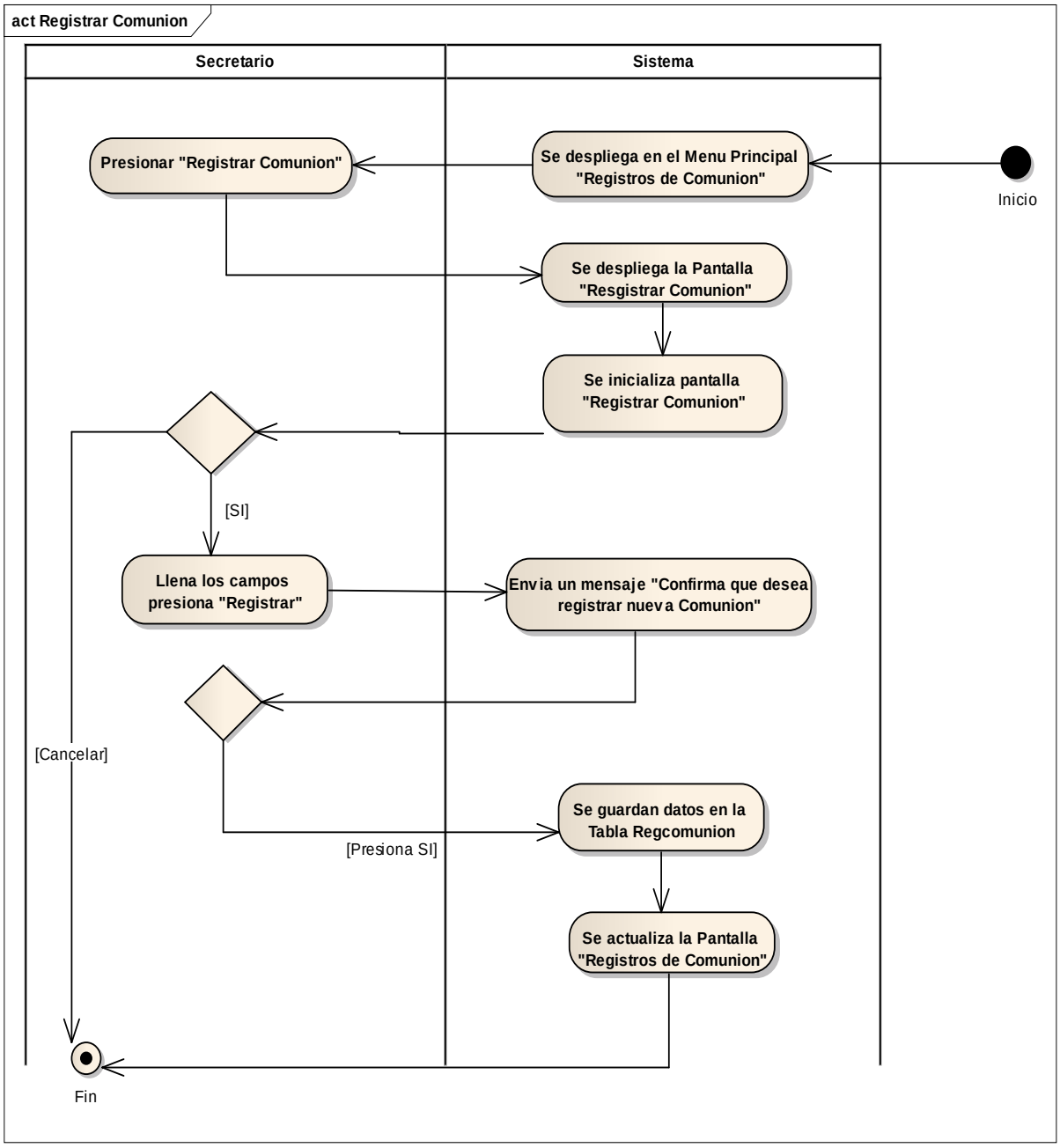


Figura Nº110. Diagrama de actividad: CU Registrar Comunión

2.1.2.2.3.10.1.4.24 Diagrama de Actividad: Caso de Uso Gestionar Confirmación

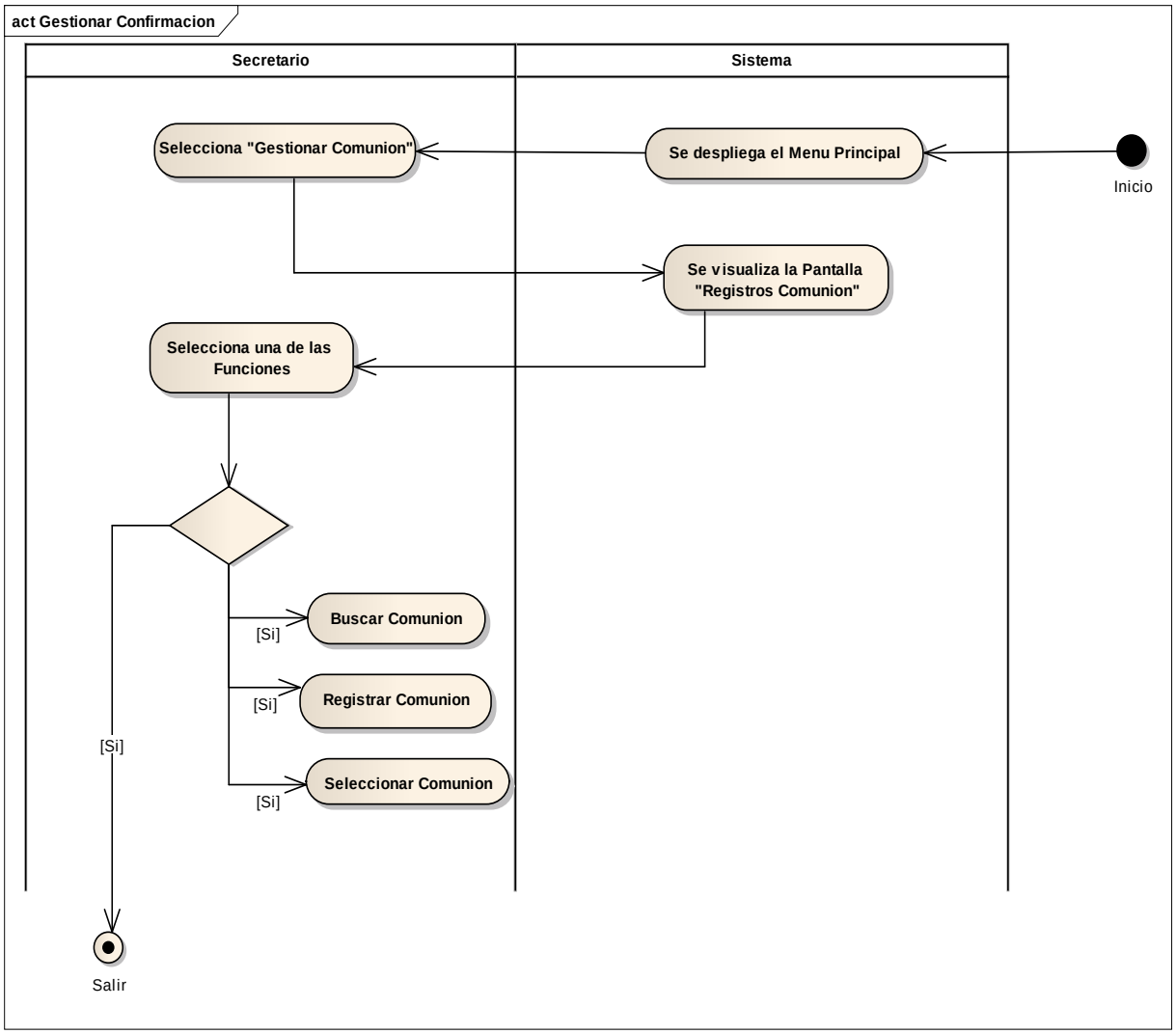


Figura Nº111. Diagrama de actividad: CU Gestionar Confirmación

2.1.2.2.3.10.1.4.25 Diagrama de Actividad: Caso de Uso Registrar Confirmación

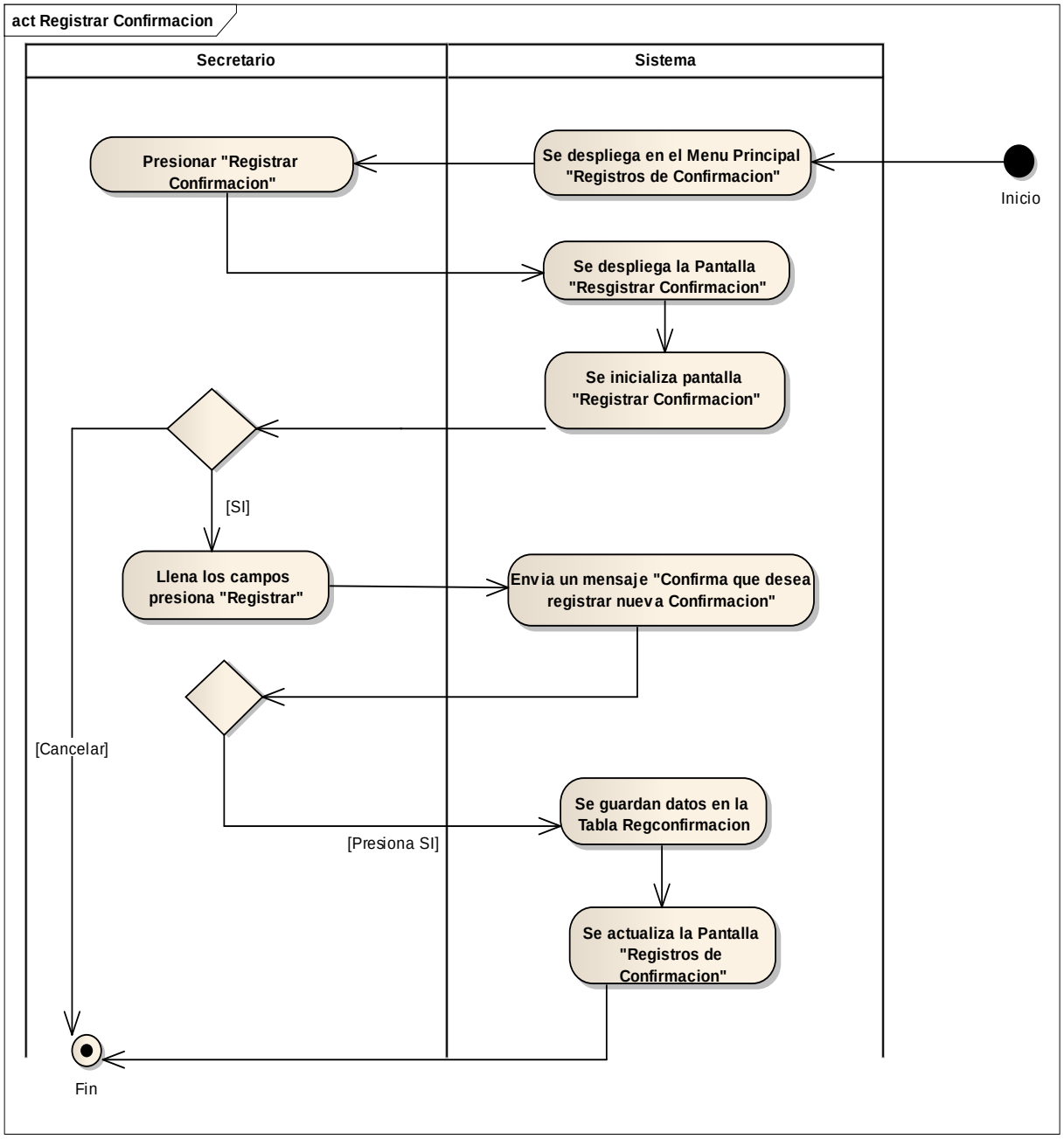


Figura Nº112. Diagrama de actividad: CU Registrar Confirmación

2.1.2.2.3.10.1.4.26 Diagrama de Actividad: Caso de Uso Gestionar Matrimonio

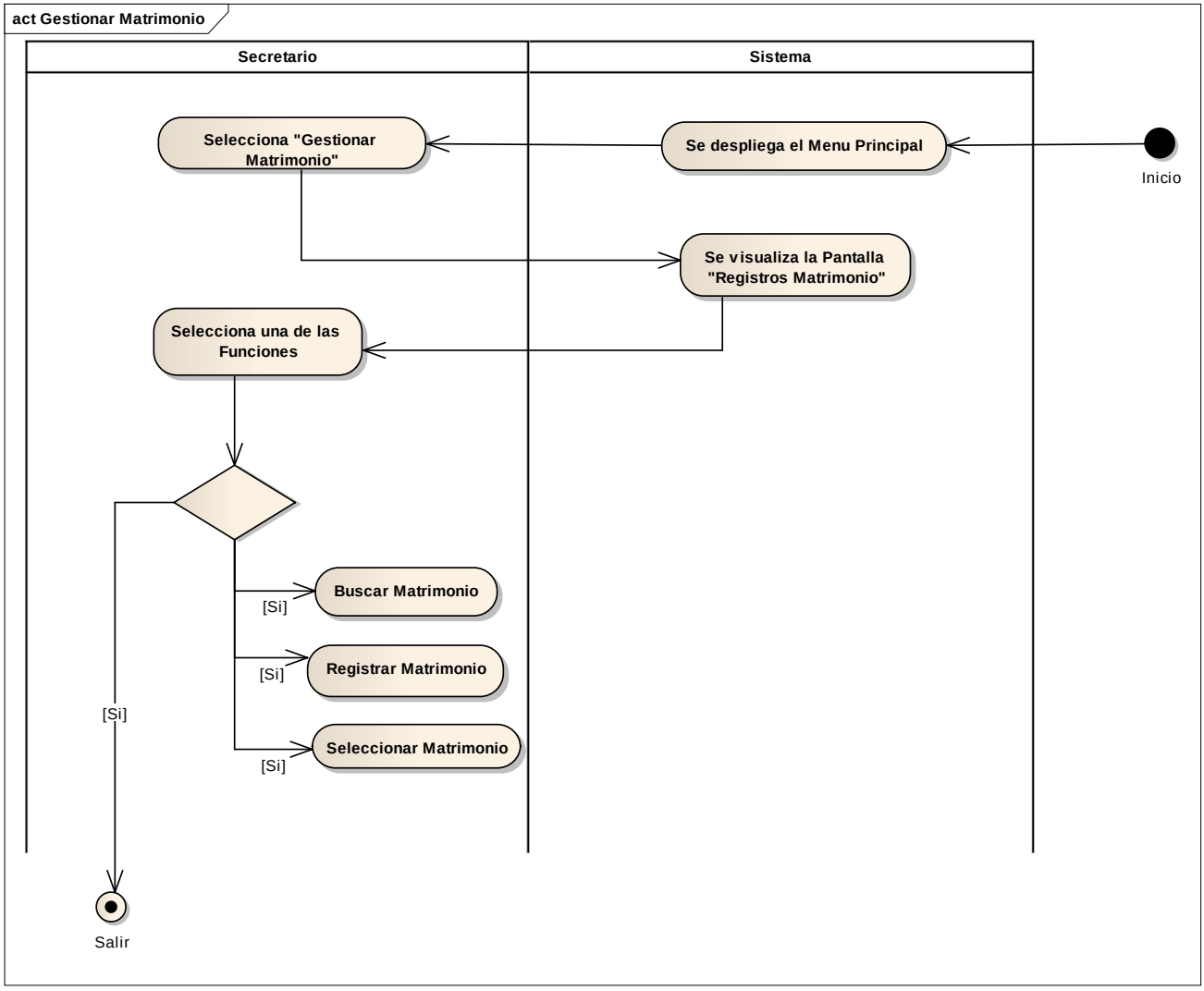


Figura Nº113. Diagrama de actividad: CU Gestionar Matrimonio

2.1.2.2.3.10.1.4.27 Diagrama de Actividad: Caso de Uso Registrar Matrimonio

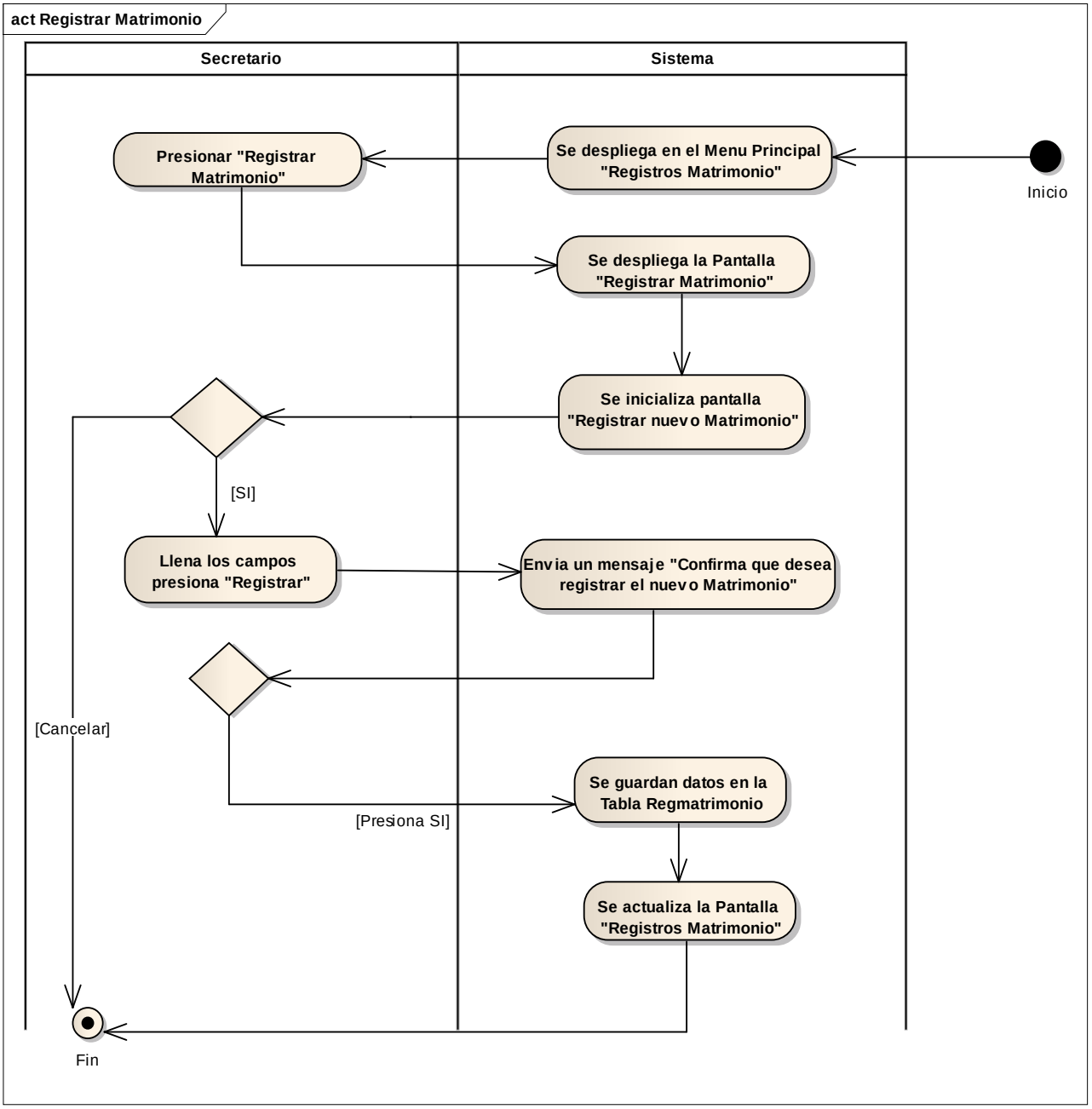


Figura Nº114. Diagrama de actividad: CU Registrar Confirmación

2.1.2.2.3.10.2 Modelado de Diagrama de Secuencias

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. En aplicaciones grandes además de los objetos se muestran también los componentes y casos de uso. El mostrar los componentes tiene sentido ya que se trata de objetos reutilizables, en cuanto a los casos de uso hay que recordar que se implementan como objetos cuyo rol es encapsular lo definido en el caso de uso.

Para mostrar la interacción con el usuario o con otro sistema se introducen en los diagramas de secuencia las ***boundary classes***. En las primeras fases de diseño el propósito de introducir estas clases es capturar y documentar los requisitos de interfaz, pero no el mostrar cómo se va a implementar dicha interfaz.

Los diagramas de secuencia, formalmente diagramas de traza de eventos o de interacción de objetos, se utilizan con frecuencia para validar los casos de uso. Documentan el diseño desde el punto de vista de los casos de uso. Observando qué mensajes se envían a los objetos, componentes o casos de uso y viendo a grosso modo cuanto tiempo consume el método invocado, los diagramas de secuencia nos ayudan a comprender los cuellos de botella potenciales, para así poder eliminarlos. A la hora de documentar un diagrama de secuencia resulta importante mantener los enlaces de los mensajes a los métodos apropiados del diagrama de clases.

2.1.2.2.3.10.2.1 Diagrama de Secuencias

2.1.2.2.3.10.2.1.1 Diagrama de Secuencia: Caso de Uso Acceder al Sistema

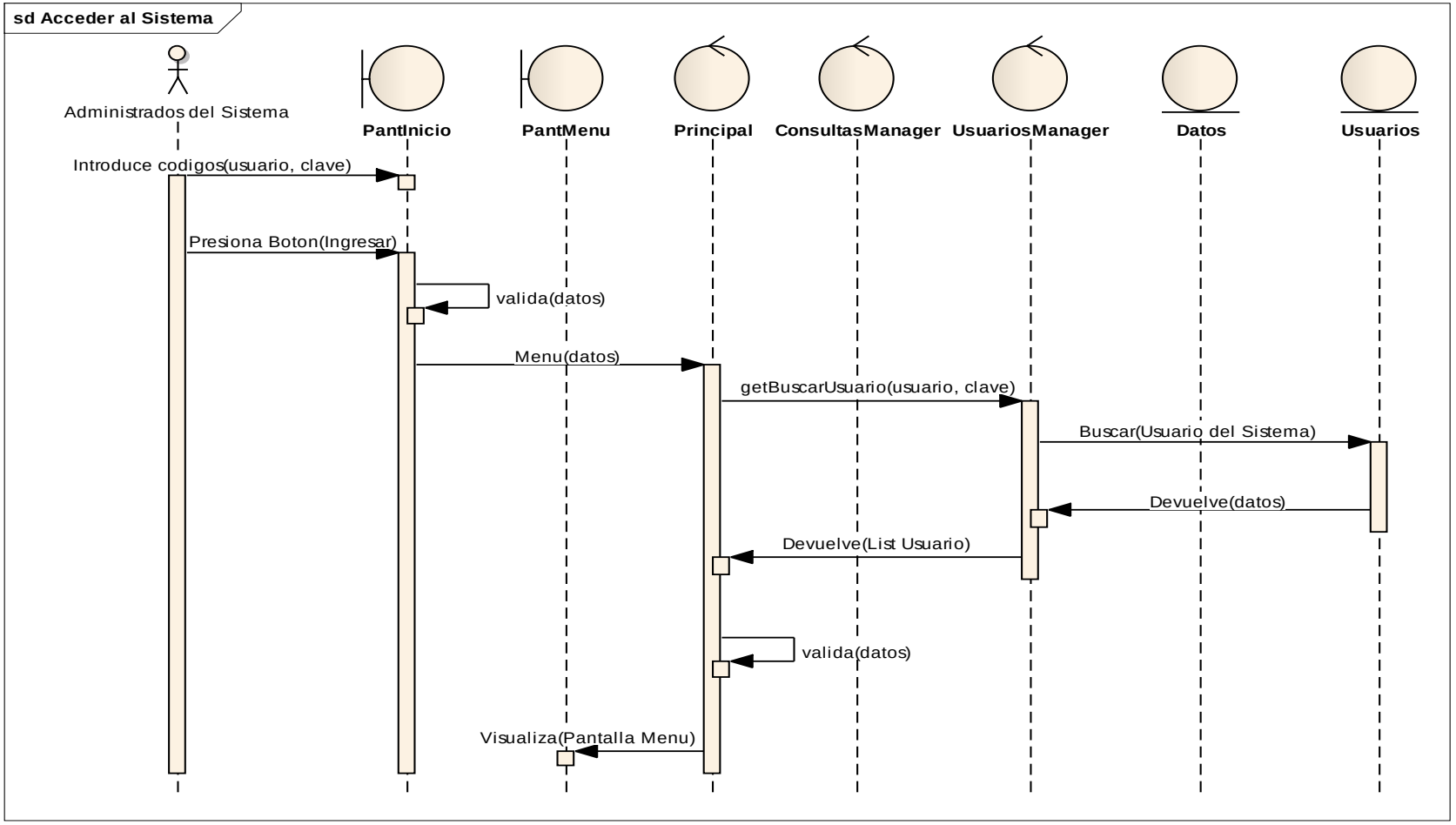


Figura Nº115. Diagrama de secuencia: CU Acceder al Sistema

2.1.2.2.3.10.2.1.2 Diagrama de Secuencia: Caso de Uso Administrar Perfil

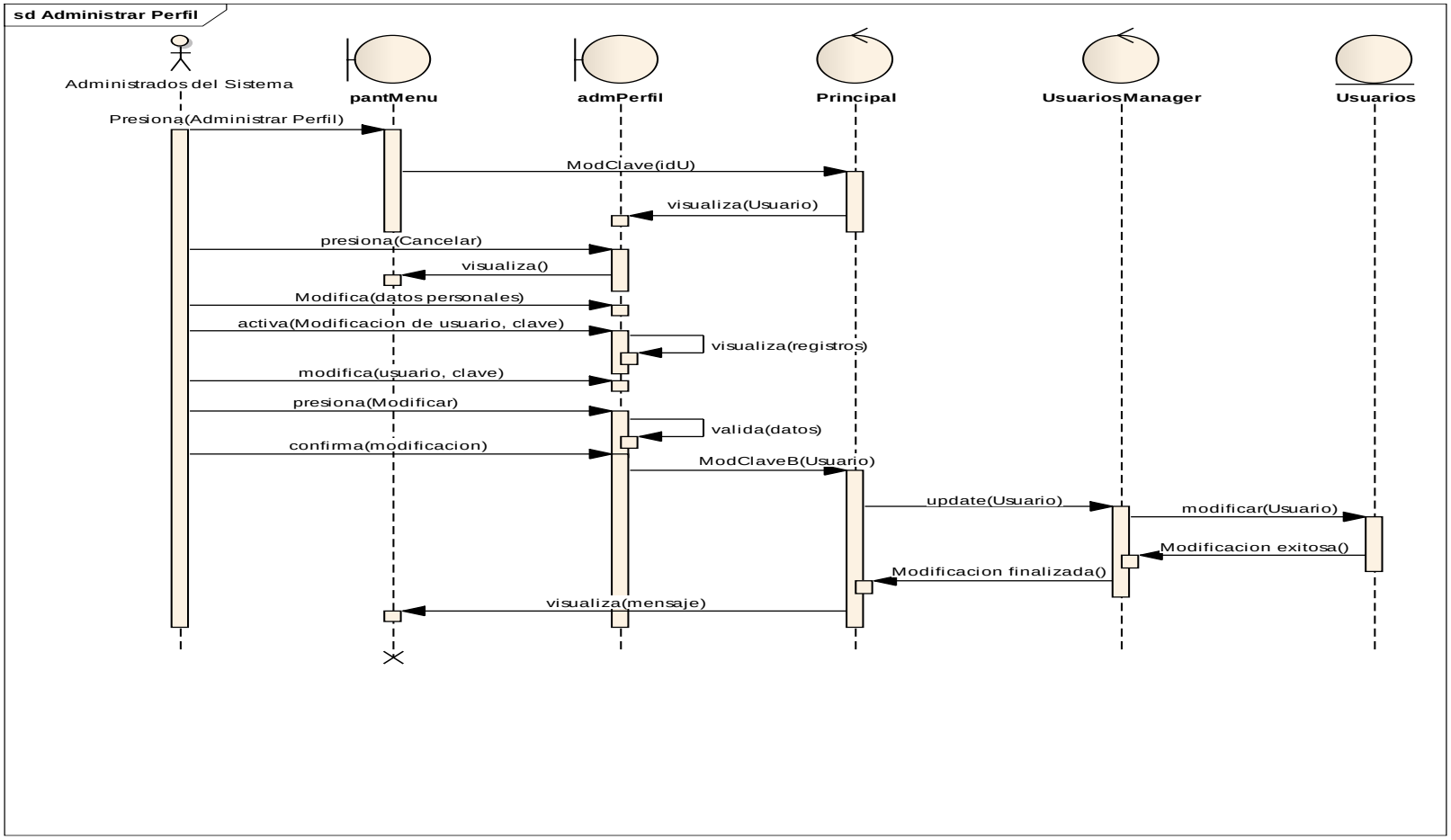


Figura Nº116. Diagrama de secuencia: CU Administrar Perfil

2.1.2.2.3.10.2.1.3 Diagrama de Secuencia: Caso de Uso Gestionar Resguardos

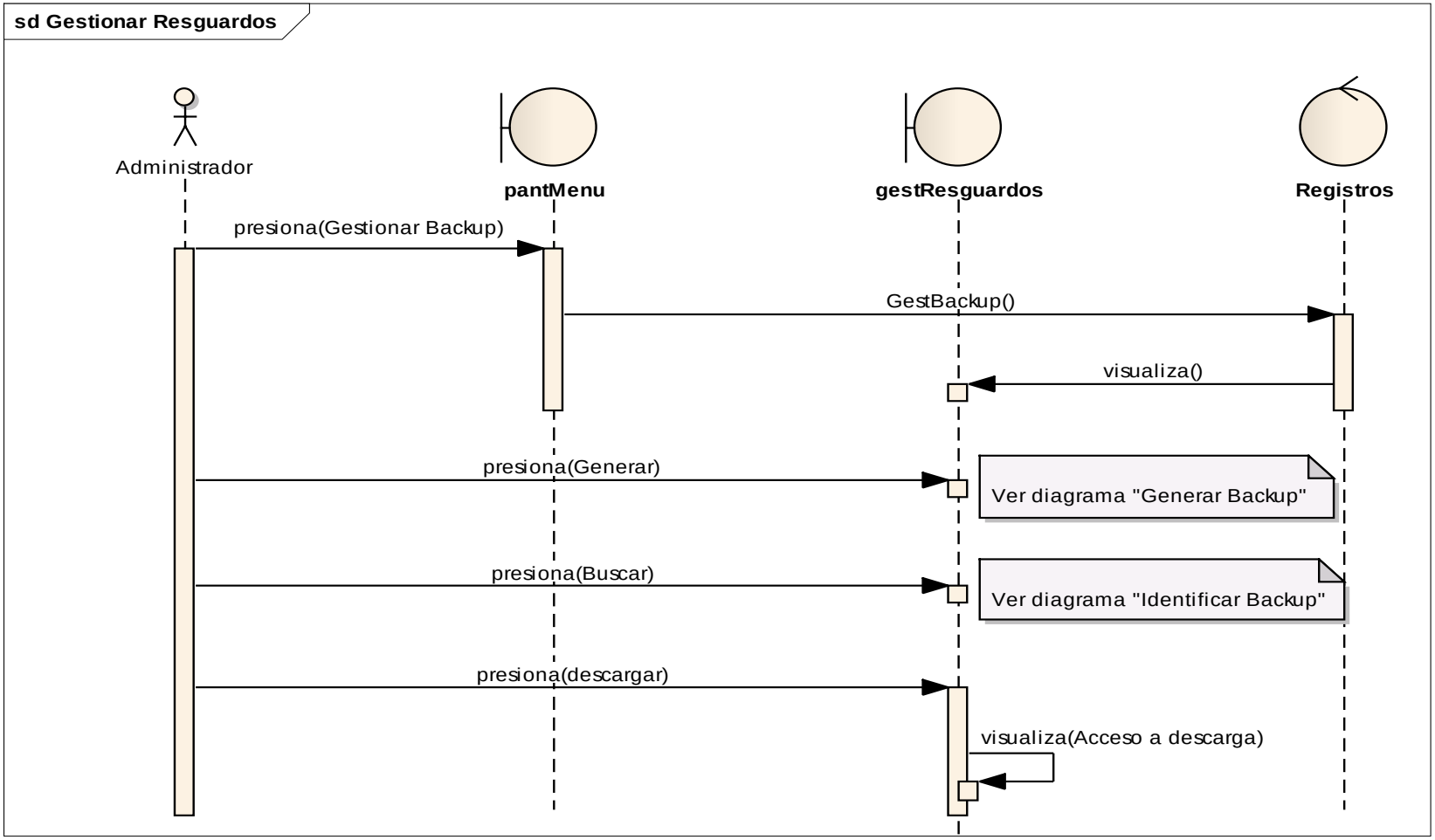


Figura Nº117. Diagrama de secuencia: CU Gestionar Resguardos

2.1.2.2.3.10.2.1.4 Diagrama de Secuencia: Caso de Uso Identificar Resguardo

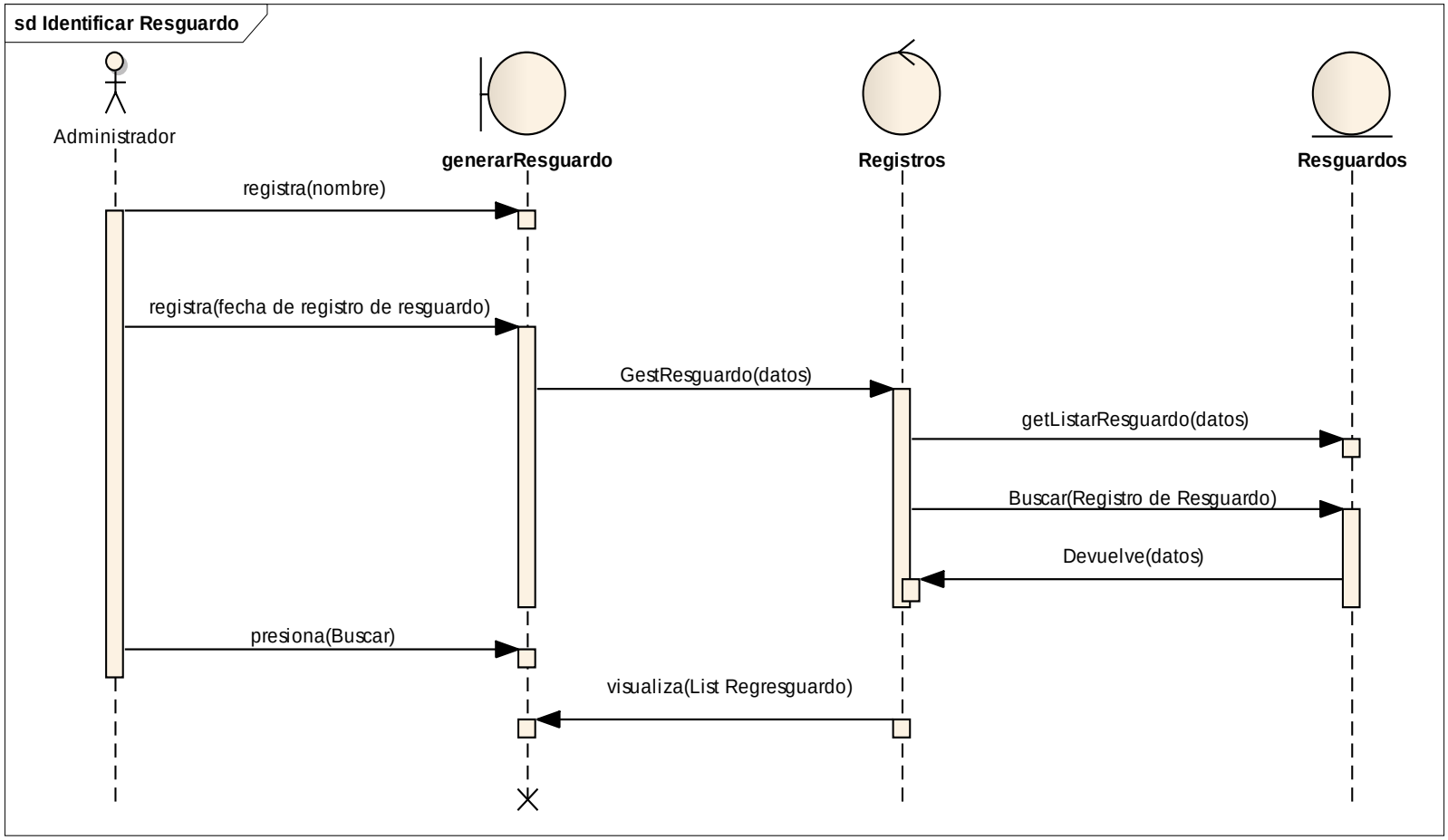


Figura Nº118. Diagrama de secuencia: CU Identificar Resguardo

2.1.2.2.3.10.2.1.5 Diagrama de Secuencia: Caso de Uso Generar Resguardo

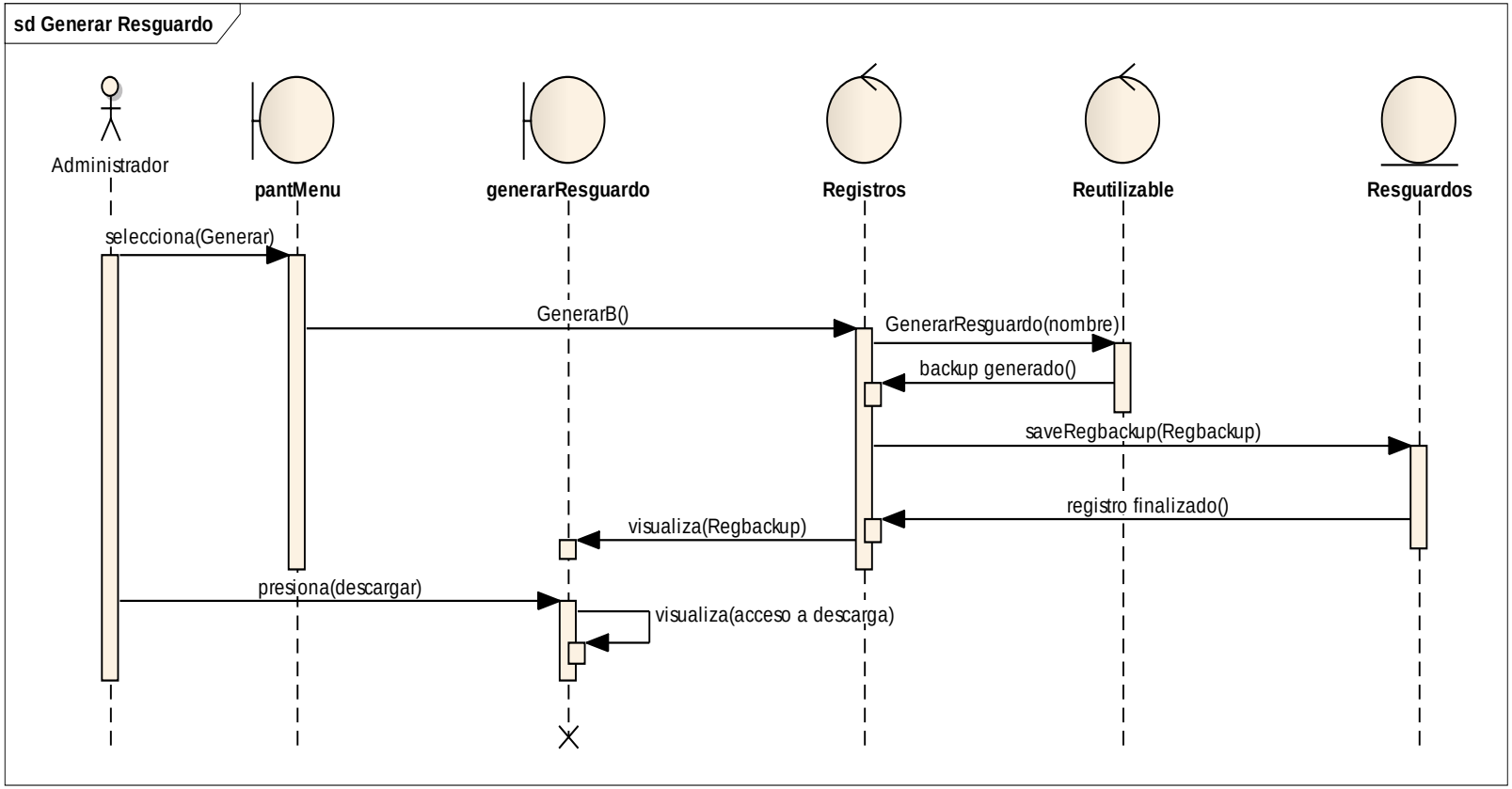


Figura Nº119. Diagrama de secuencia: CU Generar Resguardo

2.1.2.2.3.10.2.1.6 Diagrama de Secuencia: Caso de Uso Gestionar Usuarios

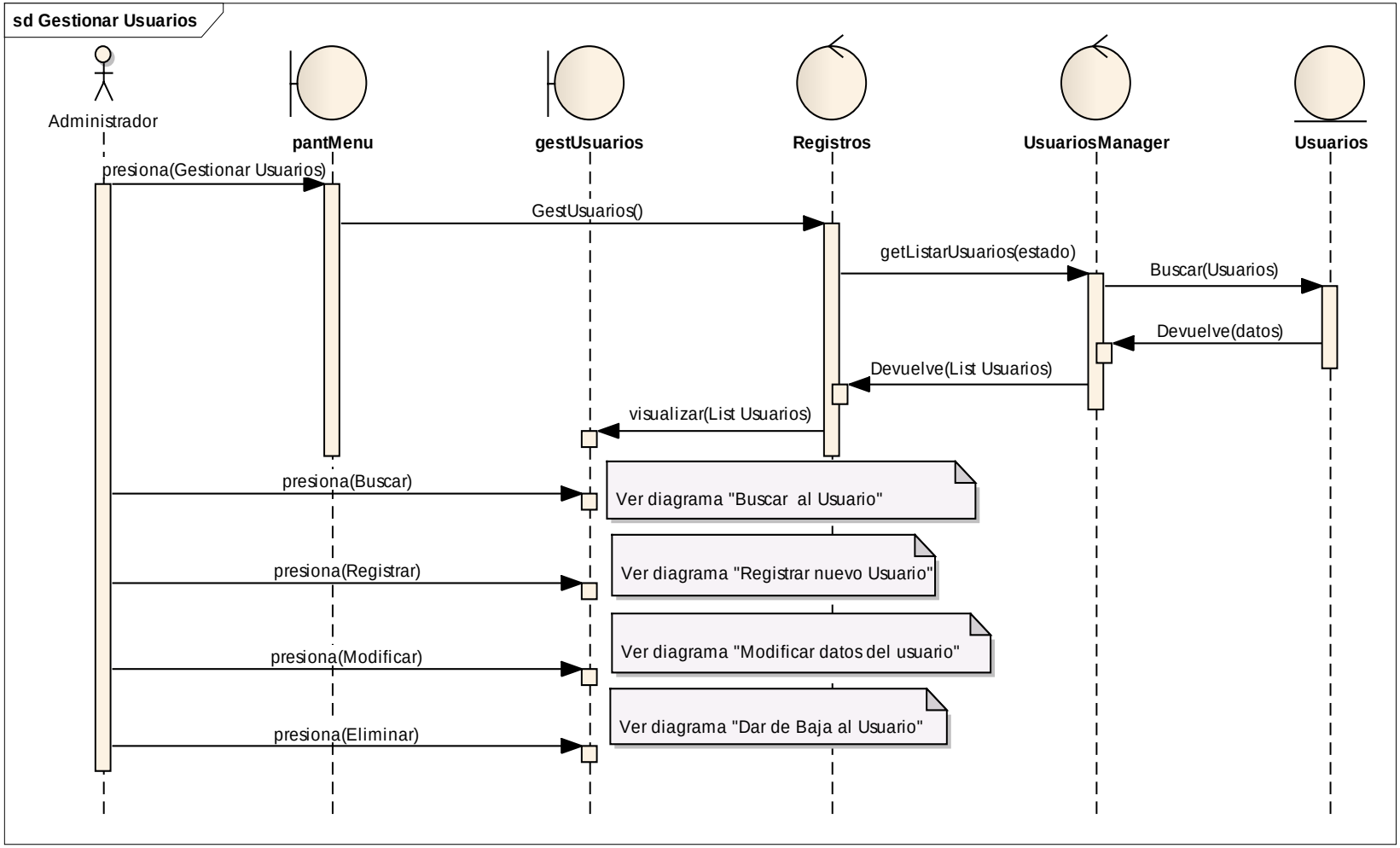


Figura Nº120. Diagrama de secuencia: CU Gestionar Usuarios

2.1.2.2.3.10.2.1.7 Diagrama de Secuencia: Caso de Uso Registrar nuevo Usuario

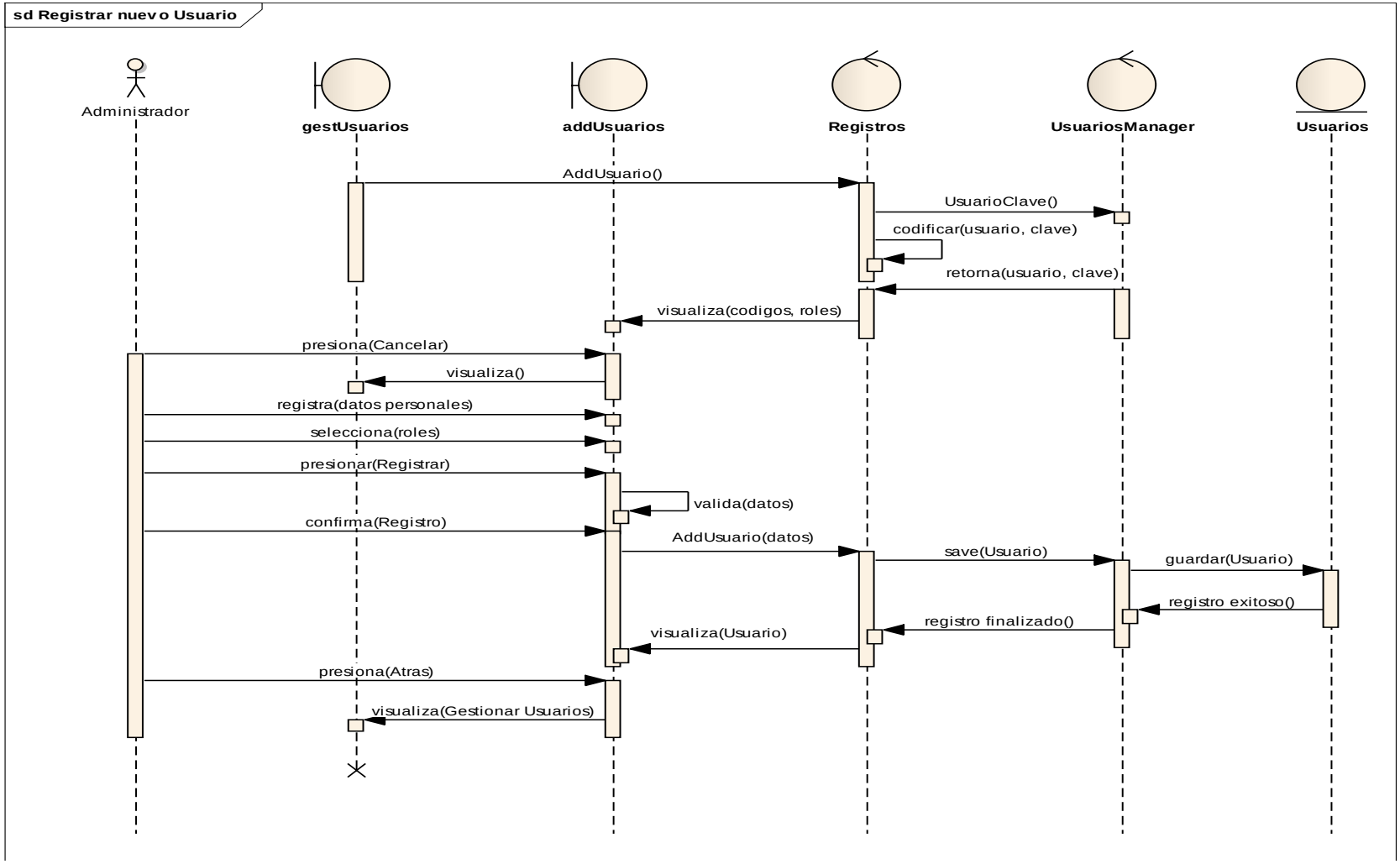


Figura Nº121. Diagrama de secuencia: CU Registrar Nuevo Usuario

2.1.2.2.3.10.2.1.8 Diagrama de Secuencia: Caso de Uso Modificar datos del Usuario

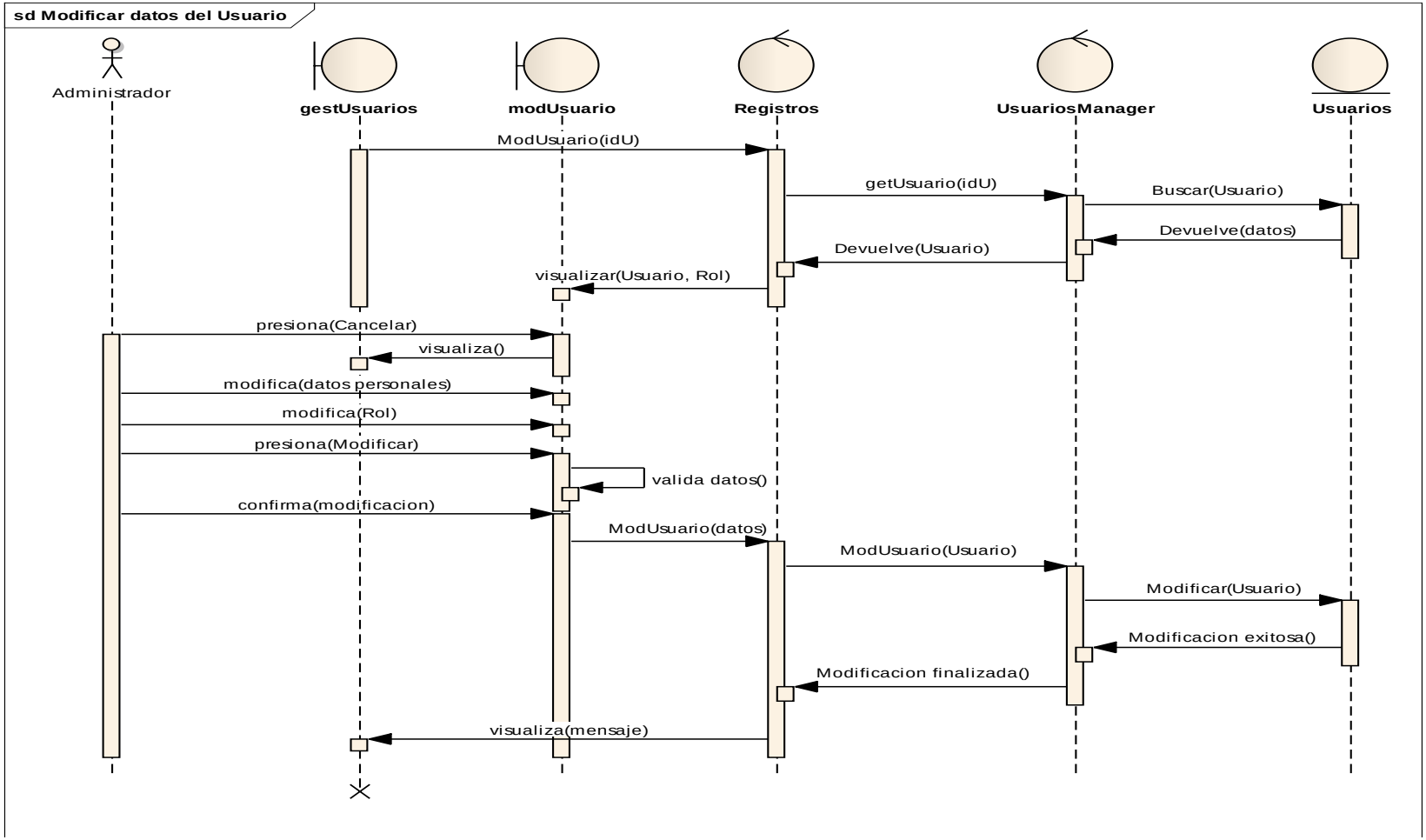


Figura Nº122. Diagrama de secuencia: CU Modificar datos del Usuario

2.1.2.2.3.10.2.1.9 Diagrama de Secuencia: Caso de Uso Dar de baja al Usuario

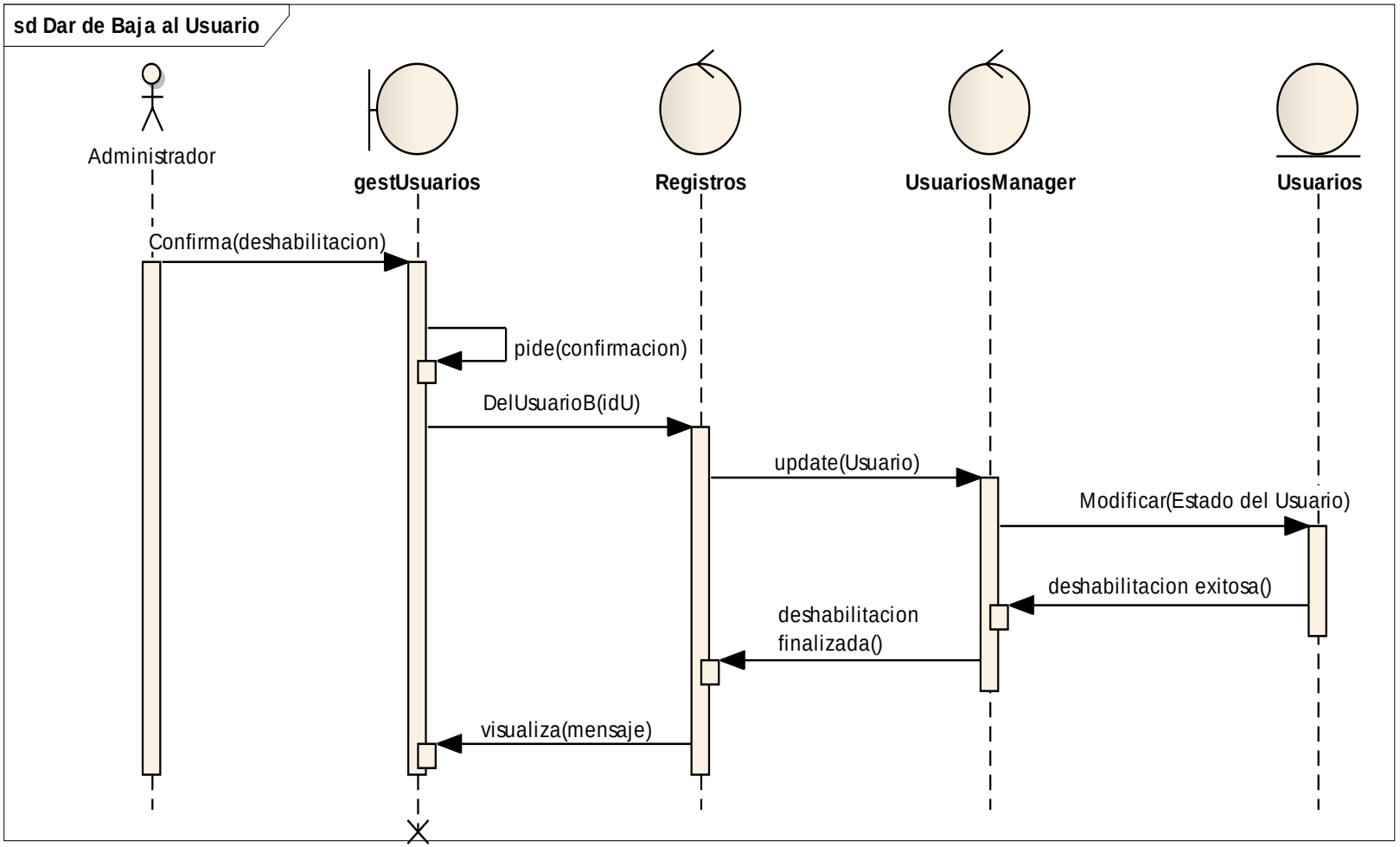


Figura Nº123. Diagrama de secuencia: CU Dar de baja al Usuario

2.1.2.2.3.10.2.1.10 Diagrama de Secuencia: Caso de Uso Gestionar Feligreses

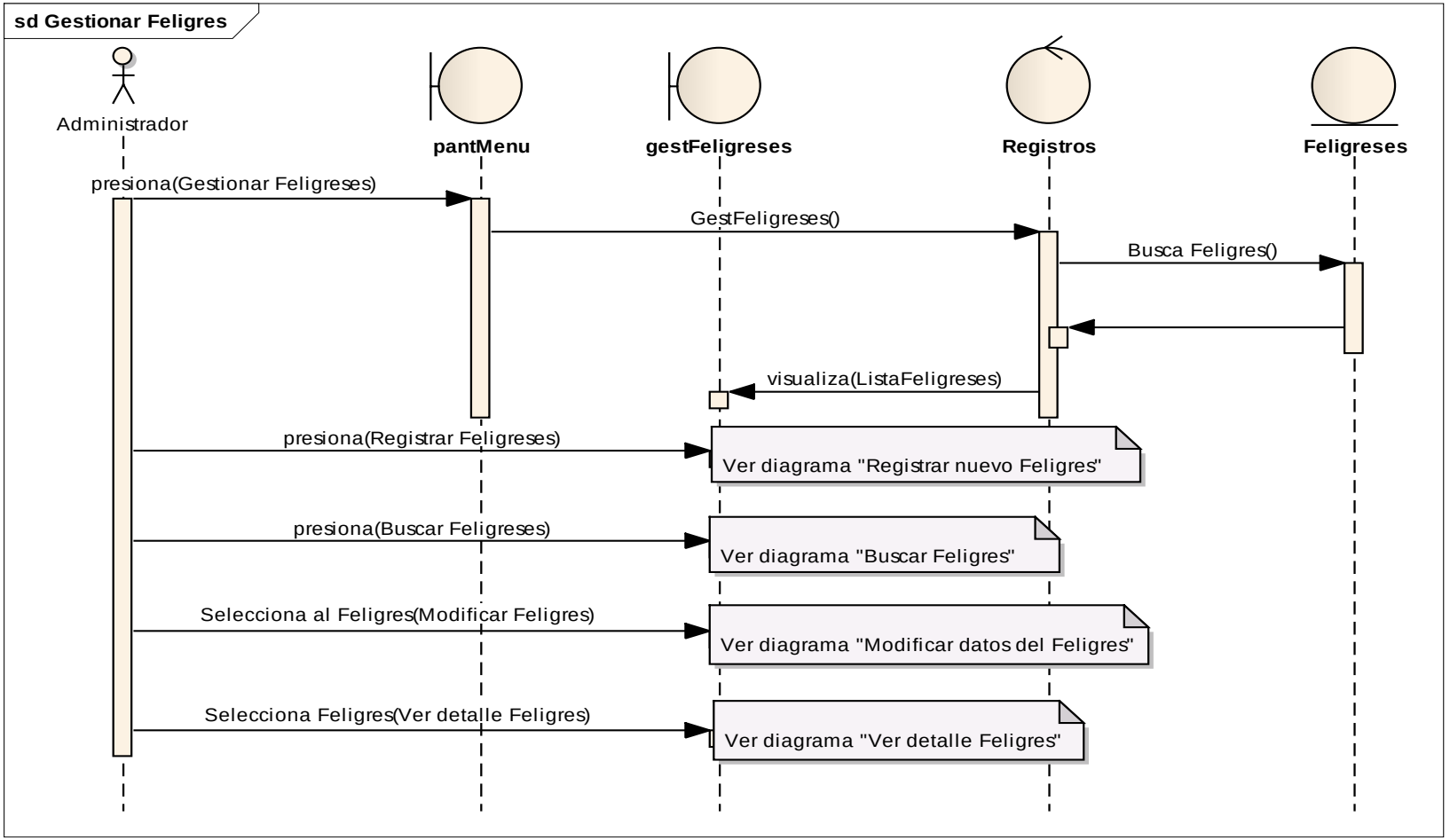


Figura Nº124. Diagrama de secuencia: CU Gestionar Feligreses

2.1.2.2.3.10.2.1.11 Diagrama de Secuencia: Caso de Uso Registrar nuevo Feligrés

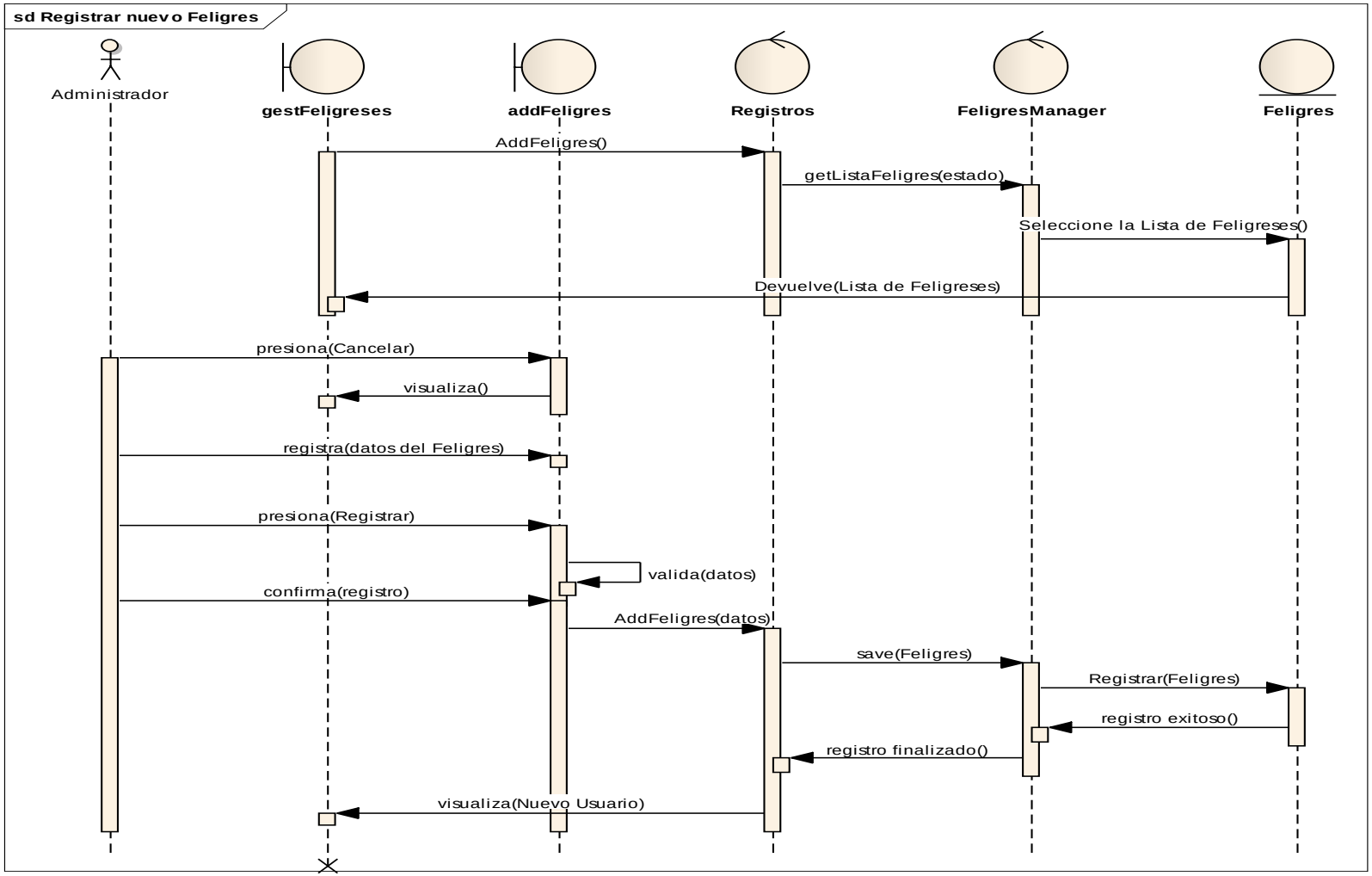


Figura Nº125. Diagrama de secuencia: CU Registrar nuevo Feligrés

2.1.2.2.3.10.2.1.12 Diagrama de Secuencia: Caso de Uso Modificar datos Feligreses

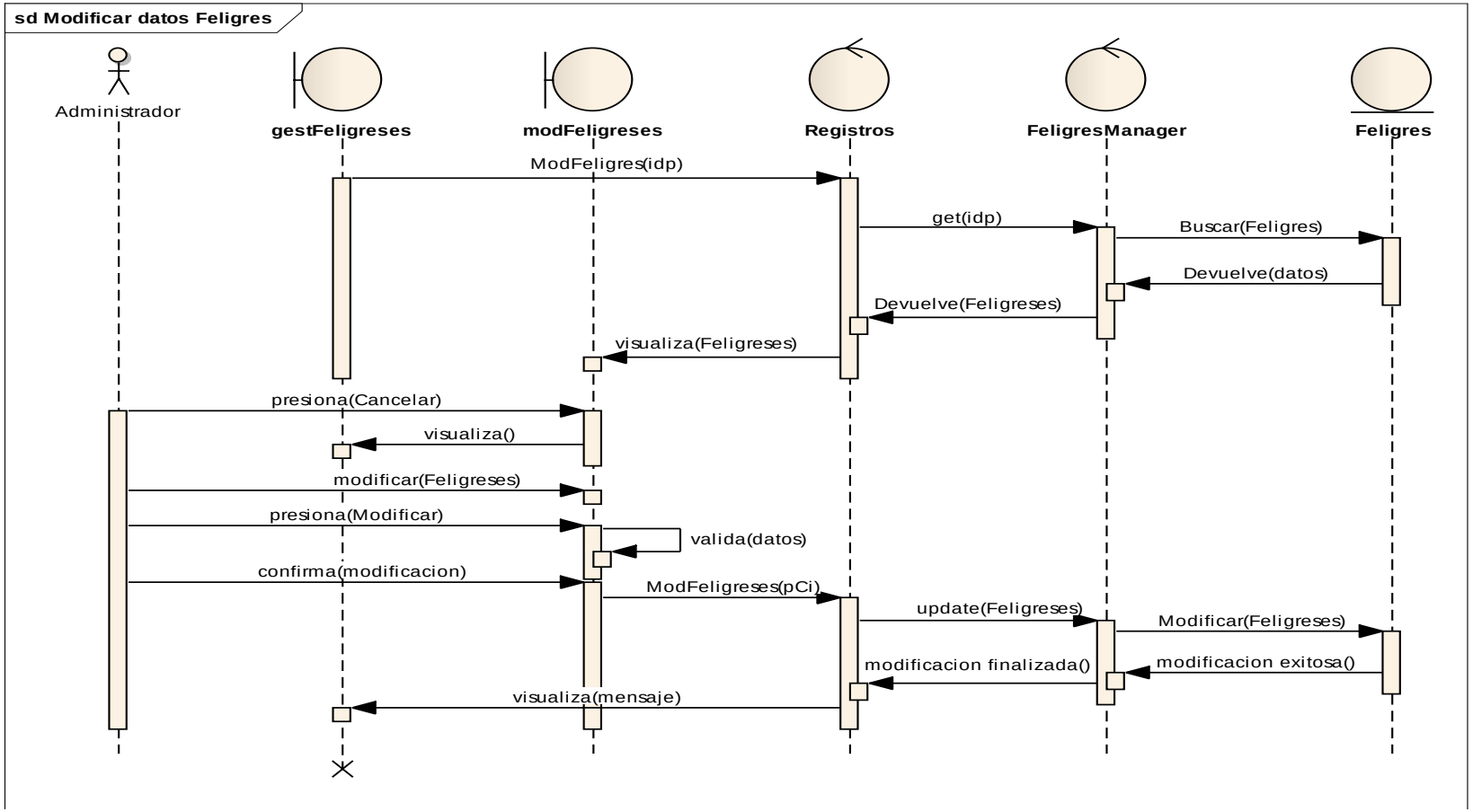


Figura Nº126. Diagrama de secuencia: CU Modificar datos de Feligreses

2.1.2.2.3.10.2.1.13 Diagrama de Secuencia: Caso de Uso Buscar Feligreses

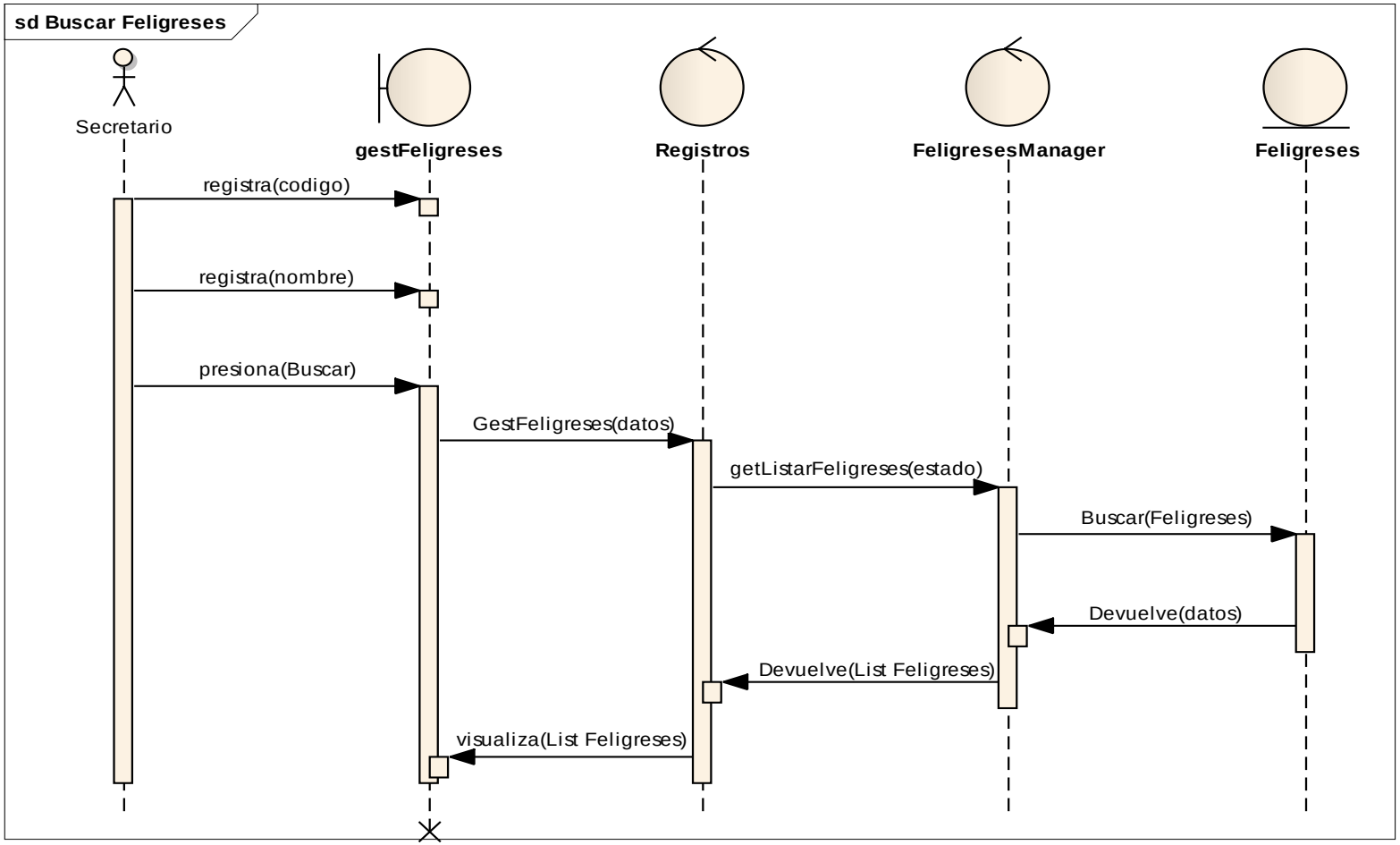


Figura Nº127. Diagrama de secuencia: CU Buscar Feligreses

2.1.2.2.3.10.2.1.14 Diagrama de Secuencia: Caso de Uso Ver detalle Feligreses

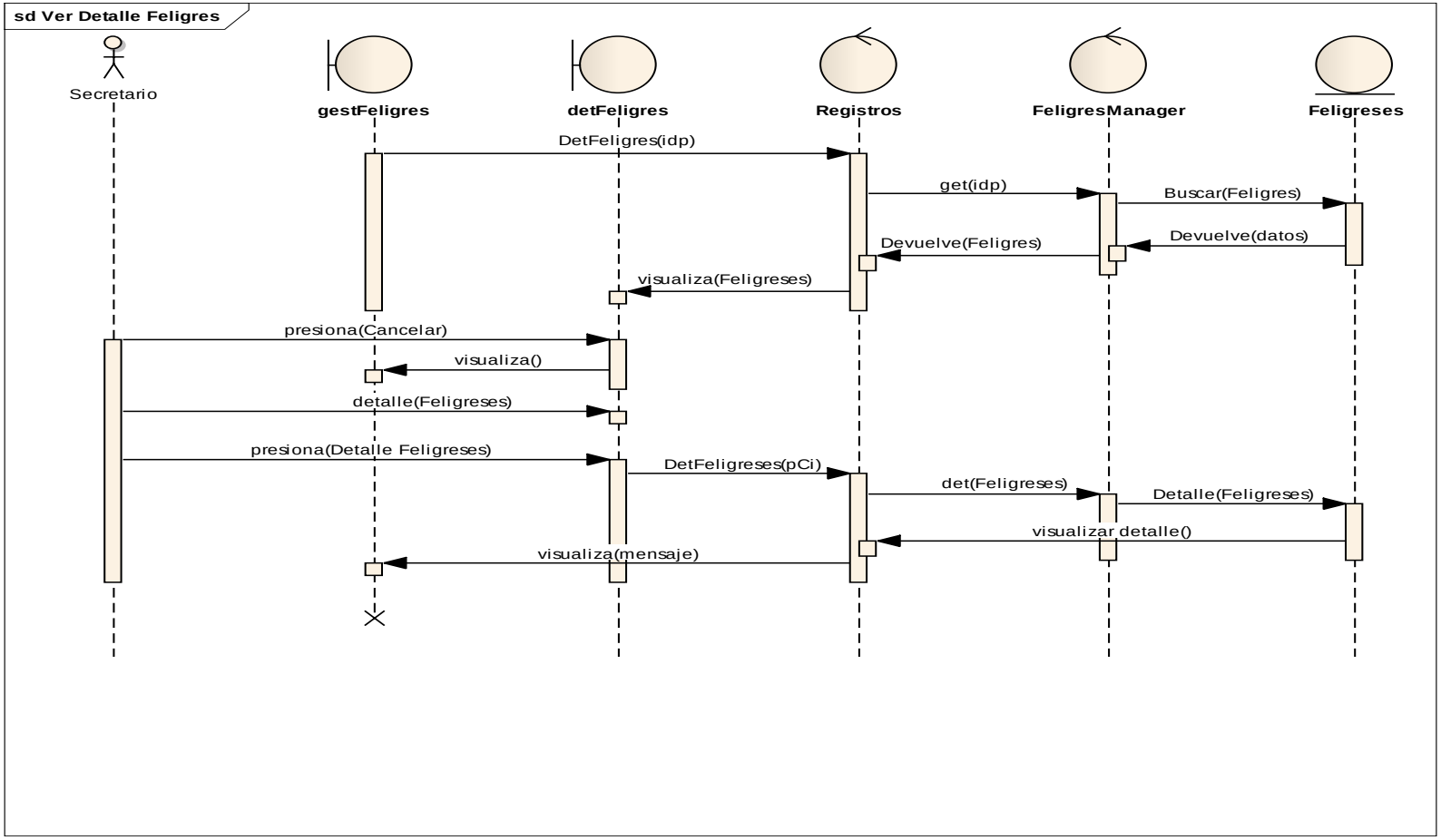


Figura Nº128. Diagrama de secuencia: CU Ver detalle Feligreses

2.1.2.2.3.10.2.1.15 Diagrama de Secuencia: Caso de Uso Gestionar Parroquia

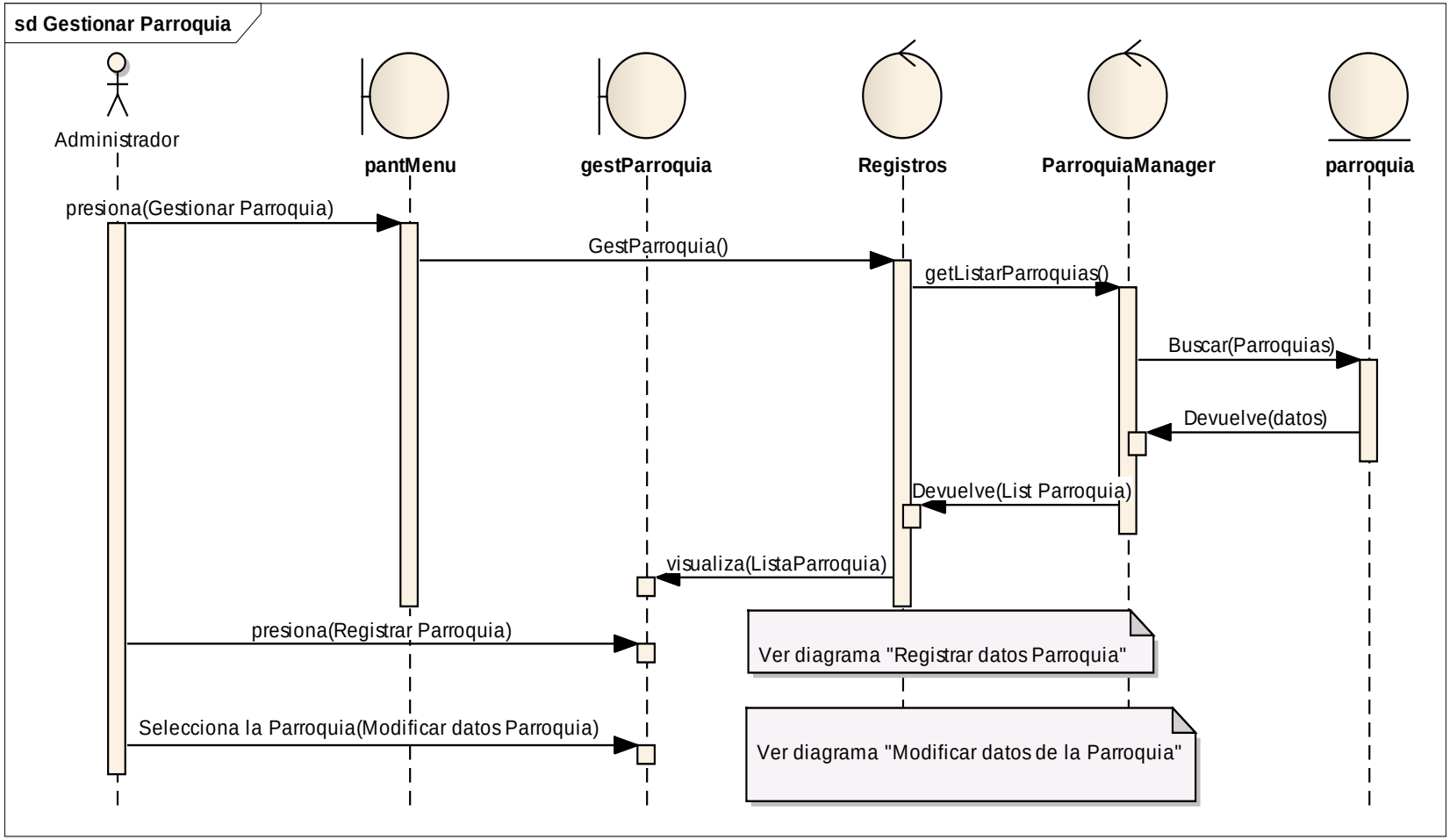


Figura Nº129. Diagrama de secuencia: CU Gestionar Parroquia

2.1.2.2.3.10.2.1.16 Diagrama de Secuencia: Caso de Uso Registrar datos Parroquia

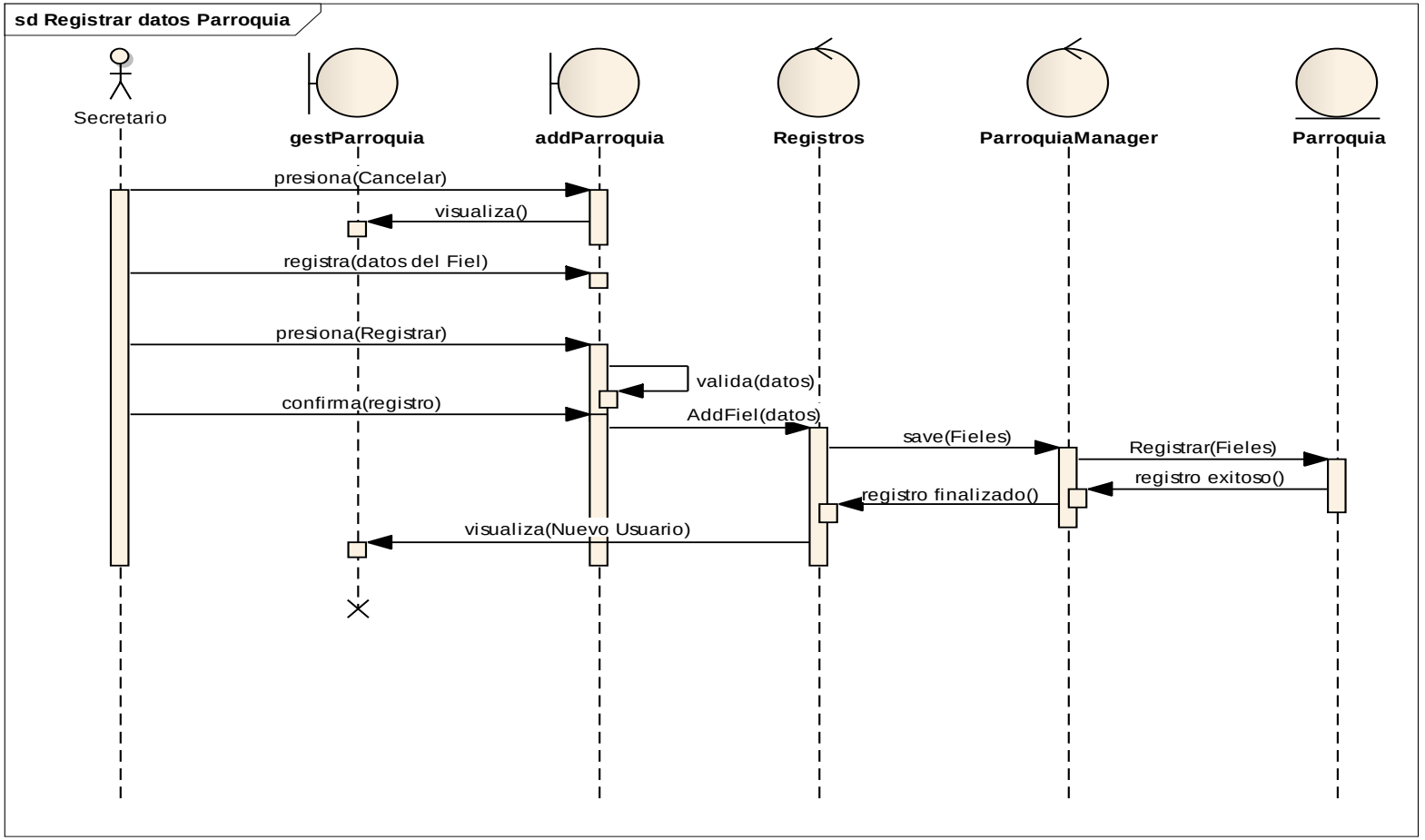


Figura Nº130. Diagrama de secuencia: CU Registrar datos Parroquia

2.1.2.2.3.10.2.1.17 Diagrama de Secuencia: Caso de Uso Gestionar Libro

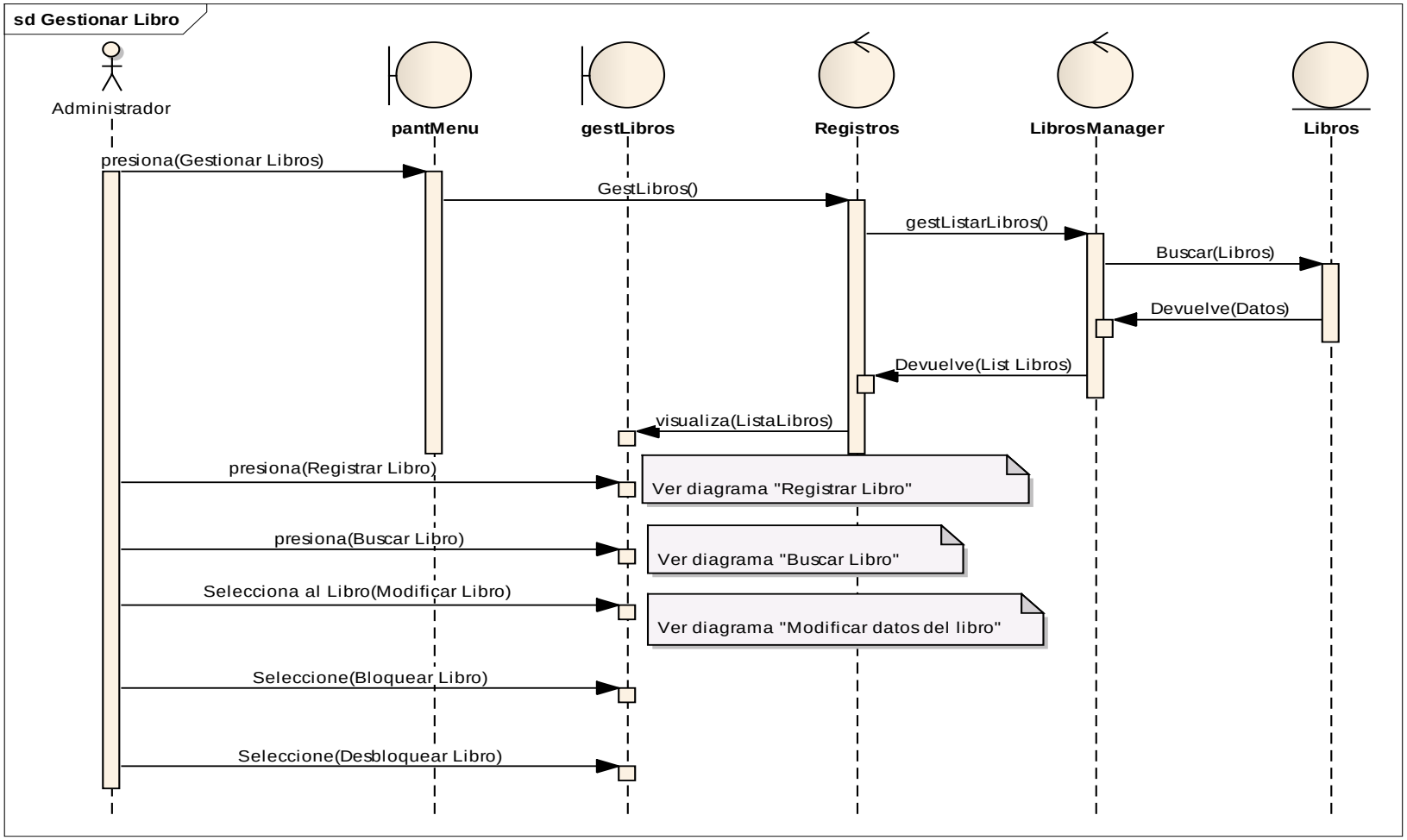


Figura Nº131. Diagrama de secuencia: CU Gestionar Libro

2.1.2.2.3.10.2.1.18 Diagrama de Secuencia: Caso de Uso Registrar nuevo Libro

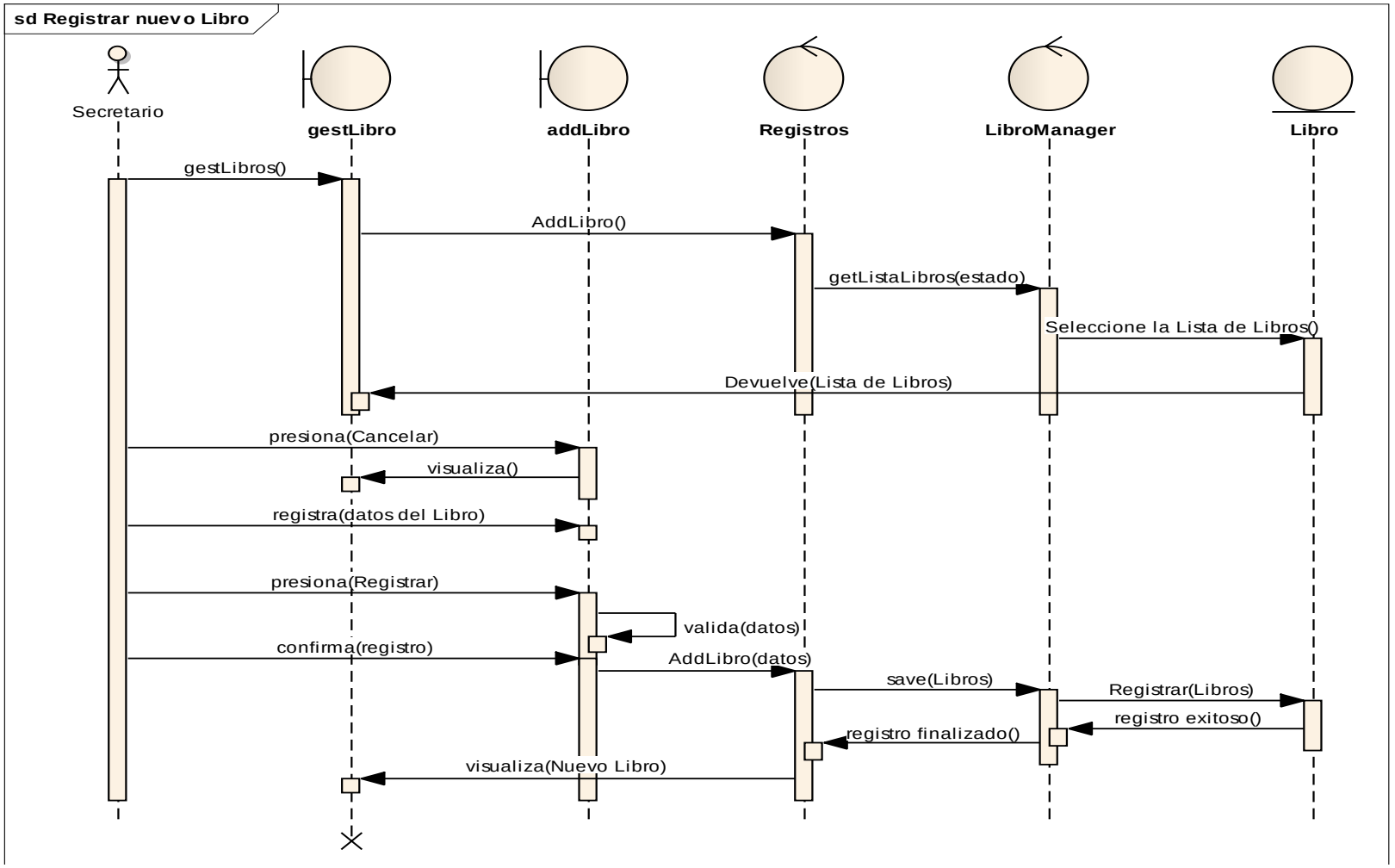


Figura Nº132. Diagrama de secuencia: CU Registrar nuevo Libro

2.1.2.2.3.10.2.1.19 Diagrama de Secuencia: Caso de Uso Modificar Libro

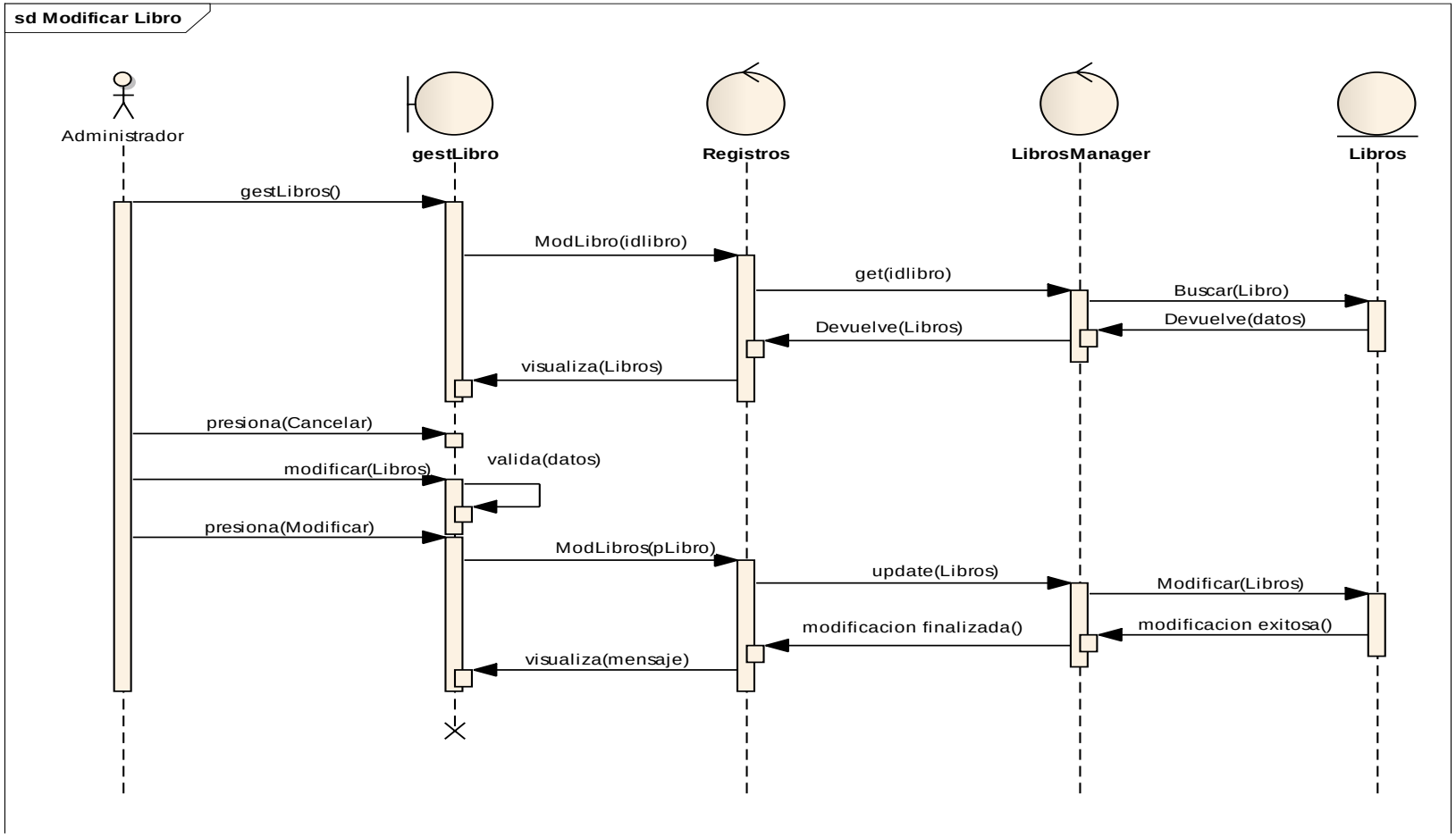


Figura Nº133. Diagrama de secuencia: CU Modificar Libro

2.1.2.2.3.10.2.1.20 Diagrama de Secuencia: Caso de Uso Buscar Libro

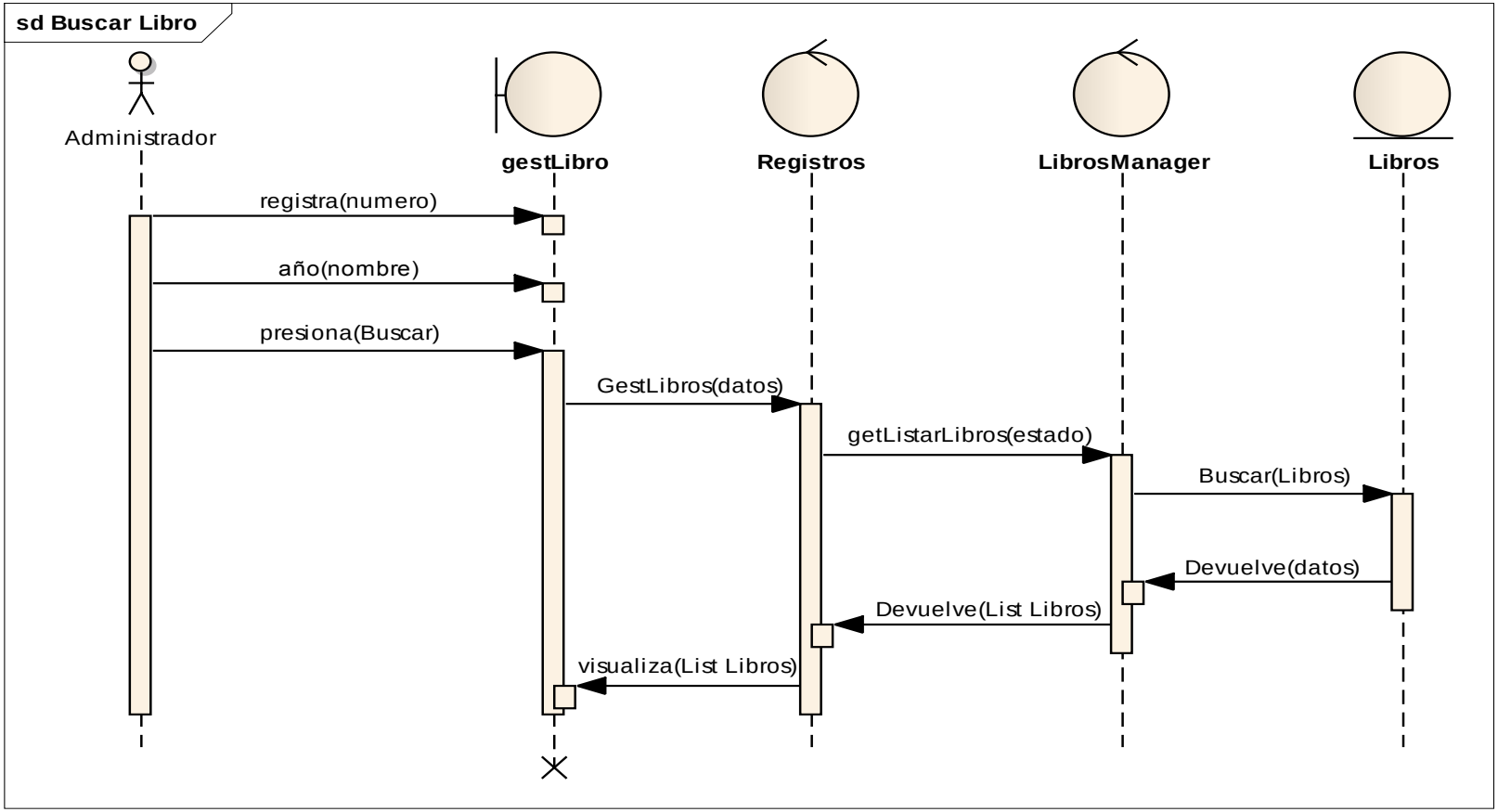


Figura Nº134. Diagrama de secuencia: CU Buscar Libro

2.1.2.2.3.10.2.1.21 Diagrama de Secuencia: Caso de Uso Ver Actas de Libro

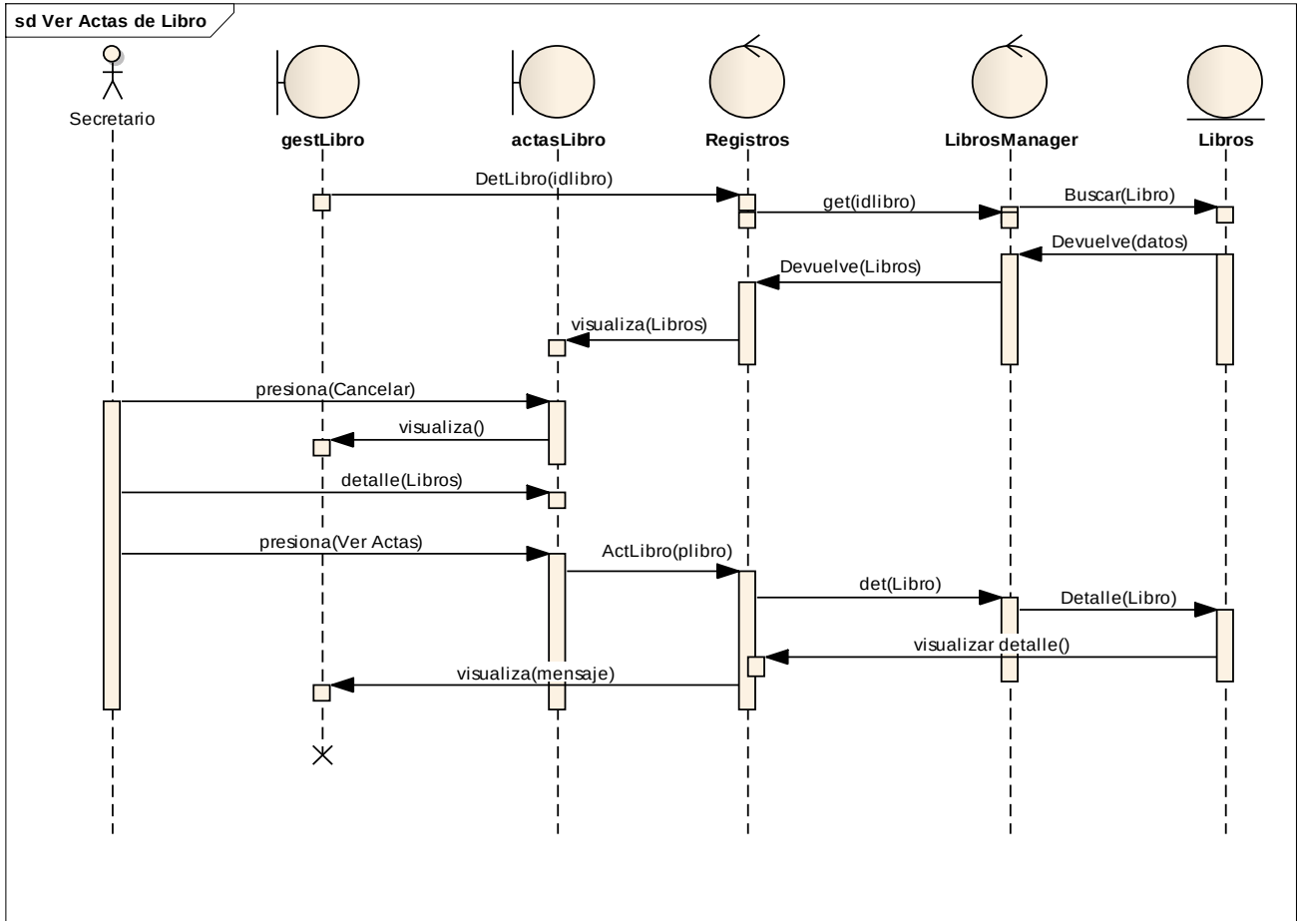


Figura Nº135. Diagrama de secuencia: CU Ver Actas de Libro

2.1.2.2.3.10.2.1.22 Diagrama de Secuencia: Caso de Uso Gestionar Bautismo

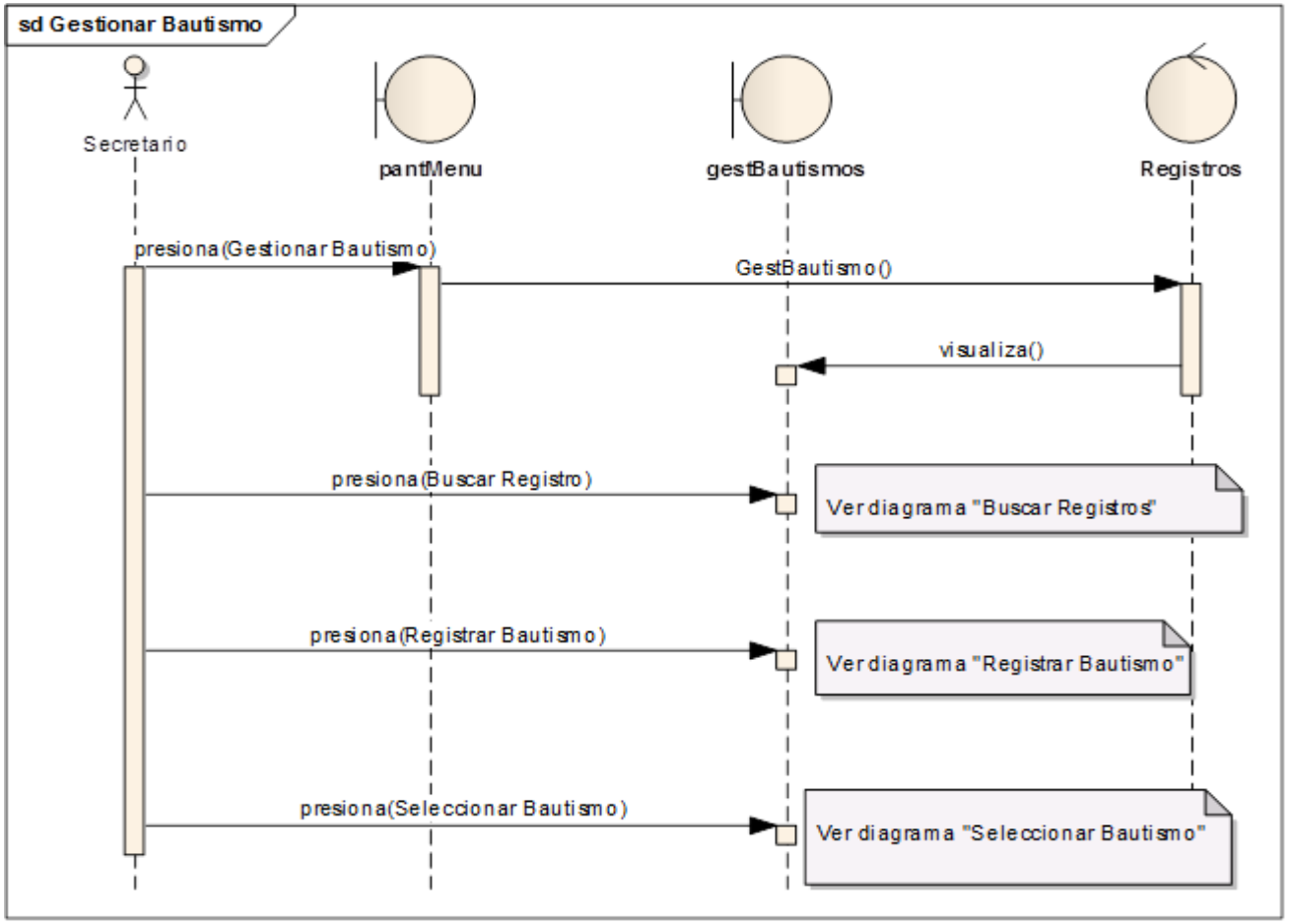


Figura Nº136. Diagrama de secuencia: CU Gestionar Bautismo

2.1.2.2.3.10.2.1.23 Diagrama de Secuencia: Caso de Uso Gestionar Reporte Bautismo

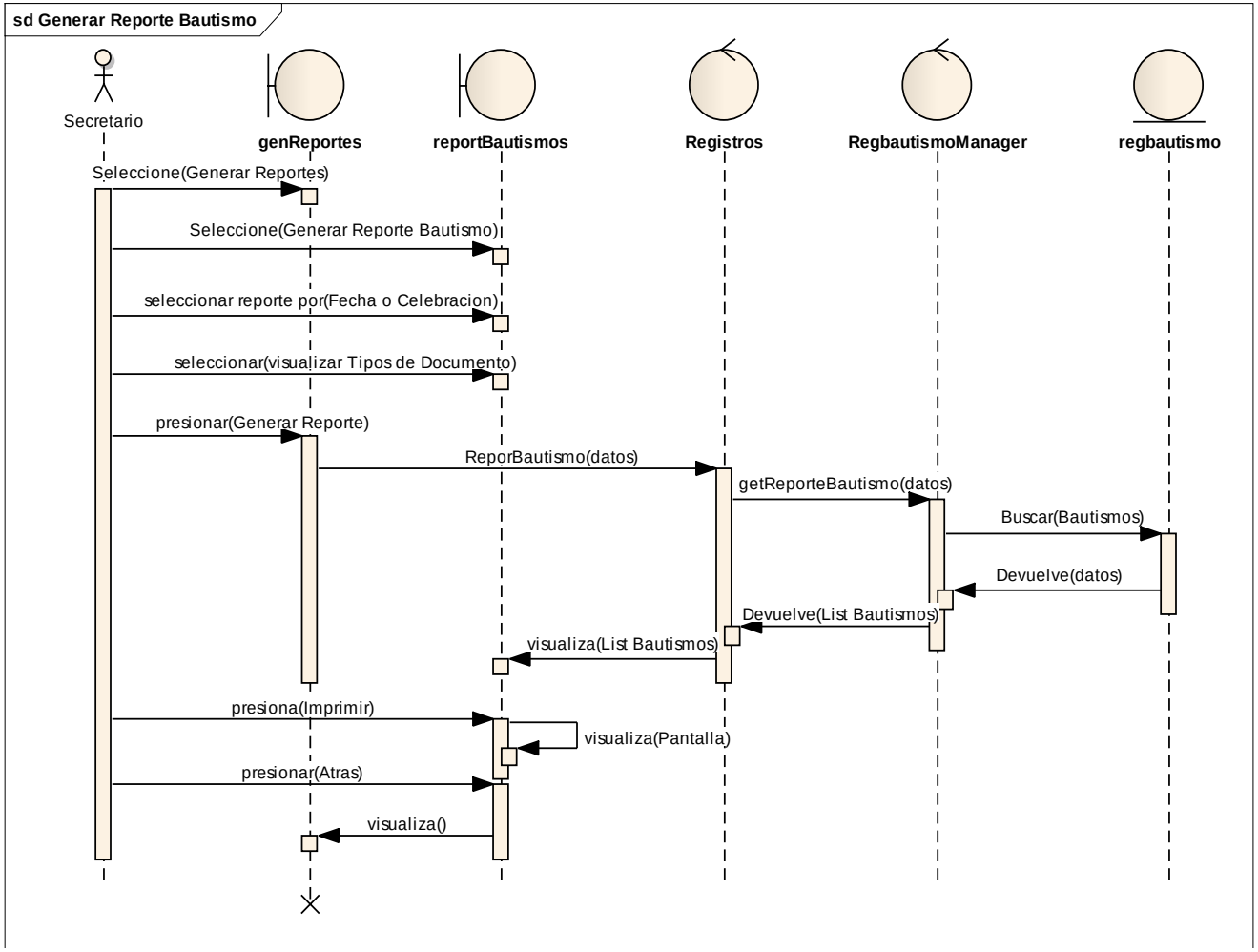


Figura Nº137. Diagrama de secuencia: CU Gestionar Reporte Bautismo

2.1.2.2.3.10.2.1.24 Diagrama de Secuencia: Caso de Uso Gestionar Comunión

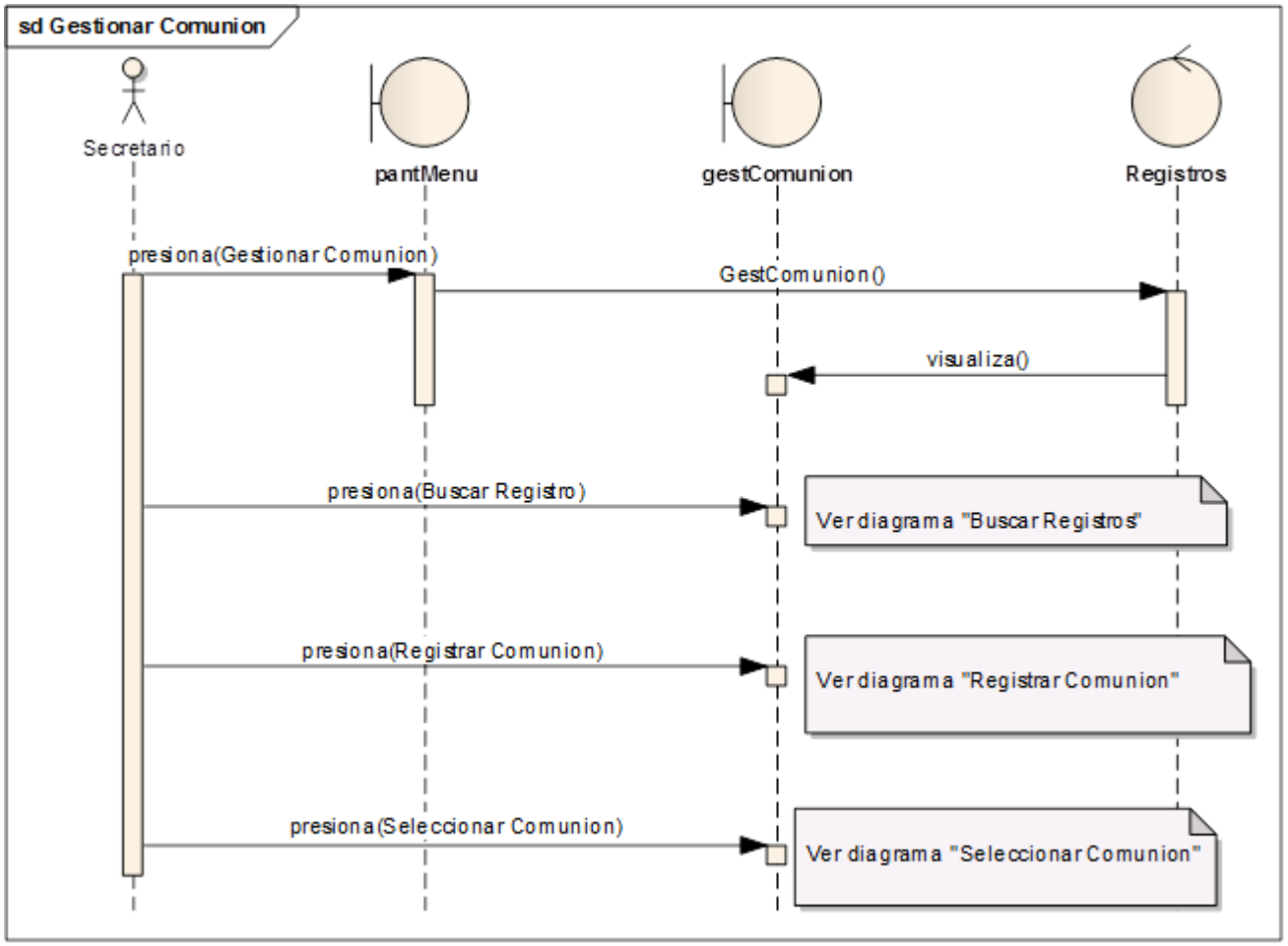


Figura Nº138. Diagrama de secuencia: CU Gestionar Comunión

2.1.2.2.3.10.2.1.25 Diagrama de Secuencia: Caso de Uso Generar Reporte Comunión

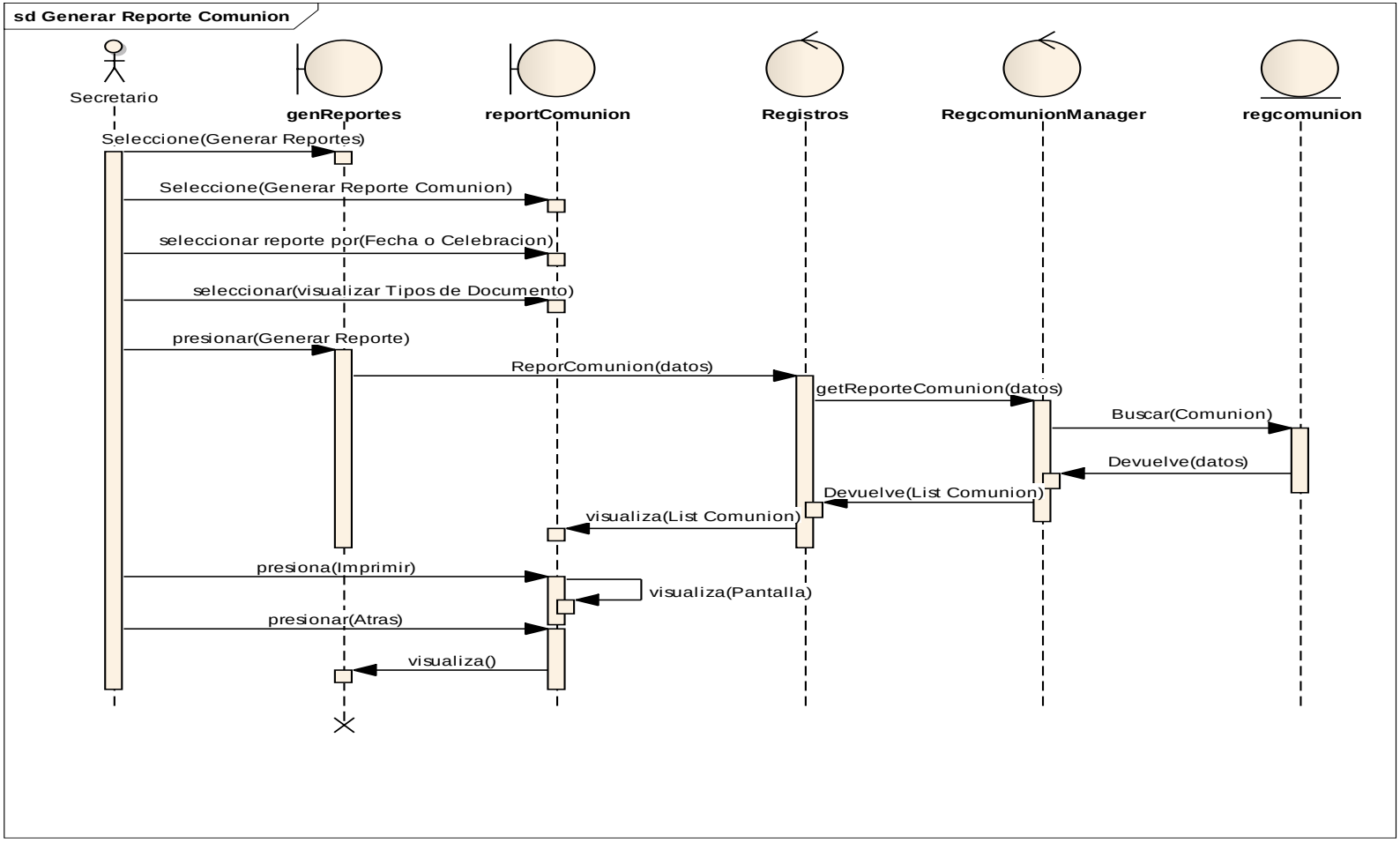


Figura Nº139. Diagrama de secuencia: CU Generar Reporte Comunión

Figura Nº140.

2.1.2.2.3.10.2.1.26 Diagrama de Secuencia: Caso de Uso Gestionar Confirmación

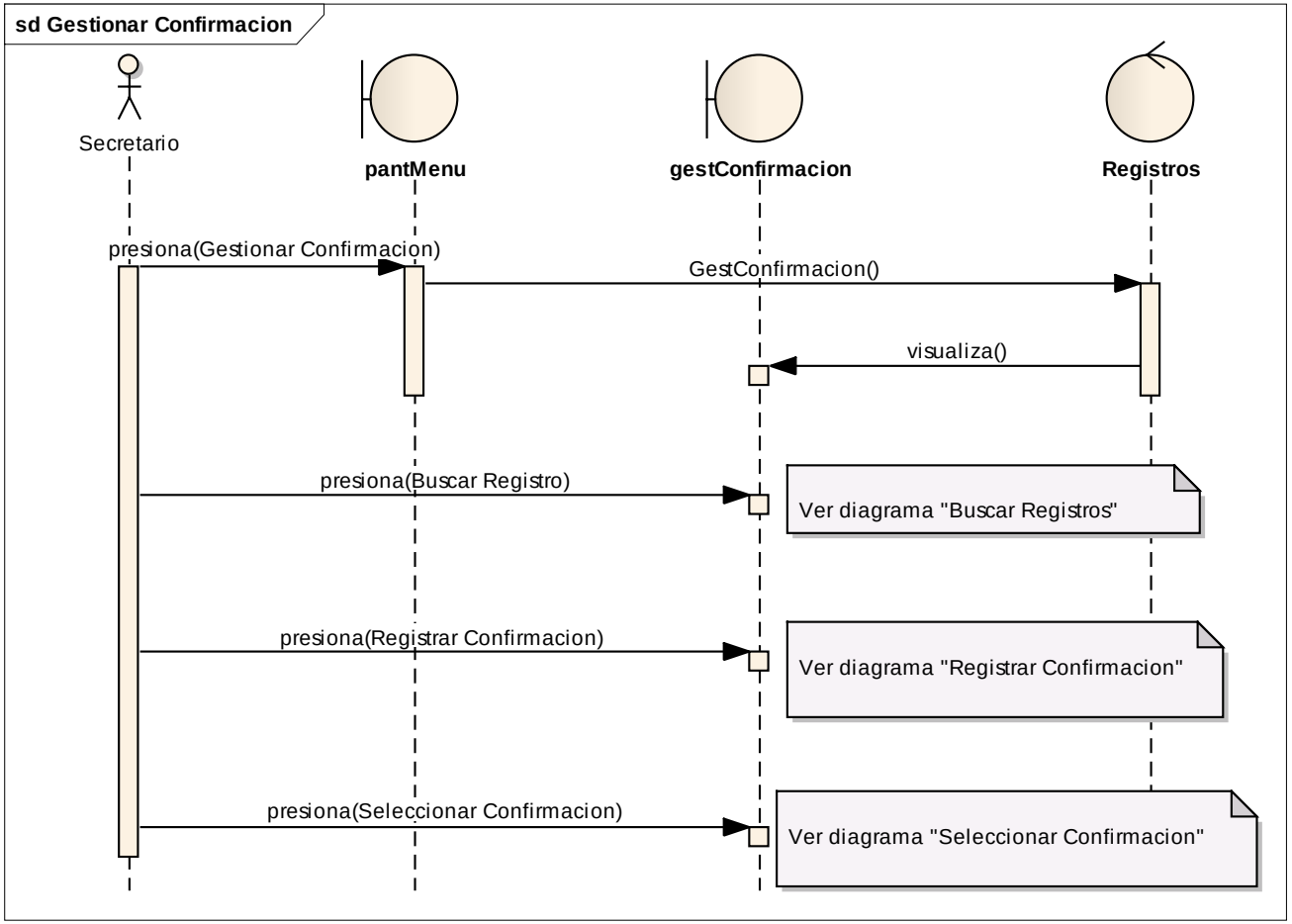


Figura Nº141. Diagrama de secuencia: CU Gestionar Confirmación

2.1.2.2.3.10.2.1.27 Diagrama de Secuencia: Caso de Uso Generar Reporte Confirmación

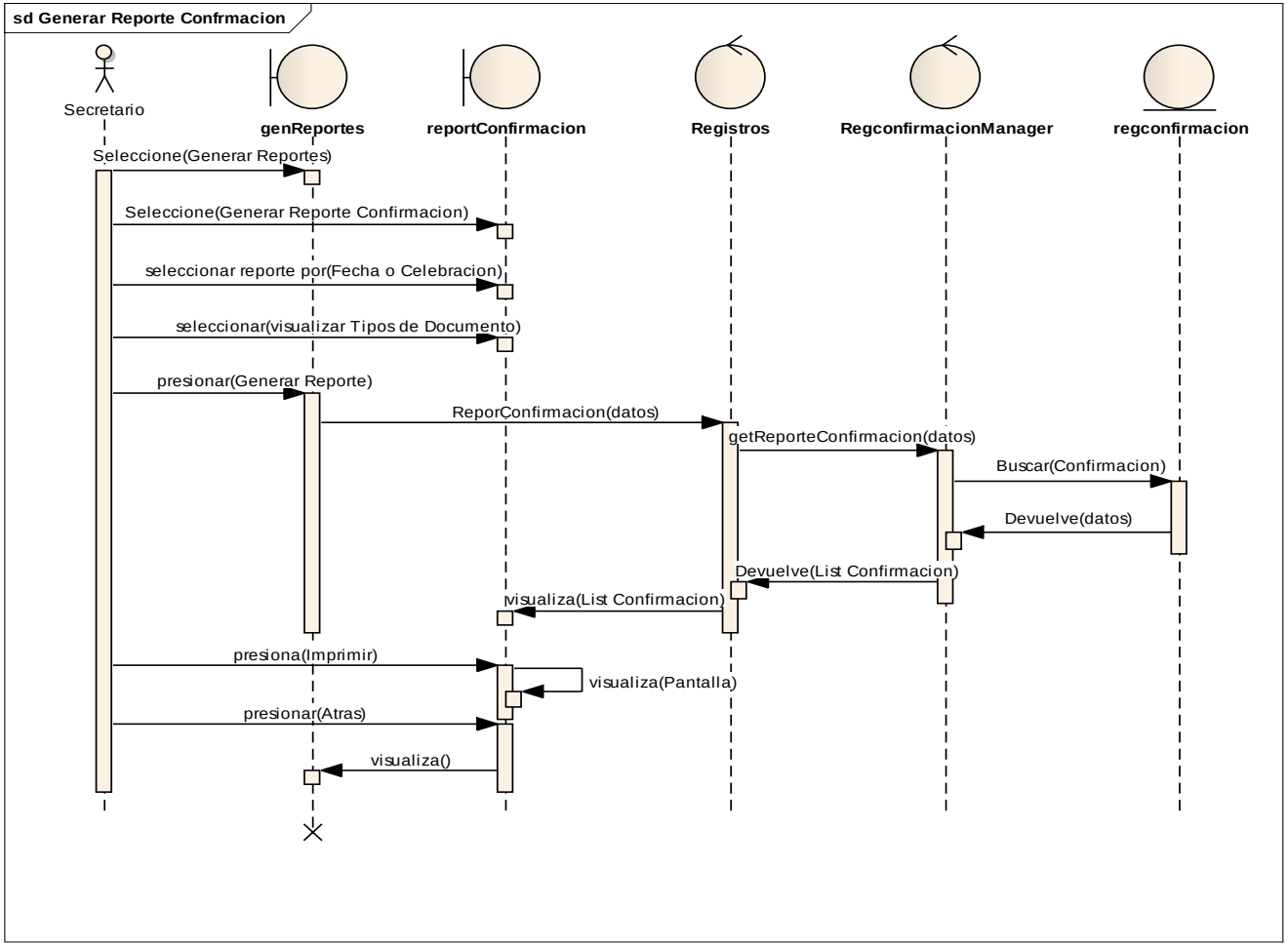


Figura Nº142. Diagrama de secuencia: CU Generar Reporte Confirmación

2.1.2.2.3.10.2.1.28 Diagrama de Secuencia: Caso de Uso Gestionar Matrimonio

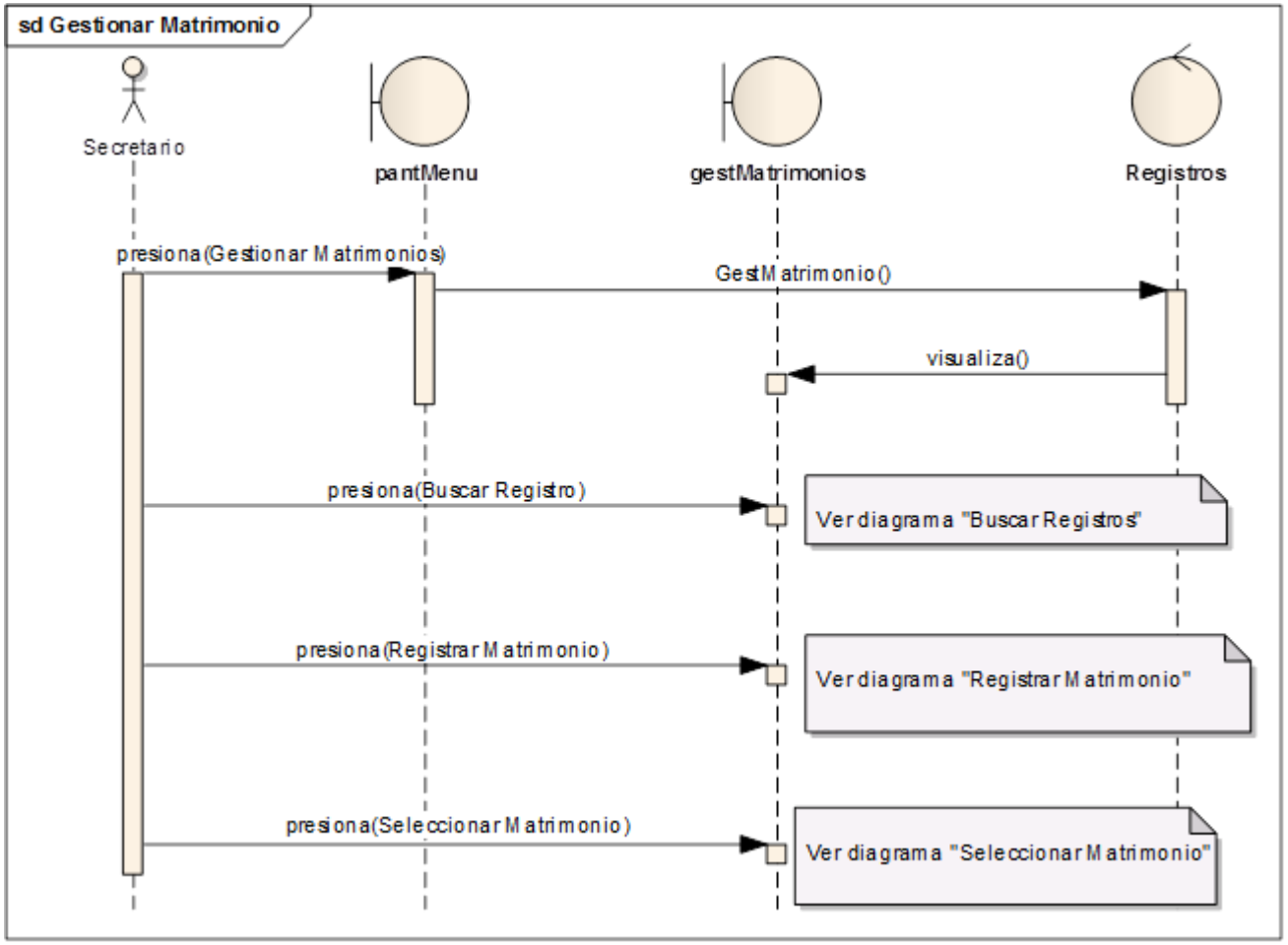


Figura Nº143. Diagrama de secuencia: CU Gestionar Matrimonio

2.1.2.2.3.10.2.1.29 Diagrama de Secuencia: Caso de Uso Generar Reporte Matrimonio

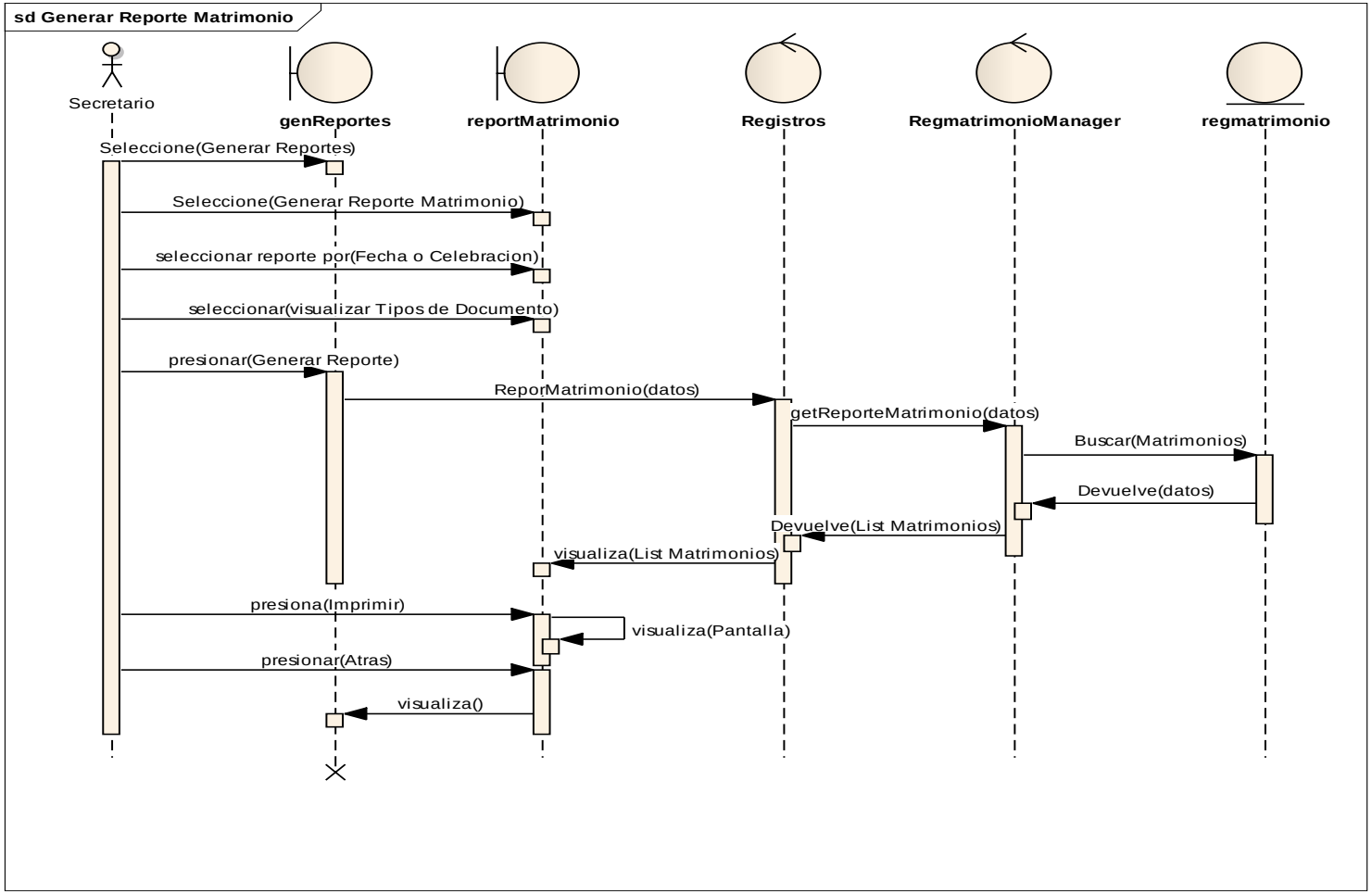


Figura Nº144. Diagrama de secuencia: CU Generar Reporte Matrimonio

2.1.2.2.3.11 Modelo de Datos

Previendo que la persistencia de la información del sistema será soportada por una base de datos relacional, este modelo describe la representación lógica de los datos persistentes, de acuerdo con el enfoque para modelado relacional de datos. Para expresar este modelo se utiliza un Diagrama de Clases (donde se utiliza un profile UML para Modelado de Datos, para conseguir la representación de tablas, claves, etc.).

2.1.2.2.3.11.1 Modelado de Diagrama de Clases

2.1.2.2.3.11.1.1 Introducción

El diagrama de clases es el diagrama principal para el análisis y diseño. Un diagrama de clases representa las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones.

2.1.2.2.3.11.1.2 Mecanismos de Abstracción

- Clasificación / Instanciación
- Composición / Descomposición
- Agrupación / Individualización
- Especialización / Generalización

La clasificación es una de los mecanismos de abstracción más utilizados. La clase define el ámbito de definición de un conjunto de objetos, y cada objeto pertenece a una clase. Los objetos se crean por instanciación de las clases.

Cada clase se representa en un rectángulo con tres compartimientos.

- Nombre de la clase
- Atributos de la clase
- Operaciones de la clase

Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos. Por esta razón se crearon niveles de visibilidad para los elementos que son:

- (-) Privado: es el más fuerte. Esta parte es totalmente invisible.
- (#) Los atributos u operaciones protegidas están visibles para las clases de herencia.
- (+) Los atributos u operaciones públicas son visibles desde otras clases y también por clases de herencia.

2.1.2.2.3.11.1.3 Diagrama de Clases

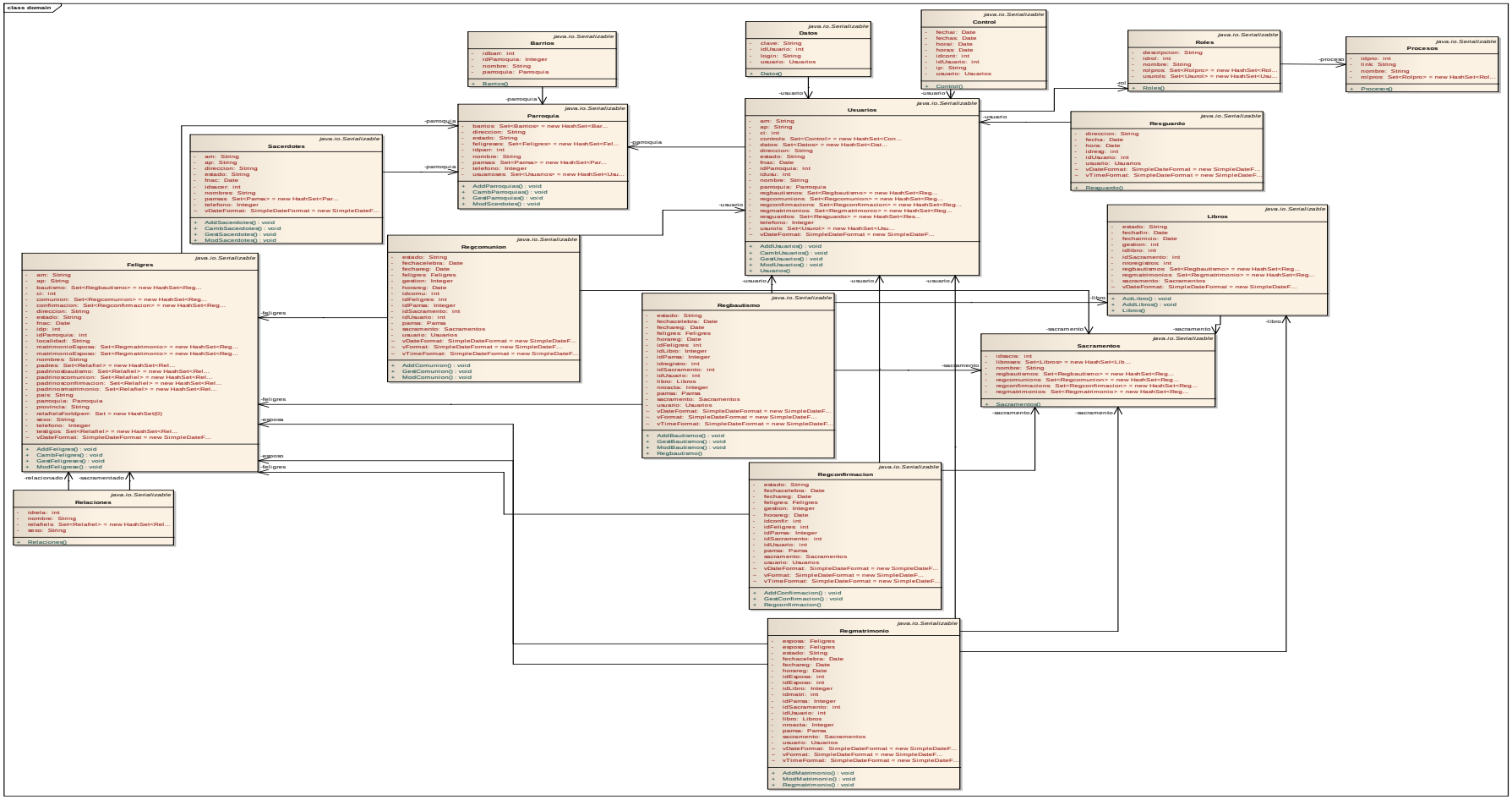


Figura Nº145. Diagrama de clases

2.1.2.2.3.11.2 Modelado Entidad Relación

2.1.2.2.3.11.2.1 Introducción

Cuando se utiliza una base de datos para gestionar información, se está plasmando una parte del mundo real en una serie de tablas, registros y campos ubicados en un ordenador; creándose un modelo parcial de la realidad. Antes de crear físicamente estas tablas en el ordenador se debe realizar un modelo de datos.

Se suele cometer el error de ir creando nuevas tablas a medida que se van necesitando, haciendo así el modelo de datos y la construcción física de las tablas simultáneamente. El resultado de esto acaba siendo un sistema de información parcheado, con datos dispersos que terminan por no cumplir adecuadamente los requisitos necesarios.

2.1.2.2.3.11.2.2 Modelo Entidad / Relación

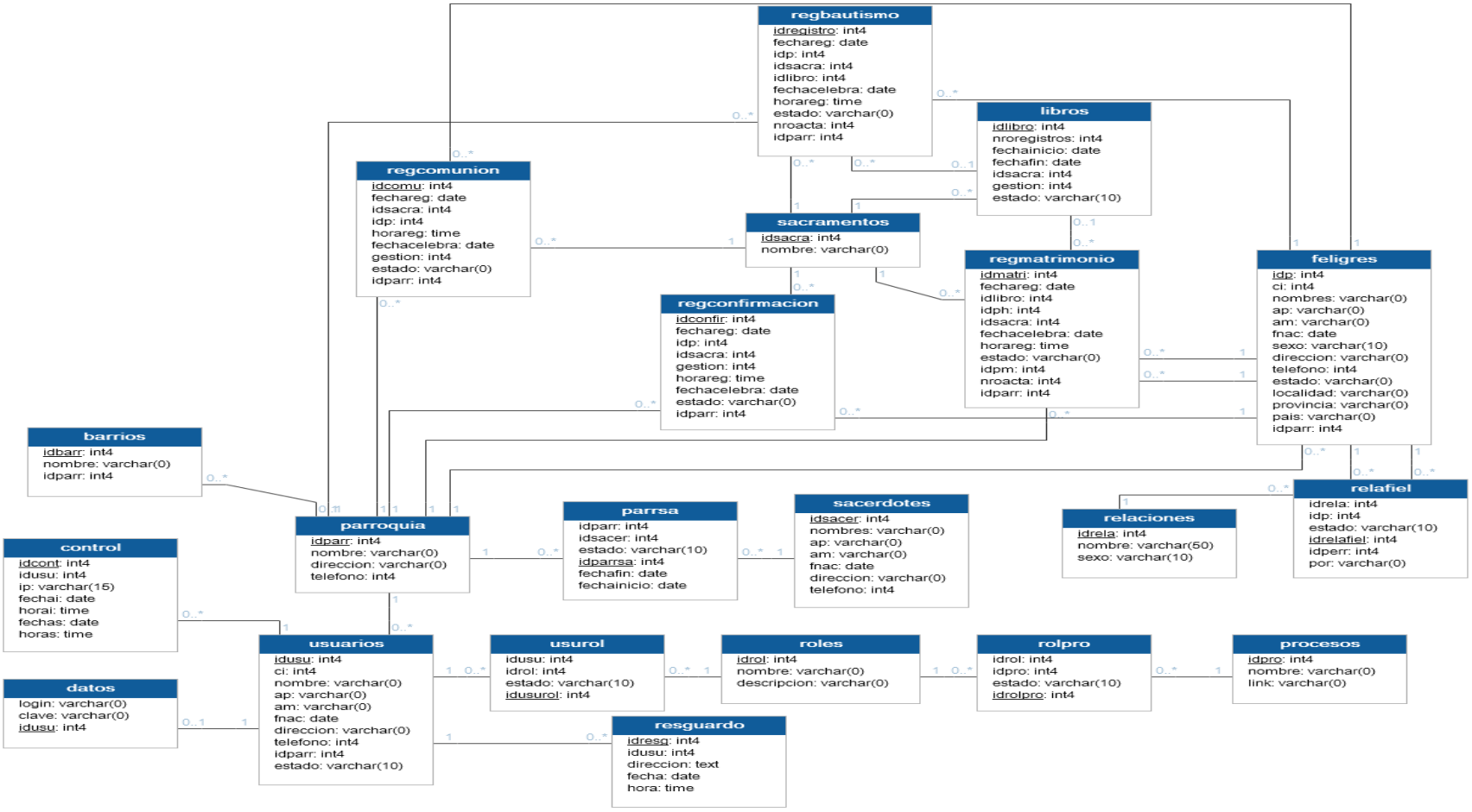


Figura Nº146. Modelo Entidad Relación

2.1.2.2.3.11.2.3 Creación de la Base de Datos (SQL)

```
Table: barrios
-- DROP TABLE barrios;
CREATE TABLE barrios (
  idbarr serial NOT NULL,
  nombre character varying,
  idparr integer,
  CONSTRAINT barrios_pkey PRIMARY KEY (idbarr),
  CONSTRAINT barrios_idparr_fkey FOREIGN KEY (idparr)
      REFERENCES parroquia (idparr) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE barrios
  OWNER TO postgres;
Table: control
-- DROP TABLE control;
CREATE TABLE control(
  idcont serial NOT NULL,
  idusu integer NOT NULL,
  ip character varying(15) NOT NULL,
  fechai date NOT NULL,
  horai time without time zone NOT NULL,
  fechas date,
  horas time without time zone,
  CONSTRAINT control_pkey PRIMARY KEY (idcont ),
  CONSTRAINT control_idusu_fkey FOREIGN KEY (idusu)
      REFERENCES usuarios (idusu) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (OIDS=FALSE);
ALTER TABLE control
```

```
  OWNER TO postgres;
Table: datos
-- DROP TABLE datos;
CREATE TABLE datos(
  login character varying,
  clave character varying,
  idusu integer NOT NULL,
  estado character varying,
  ci integer,
  CONSTRAINT datos_pkey PRIMARY KEY (idusu ),
  CONSTRAINT datos_idusu_fkey FOREIGN KEY (idusu)
      REFERENCES usuarios (idusu) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE datos
  OWNER TO postgres;
Table: feligrés
-- DROP TABLE feligrés;
CREATE TABLE feligrés(
  idp integer NOT NULL,
  ci integer NOT NULL,
  nombres character varying NOT NULL,
  ap character varying NOT NULL,
  am character varying,
  fnac date,
  sexo character varying(10) NOT NULL,
  direccion character varying,
  telefono integer,
  estado character varying NOT NULL,
  localidad character varying,
```

```
  provincia character varying,
  pais character varying,
  idparr integer NOT NULL,
  CONSTRAINT fieles_pkey PRIMARY KEY (idp ),
  CONSTRAINT fieles_idparr_fkey FOREIGN KEY (idparr)
      REFERENCES parroquia (idparr) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE feligrés
  OWNER TO postgres;
Table: libros
-- DROP TABLE libros;
CREATE TABLE libros(
  idlibro integer NOT NULL,
  nroregistros integer NOT NULL DEFAULT 0,
  fechainicio date NOT NULL,
  fechafin date,
  idsacra integer NOT NULL,
  gestion integer NOT NULL,
  estado character(10) NOT NULL DEFAULT 'Activo'::bpchar,
  CONSTRAINT libros_pkey PRIMARY KEY (idlibro ),
  CONSTRAINT libros_idsacra_fkey FOREIGN KEY (idsacra)
      REFERENCES sacramentos (idsacra) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE libros
  OWNER TO postgres;
Table: parroquia
-- DROP TABLE parroquia;
CREATE TABLE parroquia(
```

```
  idparr serial NOT NULL,
  nombre character varying NOT NULL,
  direccion character varying,
  telefono integer,
  CONSTRAINT parroquia_pkey PRIMARY KEY (idparr ))
WITH ( OIDS=FALSE);
ALTER TABLE parroquia
  OWNER TO postgres;
Table: parrsa
-- DROP TABLE parrsa;
CREATE TABLE parrsa(
  idparr integer NOT NULL,
  idsacer integer NOT NULL,
  estado character(10) NOT NULL DEFAULT 'Activo'::bpchar,
  idparrsa integer NOT NULL,
  fechafin date,
  fechainicio date NOT NULL,
  CONSTRAINT parrsa_pkey PRIMARY KEY (idparrsa ),
  CONSTRAINT parrsa_idparr_fkey FOREIGN KEY (idparr)
      REFERENCES parroquia (idparr) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT parrsa_idsacer_fkey FOREIGN KEY (idsacer)
      REFERENCES sacerdotes (idsacer) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE parrsa
  OWNER TO postgres;
Table: procesos
-- DROP TABLE procesos;
```

```
CREATE TABLE procesos(
  idpro serial NOT NULL,
  nombre character varying(100),
  link character varying(100),
  CONSTRAINT procesos_pkey PRIMARY KEY (idpro ))
WITH ( OIDS=FALSE);
ALTER TABLE procesos
  OWNER TO postgres;
Table: regbautismo
-- DROP TABLE regbautismo;
CREATE TABLE regbautismo(
  idregistro integer NOT NULL,
  fechareg date NOT NULL,
  idp integer NOT NULL,
  idsacra integer NOT NULL,
  idlibro integer,
  fechacelebra date,
  horareg time without time zone NOT NULL,
  estado character varying NOT NULL,
  nroacta integer,
  CONSTRAINT regbautismo_pkey PRIMARY KEY (idregistro ),
  CONSTRAINT regbautismo_idlibro_fkey FOREIGN KEY (idlibro)
     REFERENCES libros (idlibro) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regbautismo_idp_fkey FOREIGN KEY (idp)
     REFERENCES feligrés (idp) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regbautismo_idsacra_fkey FOREIGN KEY (idsacra)
     REFERENCES sacramentos (idsacra) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION)
```

```
WITH ( OIDS=FALSE);
ALTER TABLE regbautismo
  OWNER TO postgres;
Table: regcomunion
-- DROP TABLE regcomunion;
CREATE TABLE regcomunion(
  idcomu integer NOT NULL,
  fechareg date NOT NULL,
  idsacra integer NOT NULL,
  idp integer NOT NULL,
  horareg time without time zone NOT NULL,
  fechacelebra date,
  gestion integer,
  estado character varying NOT NULL,
  CONSTRAINT regcomunion_pkey PRIMARY KEY (idcomu ),
  CONSTRAINT regcomunion_idp_fkey FOREIGN KEY (idp)
     REFERENCES feligrés (idp) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regcomunion_idsacra_fkey FOREIGN KEY (idsacra)
     REFERENCES sacramentos (idsacra) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE regcomunion
  OWNER TO postgres;
Table: regconfirmacion
-- DROP TABLE regconfirmacion;
CREATE TABLE regconfirmacion(
  idconfir integer NOT NULL,
  fechareg date NOT NULL,
  idp integer NOT NULL,
```

```
  idsacra integer NOT NULL,
  gestion integer,
  horareg time without time zone NOT NULL,
  fechacelebra date,
  estado character varying NOT NULL,
  CONSTRAINT regconfirmacion_pkey PRIMARY KEY (idconfir ),
  CONSTRAINT regconfirmacion_idp_fkey FOREIGN KEY (idp)
     REFERENCES feligrés (idp) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regconfirmacion_idsacra_fkey FOREIGN KEY (idsacra)
     REFERENCES sacramentos (idsacra) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE regconfirmacion
  OWNER TO postgres;
Table: regmatrimonio
-- DROP TABLE regmatrimonio;
CREATE TABLE regmatrimonio(
  idmatri integer NOT NULL,
  fechareg date NOT NULL,
  idlibro integer,
  idph integer NOT NULL,
  idsacra integer NOT NULL,
  fechacelebra date,
  horareg time without time zone NOT NULL,
  estado character varying NOT NULL,
  idpm integer NOT NULL,
  nroacta integer,
  CONSTRAINT regmatrimonio_pkey PRIMARY KEY (idmatri ),
  CONSTRAINT regmatrimonio_idlibro_fkey FOREIGN KEY (idlibro)
```

```
    REFERENCES libros (idlibro) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regmatrimonio_idph_fkey FOREIGN KEY (idph)
    REFERENCES feligrés (idp) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regmatrimonio_idpm_fkey FOREIGN KEY (idpm)
    REFERENCES feligrés (idp) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT regmatrimonio_idsacra_fkey FOREIGN KEY (idsacra)
    REFERENCES sacramentos (idsacra) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE regmatrimonio
  OWNER TO postgres;
Table: relaciones
-- DROP TABLE relaciones;
CREATE TABLE relaciones(
  idrela serial NOT NULL,
  nombre character varying(50) NOT NULL,
  sexo character varying(10),
  CONSTRAINT relaciones_pkey PRIMARY KEY (idrela ))
WITH ( OIDS=FALSE);
ALTER TABLE relaciones
  OWNER TO postgres;
Table: relafiel
-- DROP TABLE relafiel;
CREATE TABLE relafiel(
  idrela integer NOT NULL,
  idp integer NOT NULL,
  estado character(10) NOT NULL DEFAULT 'Activo'::bpchar,
```

```
  idrelafiel integer NOT NULL,
  idperr integer NOT NULL,
  por character varying NOT NULL,
  CONSTRAINT relafiel_pkey PRIMARY KEY (idrelafiel ),
  CONSTRAINT relafiel_idp_fkey FOREIGN KEY (idp)
      REFERENCES feligrés (idp) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT relafiel_idperr_fkey FOREIGN KEY (idperr)
      REFERENCES feligrés (idp) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT relafiel_idrela_fkey FOREIGN KEY (idrela)
      REFERENCES relaciones (idrela) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE relafiel
  OWNER TO postgres;
Table: roles
-- DROP TABLE roles;
CREATE TABLE roles(
  idrol serial NOT NULL,
  nombre character varying,
  descripcion character varying(100),
  CONSTRAINT roles_pkey PRIMARY KEY (idrol ))
WITH ( OIDS=FALSE);
ALTER TABLE roles
  OWNER TO postgres;
Table: rolpro
-- DROP TABLE rolpro;
CREATE TABLE rolpro(
  idrol integer NOT NULL,
```

```
  idpro integer NOT NULL,
  estado boolean,
  idrolpro integer NOT NULL,
  CONSTRAINT rolpro_pkey PRIMARY KEY (idrolpro ),
  CONSTRAINT rolpro_idpro_fkey FOREIGN KEY (idpro)
      REFERENCES procesos (idpro) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT rolpro_idrol_fkey FOREIGN KEY (idrol)
      REFERENCES roles (idrol) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE rolpro
  OWNER TO postgres;
Table: sacerdotes
-- DROP TABLE sacerdotes;
CREATE TABLE sacerdotes(
  idsacer serial NOT NULL,
  nombres character varying(100),
  ap character varying(100),
  am character varying(100),
  fnac date, --
  direccion character varying,
  telefono integer,
  CONSTRAINT sacerdotes_pkey PRIMARY KEY (idsacer ))
WITH (  OIDS=FALSE);
ALTER TABLE sacerdotes
  OWNER TO postgres;
COMMENT ON COLUMN sacerdotes.fnac IS ';
Table: sacramentos
-- DROP TABLE sacramentos;
```

```
CREATE TABLE sacramentos(
  idsacra serial NOT NULL,
  nombre character varying(100),
  CONSTRAINT sacramentos_pkey PRIMARY KEY (idsacra ))
WITH (  OIDS=FALSE);
ALTER TABLE sacramentos
  OWNER TO postgres;
Table: usuarios
-- DROP TABLE usuarios;
CREATE TABLE usuarios(
  idusu serial NOT NULL,
  ci integer NOT NULL,
  nombre character varying NOT NULL,
  ap character varying NOT NULL,
  am character varying,
  fnac date,
  direccion character varying,
  telefono integer,
  idparr integer NOT NULL,
  estado character(10) NOT NULL DEFAULT 'Activo'::bpchar,
  CONSTRAINT usuarios_pkey PRIMARY KEY (idusu ),
  CONSTRAINT usuarios_idparr_fkey FOREIGN KEY (idparr)
      REFERENCES parroquia (idparr) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (OIDS=FALSE);
ALTER TABLE usuarios
  OWNER TO postgres;
Table: usurol
-- DROP TABLE usurol;
CREATE TABLE usurol(
```

```
  idusu integer NOT NULL,
  idrol integer NOT NULL,
  estado boolean,
  idusurol integer NOT NULL,
  CONSTRAINT usurol_pkey PRIMARY KEY (idusurol ),
  CONSTRAINT usurol_idrol_fkey FOREIGN KEY (idrol)
     REFERENCES roles (idrol) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT usurol_idusu_fkey FOREIGN KEY (idusu)
     REFERENCES usuarios (idusu) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH (  OIDS=FALSE);
ALTER TABLE usurol
  OWNER TO postgres;
```

2.1.2.2.3.11.2.4 Diccionario de Datos

A continuación se describe la creación de la base de datos del software, apuntando su característica orientada a objetos. Para expresar este modelo se utiliza un Diagrama de Clases de Objetos y luego su correspondiente implementación en el lenguaje utilizado para el desarrollo del proyecto el cual es JAVA.

```
public class Barrios implements java.io.Serializable {
        private int idbarr;
        private Parroquia parroquia;
        private String nombre;
        public Barrios() {}
        public Barrios(int idbarr) {
                this.idbarr = idbarr;}
        public Barrios(int idbarr, Parroquia parroquia, String nombre) {
                this.idbarr = idbarr;
                this.parroquia = parroquia;
                this.nombre = nombre;}
        @Id
        @Column(name = "idbarr", unique = true, nullable = false)
        public int getIdbarr() {
                return this.idbarr;}
        public void setIdbarr(int idbarr) {
                this.idbarr = idbarr;}
        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "idparr")
        public Parroquia getParroquia() {
                return this.parroquia;}
        public void setParroquia(Parroquia parroquia) {
                this.parroquia = parroquia;}
        @Column(name = "nombre")
```

```
	public String getNombre() {
		return this.nombre;}
	public void setNombre(String nombre) {
		this.nombre = nombre;}
}
public class Datos implements java.io.Serializable {
	private int idusu;
	private Usuarios usuarios;
	private String login;
	private String clave;
	private String estado;
	private Integer ci;
	public Datos() {}
	public Datos(int idusu, Usuarios usuarios) {
		this.idusu = idusu;
		this.usuarios = usuarios;
	}
	public Datos(int idusu, Usuarios usuarios, String login, String clave,
		String estado, Integer ci) {
		this.idusu = idusu;
		this.usuarios = usuarios;
		this.login = login;
		this.clave = clave;
		this.estado = estado;
		this.ci = ci;}
	@Id
	@Column(name = "idusu", unique = true, nullable = false)
	public int getIdusu() {
		return this.idusu;}
	public void setIdusu(int idusu) {
```

```
		this.idusu = idusu;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idusu", unique = true, nullable = false, insertable = false, updatable = false)
	public Usuarios getUsuarios() {
		return this.usuarios;}
	public void setUsuarios(Usuarios usuarios) {
		this.usuarios = usuarios;}
	@Column(name = "login")
	public String getLogin() {
		return this.login;}
	public void setLogin(String login) {
		this.login = login;}
	@Column(name = "clave")
	public String getClave() {
		return this.clave;}
	public void setClave(String clave) {
		this.clave = clave;}
	@Column(name = "estado")
	public String getEstado() {
		return this.estado;}
	public void setEstado(String estado) {
		this.estado = estado;}
	@Column(name = "ci")
	public Integer getCi() {
		return this.ci;}
	public void setCi(Integer ci) {
		this.ci = ci;}
}
```

```
public class Fieles implements java.io.Serializable {
        private int idp;
        private Parroquia parroquia;
        private Integer ci;
        private String nombres;
        private String ap;
        private String am;
        private Date fnac;
        private String sexo;
        private String direccion;
        private Integer telefono;
        private String estado;
        private String localidad;
        private String provincia;
        private String pais;
        private Set<Regmatrimonio> regmatrimonios = new
HashSet<Regmatrimonio>(0);
        private Set<Relafiel> relafielsForIdperr = new HashSet<Relafiel>(0);
        private Set<Regbautismo> regbautismos = new HashSet<Regbautismo>(0);
        private Set<Relafiel> relafielsForIdp = new HashSet<Relafiel>(0);
        private Set<Inscripciones> inscripcioneses = new HashSet<Inscripciones>(0);
        private Set<Regconfirmacion> regconfirmacions = new
HashSet<Regconfirmacion>(0);
        private Set<Regcomunion> regcomunions = new HashSet<Regcomunion>(0);
        //Variables incrementadas
        SimpleDateFormat vDateFormat = new SimpleDateFormat("dd-MM-yyyy");
        public Fieles() {}
        public Fieles(int idp) {
                this.idp = idp;}
        public Fieles(int idp, Parroquia parroquia, Integer ci, String nombres,
```

```
            String ap, String am, Date fnac, String sexo, String direccion,
            Integer telefono, String estado, String localidad,
            String provincia, String pais, Integer idpapa, Integer idmama,
            Set regmatrimonios, Set regbautismos, Set relafiels, Set relafielsperr,
            Set inscripcioneses, Set regconfirmacions, Set regcomunions) {
            this.idp = idp;
            this.parroquia = parroquia;
            this.ci = ci;
            this.nombres = nombres;
            this.ap = ap;
            this.am = am;
            this.fnac = fnac;
            this.sexo = sexo;
            this.direccion = direccion;
            this.telefono = telefono;
            this.estado = estado;
            this.localidad = localidad;
            this.provincia = provincia;
            this.pais = pais;
            this.regmatrimonios = regmatrimonios;
            this.regbautismos = regbautismos;
            this.inscripcioneses = inscripcioneses;
            this.regconfirmacions = regconfirmacions;
            this.regcomunions = regcomunions;
    }
    @Id
    @Column(name = "idp", unique = true, nullable = false)
    public int getIdp() {
            return this.idp;}
    public void setIdp(int idp) {
```

```
        this.idp = idp;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idparr")
public Parroquia getParroquia() {
        return this.parroquia;}
public void setParroquia(Parroquia parroquia) {
        this.parroquia = parroquia;}
@Column(name = "ci")
public Integer getCi() {
        return this.ci;}
public void setCi(Integer ci) {
        this.ci = ci;}
@Column(name = "nombres", length = 100)
public String getNombres() {
return this.nombres;}
public void setNombres(String nombres) {
        this.nombres = nombres;}
@Column(name = "ap", length = 100)
public String getAp() {
        return this.ap;}
public void setAp(String ap) {
        this.ap = ap;}
@Column(name = "am", length = 100)
public String getAm() {
        return this.am;}
public void setAm(String am) {
        this.am = am;}
@Temporal(TemporalType.DATE)
@Column(name = "fnac", length = 13)
public Date getFnac() {
```

```
        return this.fnac;}
public void setFnac(Date fnac) {
        this.fnac = fnac;}
@Column(name = "sexo", length = 50)
public String getSexo() {
        return this.sexo;}
public void setSexo(String sexo) {
        this.sexo = sexo;}
@Column(name = "direccion", length = 100)
public String getDireccion() {
        return this.direccion;}
public void setDireccion(String direccion) {
        this.direccion = direccion;}
@Column(name = "telefono")
public Integer getTelefono() {
        return this.telefono;}
public void setTelefono(Integer telefono) {
        this.telefono = telefono;}
@Column(name = "estado")
public String getEstado() {
        return this.estado;}
public void setEstado(String estado) {
        this.estado = estado;}
@Column(name = "localidad", length = 100)
public String getLocalidad() {
        return this.localidad;}
public void setLocalidad(String localidad) {
        this.localidad = localidad;}
@Column(name = "provincia", length = 100)
public String getProvincia() {
```

```
            return this.provincia;}
        public void setProvincia(String provincia) {
            this.provincia = provincia;}
        @Column(name = "pais", length = 100)
        public String getPais() {
            return this.pais;}
        public void setPais(String pais) {
            this.pais = pais;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "fieles")
        public Set<Regmatrimonio> getRegmatrimonios() {
            return this.regmatrimonios;}
        public void setRegmatrimonios(Set<Regmatrimonio> regmatrimonios) {
            this.regmatrimonios = regmatrimonios;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "fielesByIdperr")
        public Set<Relafiel> getRelafielsForIdperr() {
            return this.relafielsForIdperr;}
        public void setRelafielsForIdperr(Set<Relafiel> relafielsForIdperr) {
            this.relafielsForIdperr = relafielsForIdperr;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "fieles")
        public Set<Regbautismo> getRegbautismos() {
            return this.regbautismos;}
        public void setRegbautismos(Set<Regbautismo> regbautismos) {
            this.regbautismos = regbautismos;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "fielesByIdp")
        public Set<Relafiel> getRelafielsForIdp() {
            return this.relafielsForIdp;}
```

```
	public void setRelafielsForIdp(Set<Relafiel> relafielsForIdp) {
		this.relafielsForIdp = relafielsForIdp;}
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "fieles")
	public Set<Inscripciones> getInscripcioneses() {
		return this.inscripcioneses;}
	public void setInscripcioneses(Set<Inscripciones> inscripcioneses) {
		this.inscripcioneses = inscripcioneses;}
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "fieles")
	public Set<Regconfirmacion> getRegconfirmacions() {
		return this.regconfirmacions;}
	public void setRegconfirmacions(Set<Regconfirmacion> regconfirmacions) {
		this.regconfirmacions = regconfirmacions;}
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "fieles")
	public Set<Regcomunion> getRegcomunions() {
		return this.regcomunions;}
	public void setRegcomunions(Set<Regcomunion> regcomunions) {
		this.regcomunions = regcomunions;}
		//Metodos incrementados
	public String getFechaNac() {
		if(this.fnac != null)
			return vDateFormat.format(this.fnac);
		else
			return "";}
}
public class Libros implements java.io.Serializable {
	private int idlibro;
	private Sacramentos sacramentos;
```

```
private int nlibro;
private Integer nregistroinicial;
private Integer nregistrofinal;
private Integer cantidadregistros;
private Date fechainicio;
private Date fechafin;
private String ano;
private Set regcomunions = new HashSet(0);
private Set regbautismos = new HashSet(0);
private Set regmatrimonios = new HashSet(0);
private Set regconfirmacions = new HashSet(0);
public Libros() {}
public Libros(int idlibro, int nlibro) {
        this.idlibro = idlibro;
        this.nlibro = nlibro;}
public Libros(int idlibro, Sacramentos sacramentos, int nlibro,
        Integer nregistroinicial, Integer nregistrofinal,
        Integer cantidadregistros, Date fechainicio, Date fechafin,
        String ano, Set regcomunions, Set regbautismos, Set regmatrimonios,
        Set regconfirmacions) {
        this.idlibro = idlibro;
        this.sacramentos = sacramentos;
        this.nlibro = nlibro;
        this.nregistroinicial = nregistroinicial;
        this.nregistrofinal = nregistrofinal;
        this.cantidadregistros = cantidadregistros;
        this.fechainicio = fechainicio;
        this.fechafin = fechafin;
        this.ano = ano;
        this.regcomunions = regcomunions;
```

```
        this.regbautismos = regbautismos;
        this.regmatrimonios = regmatrimonios;
        this.regconfirmacions = regconfirmacions;
}
@Id
@Column(name = "idlibro", unique = true, nullable = false)
public int getIdlibro() {
        return this.idlibro;}
public void setIdlibro(int idlibro) {
        this.idlibro = idlibro;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idsacra")
public Sacramentos getSacramentos() {
        return this.sacramentos;}
public void setSacramentos(Sacramentos sacramentos) {
        this.sacramentos = sacramentos;}
@Column(name = "nlibro", nullable = false)
public int getNlibro() {
        return this.nlibro;}
public void setNlibro(int nlibro) {
        this.nlibro = nlibro;}
@Column(name = "nregistroinicial")
public Integer getNregistroinicial() {
        return this.nregistroinicial;}
public void setNregistroinicial(Integer nregistroinicial) {
        this.nregistroinicial = nregistroinicial;}
@Column(name = "nregistrofinal")
public Integer getNregistrofinal() {
        return this.nregistrofinal;}
public void setNregistrofinal(Integer nregistrofinal) {
```

```
            this.nregistrofinal = nregistrofinal;}
        @Column(name = "cantidadregistros")
        public Integer getCantidadregistros() {
            return this.cantidadregistros;}
        public void setCantidadregistros(Integer cantidadregistros) {
            this.cantidadregistros = cantidadregistros;}
        @Temporal(TemporalType.DATE)
        @Column(name = "fechainicio", length = 13)
        public Date getFechainicio() {
            return this.fechainicio;}
        public void setFechainicio(Date fechainicio) {
            this.fechainicio = fechainicio;}
        @Temporal(TemporalType.DATE)
        @Column(name = "fechafin", length = 13)
        public Date getFechafin() {
            return this.fechafin;}
        public void setFechafin(Date fechafin) {
            this.fechafin = fechafin;}
        @Column(name = "ano")
        public String getAno() {
            return this.ano;}
        public void setAno(String ano) {
            this.ano = ano;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "libros")
        public Set getRegcomunions() {
            return this.regcomunions;}
        public void setRegcomunions(Set regcomunions) {
            this.regcomunions = regcomunions;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
```

```
mappedBy = "libros")
	public Set getRegbautismos() {
		return this.regbautismos;}
	public void setRegbautismos(Set regbautismos) {
		this.regbautismos = regbautismos;}
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "libros")
	public Set getRegmatrimonios() {
		return this.regmatrimonios;}
	public void setRegmatrimonios(Set regmatrimonios) {
		this.regmatrimonios = regmatrimonios;}
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "libros")
	public Set getRegconfirmacions() {
		return this.regconfirmacions;}
	public void setRegconfirmacions(Set regconfirmacions) {
		this.regconfirmacions = regconfirmacions;}
}
public class Parroquia implements java.io.Serializable {
	private int idparr;
	private String nombre;
	private String direccion;
	private Integer telefono;
	private Set<Parrsa> parrsas = new HashSet(0);
	private Set usuarioses = new HashSet(0);
	private Set barrioses = new HashSet(0);
	private Set fieleses = new HashSet(0);
	public Parroquia() {}
	public Parroquia(int idparr) {
		this.idparr = idparr;}
```

```
public Parroquia(int idparr, String nombre, String direccion,
        Integer telefono, Set parrsas, Set usuarioses, Set barrioses,
        Set fieleses) {
        this.idparr = idparr;
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
        this.parrsas = parrsas;
        this.usuarioses = usuarioses;
        this.barrioses = barrioses;
        this.fieleses = fieleses;
}
@Id
@Column(name = "idparr", unique = true, nullable = false)
public int getIdparr() {
        return this.idparr;}
public void setIdparr(int idparr) {
        this.idparr = idparr;}
@Column(name = "nombre", length = 100)
public String getNombre() {
        return this.nombre;}
public void setNombre(String nombre) {
        this.nombre = nombre;}
@Column(name = "direccion")
public String getDireccion() {
        return this.direccion;}
public void setDireccion(String direccion) {
        this.direccion = direccion;}
@Column(name = "telefono")
public Integer getTelefono() {
```

```
        return this.telefono;}
    public void setTelefono(Integer telefono) {
        this.telefono = telefono;}
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "parroquia")
    public Set getParrsas() {
        return this.parrsas;}
    public void setParrsas(Set parrsas) {
        this.parrsas = parrsas;}
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "parroquia")
    public Set getUsuarioses() {
        return this.usuarioses;}
    public void setUsuarioses(Set usuarioses) {
        this.usuarioses = usuarioses;
    }
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "parroquia")
    public Set getBarrioses() {
        return this.barrioses;}
    public void setBarrioses(Set barrioses) {
        this.barrioses = barrioses;}
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "parroquia")
    public Set getFieleses() {
        return this.fieleses;}
    public void setFieleses(Set fieleses) {
        this.fieleses = fieleses;}
}
```

```
public class Parrsa implements java.io.Serializable {
	private int idparrsa;
	private Sacerdotes sacerdotes;
	private Parroquia parroquia;
	private Boolean estado;
	public Parrsa() {}
	public Parrsa(int idparrsa, Sacerdotes sacerdotes, Parroquia parroquia) {
		this.idparrsa = idparrsa;
		this.sacerdotes = sacerdotes;
		this.parroquia = parroquia;}
	public Parrsa(int idparrsa, Sacerdotes sacerdotes, Parroquia parroquia,
			Boolean estado) {
		this.idparrsa = idparrsa;
		this.sacerdotes = sacerdotes;
		this.parroquia = parroquia;
		this.estado = estado;}
	@Id
	@Column(name = "idparrsa", unique = true, nullable = false)
	public int getIdparrsa() {
		return this.idparrsa;}
	public void setIdparrsa(int idparrsa) {
		this.idparrsa = idparrsa;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idsacer", nullable = false)
	public Sacerdotes getSacerdotes() {
		return this.sacerdotes;}
	public void setSacerdotes(Sacerdotes sacerdotes) {
		this.sacerdotes = sacerdotes;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idparr", nullable = false)
```

```
	public Parroquia getParroquia() {
		return this.parroquia;}
	public void setParroquia(Parroquia parroquia) {
		this.parroquia = parroquia;}
	@Column(name = "estado")
	public Boolean getEstado() {
		return this.estado;}
	public void setEstado(Boolean estado) {
		this.estado = estado;}
}
public class Procesos implements java.io.Serializable {
	private int idpro;
	private String nombre;
	private String link;
	private Set rolpros = new HashSet(0);
	public Procesos() {}
	public Procesos(int idpro) {
		this.idpro = idpro;}
	public Procesos(int idpro, String nombre, String link, Set rolpros) {
		this.idpro = idpro;
		this.nombre = nombre;
		this.link = link;
		this.rolpros = rolpros;}
	@Id
	@Column(name = "idpro", unique = true, nullable = false)
	public int getIdpro() {
		return this.idpro;}
	public void setIdpro(int idpro) {
		this.idpro = idpro;}
	@Column(name = "nombre", length = 100)
```

```
        public String getNombre() {
                return this.nombre;}
        public void setNombre(String nombre) {
                this.nombre = nombre;}
        @Column(name = "link", length = 100)
        public String getLink() {
                return this.link;}
        public void setLink(String link) {
                this.link = link;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "procesos")
        public Set getRolpros() {
                return this.rolpros;}
        public void setRolpros(Set rolpros) {
                this.rolpros = rolpros;}
}
public class Regbautismo implements java.io.Serializable {
        private int idregistro;
        private Libros libros;
        private Sacramentos sacramentos;
        private Fieles fieles;
        private Date fecha;
        //Variables incrementadas
        SimpleDateFormat vDateFormat = new SimpleDateFormat("dd-MM-yyyy");
        public Regbautismo() {}
        public Regbautismo(int idregistro) {
                this.idregistro = idregistro;}
        public Regbautismo(int idregistro, Libros libros, Sacramentos sacramentos,
                Fieles fieles, Date fecha) {
                this.idregistro = idregistro;
```

```
        this.libros = libros;
        this.sacramentos = sacramentos;
        this.fieles = fieles;
        this.fecha = fecha;}
@Id
@Column(name = "idregistro", unique = true, nullable = false)
public int getIdregistro() {
        return this.idregistro;}
public void setIdregistro(int idregistro) {
        this.idregistro = idregistro;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idlibro")
public Libros getLibros() {
        return this.libros;}
public void setLibros(Libros libros) {
        this.libros = libros;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idsacra")
public Sacramentos getSacramentos() {
        return this.sacramentos;}
public void setSacramentos(Sacramentos sacramentos) {
        this.sacramentos = sacramentos;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idp")
public Fieles getFieles() {
        return this.fieles;}
public void setFieles(Fieles fieles) {
        this.fieles = fieles;}
@Temporal(TemporalType.DATE)
@Column(name = "fecha", length = 13)
```

```
        public Date getFecha() {
                return this.fecha;}
        public void setFecha(Date fecha) {
                this.fecha = fecha;}
        //Metodos incrementados
        public String getFechaReg() {
                if(this.fecha != null)
                        return vDateFormat.format(this.fecha);
                else
                        return "";}
}
public class Regcomunion implements java.io.Serializable {
        private int idcomu;
        private Libros libros;
        private Sacramentos sacramentos;
        private Fieles fieles;
        private Date fecha;
        private String ano;
        public Regcomunion() {}
        public Regcomunion(int idcomu) {
                this.idcomu = idcomu;}
        public Regcomunion(int idcomu, Libros libros, Sacramentos sacramentos,
                Fieles fieles, Date fecha, String ano) {
                this.idcomu = idcomu;
                this.libros = libros;
                this.sacramentos = sacramentos;
                this.fieles = fieles;
                this.fecha = fecha;
                this.ano = ano;}
        @Id
```

```
@Column(name = "idcomu", unique = true, nullable = false)
public int getIdcomu() {
        return this.idcomu;}
public void setIdcomu(int idcomu) {
        this.idcomu = idcomu;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idlibro")
public Libros getLibros() {
        return this.libros;}
public void setLibros(Libros libros) {
        this.libros = libros;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idsacra")
public Sacramentos getSacramentos() {
        return this.sacramentos;}
public void setSacramentos(Sacramentos sacramentos) {
        this.sacramentos = sacramentos;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idp")
public Fieles getFieles() {
        return this.fieles;}
public void setFieles(Fieles fieles) {
        this.fieles = fieles;}
@Temporal(TemporalType.DATE)
@Column(name = "fecha", length = 13)
public Date getFecha() {
        return this.fecha;}
public void setFecha(Date fecha) {
        this.fecha = fecha;}
@Column(name = "ano")
```

```
        public String getAno() {
                return this.ano;}
        public void setAno(String ano) {
                this.ano = ano;}
}
public class Regconfirmacion implements java.io.Serializable {
        private int idconfir;
        private Libros libros;
        private Sacramentos sacramentos;
        private Fieles fieles;
        private Date fecha;
        private String ano;
        public Regconfirmacion() {}
        public Regconfirmacion(int idconfir) {
                this.idconfir = idconfir;}
        public Regconfirmacion(int idconfir, Libros libros,
                Sacramentos sacramentos, Fieles fieles, Date fecha, String ano) {
                this.idconfir = idconfir;
                this.libros = libros;
                this.sacramentos = sacramentos;
                this.fieles = fieles;
                this.fecha = fecha;
                this.ano = ano;}
        @Id
        @Column(name = "idconfir", unique = true, nullable = false)
        public int getIdconfir() {
                return this.idconfir;}
        public void setIdconfir(int idconfir) {
                this.idconfir = idconfir;}
        @ManyToOne(fetch = FetchType.LAZY)
```

```
	@JoinColumn(name = "idlibro")
	public Libros getLibros() {
		return this.libros;}
	public void setLibros(Libros libros) {
		this.libros = libros;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idsacra")
	public Sacramentos getSacramentos() {
		return this.sacramentos;}
	public void setSacramentos(Sacramentos sacramentos) {
		this.sacramentos = sacramentos;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idp")
	public Fieles getFieles() {
		return this.fieles;}
	public void setFieles(Fieles fieles) {
		this.fieles = fieles;}
	@Temporal(TemporalType.DATE)
	@Column(name = "fecha", length = 13)
	public Date getFecha() {
		return this.fecha;}
	public void setFecha(Date fecha) {
		this.fecha = fecha;}
	@Column(name = "ano")
	public String getAno() {
		return this.ano;}
	public void setAno(String ano) {
		this.ano = ano;}
}
```

```
public class Regmatrimonio implements java.io.Serializable {
	private int idmatri;
	private Libros libros;
	private Sacramentos sacramentos;
	private Fieles fieles;
	private Date fecha;
	public Regmatrimonio() {}
	public Regmatrimonio(int idmatri) {
		this.idmatri = idmatri;}
	public Regmatrimonio(int idmatri, Libros libros, Sacramentos sacramentos,
		Fieles fieles, Date fecha) {
		this.idmatri = idmatri;
		this.libros = libros;
		this.sacramentos = sacramentos;
		this.fieles = fieles;
		this.fecha = fecha;}
	@Id
	@Column(name = "idmatri", unique = true, nullable = false)
	public int getIdmatri() {
		return this.idmatri;}
	public void setIdmatri(int idmatri) {
		this.idmatri = idmatri;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idlibro")
	public Libros getLibros() {
		return this.libros;}
	public void setLibros(Libros libros) {
		this.libros = libros;}
	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "idsacra")
```

```
        public Sacramentos getSacramentos() {
                return this.sacramentos;}
        public void setSacramentos(Sacramentos sacramentos) {
                this.sacramentos = sacramentos;}
        @ManyToOne(fetch = FetchType.LAZY)
        @JoinColumn(name = "idp")
        public Fieles getFieles() {
                return this.fieles;}
        public void setFieles(Fieles fieles) {
                this.fieles = fieles;}
        @Temporal(TemporalType.DATE)
        @Column(name = "fecha", length = 13)
        public Date getFecha() {
                return this.fecha;}
        public void setFecha(Date fecha) {
                this.fecha = fecha;}
}
public class Relaciones implements java.io.Serializable {
        private int idrela;
        private String nombre;
        private Set relafiels = new HashSet(0);
        public Relaciones() {}
        public Relaciones(int idrela) {
                this.idrela = idrela;}
        public Relaciones(int idrela, String nombre, Set relafiels) {
                this.idrela = idrela;
                this.nombre = nombre;
                this.relafiels = relafiels;}
        @Id
        @Column(name = "idrela", unique = true, nullable = false)
```

```
        public int getIdrela() {
                return this.idrela;}
        public void setIdrela(int idrela) {
                this.idrela = idrela;}
        @Column(name = "nombre", length = 50)
        public String getNombre() {
                return this.nombre;}
        public void setNombre(String nombre) {
                this.nombre = nombre;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "relaciones")
        public Set getRelafiels() {
                return this.relafiels;}
        public void setRelafiels(Set relafiels) {
                this.relafiels = relafiels;}
}
public class Relafiel implements java.io.Serializable {
        private int idrelafiel;
        private Fieles fielesByIdperr;
        private Fieles fielesByIdp;
        private Relaciones relaciones;
        private Boolean estado;
        private String por;
        public Relafiel() {}
        public Relafiel(int idrelafiel, Relaciones relaciones) {
                this.idrelafiel = idrelafiel;
                this.relaciones = relaciones;}
        public Relafiel(int idrelafiel, Fieles fieles, Relaciones relaciones,
                Boolean estado) {
                this.idrelafiel = idrelafiel;
```

```
        this.relaciones = relaciones;
        this.estado = estado;}
@Id
@Column(name = "idrelafiel", unique = true, nullable = false)
public int getIdrelafiel() {
        return this.idrelafiel;}
public void setIdrelafiel(int idrelafiel) {
        this.idrelafiel = idrelafiel;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idperr")
public Fieles getFielesByIdperr() {
        return this.fielesByIdperr;}
public void setFielesByIdperr(Fieles fielesByIdperr) {
        this.fielesByIdperr = fielesByIdperr;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idp")
public Fieles getFielesByIdp() {
        return this.fielesByIdp;}
public void setFielesByIdp(Fieles fielesByIdp) {
        this.fielesByIdp = fielesByIdp;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idrela", nullable = false)
public Relaciones getRelaciones() {
        return this.relaciones;}
public void setRelaciones(Relaciones relaciones) {
        this.relaciones = relaciones;}
@Column(name = "estado")
public Boolean getEstado() {
        return this.estado;}
public void setEstado(Boolean estado) {
```

```
		this.estado = estado;}
	@Column(name = "por", length = 50)
	public String getPor() {
		return this.por;}
	public void setPor(String por) {
		this.por = por;}
}
public class Roles implements java.io.Serializable {
	private int idrol;
	private String nombre;
	private String descripcion;
	private Set usurols = new HashSet(0);
	private Set rolpros = new HashSet(0);
	public Roles() {}
	public Roles(int idrol) {
		this.idrol = idrol;}
	public Roles(int idrol, String nombre, String descripcion, Set usurols,
		Set rolpros) {
		this.idrol = idrol;
		this.nombre = nombre;
		this.descripcion = descripcion;
		this.usurols = usurols;
		this.rolpros = rolpros;}
	@Id
	@Column(name = "idrol", unique = true, nullable = false)
	public int getIdrol() {
		return this.idrol;}
	public void setIdrol(int idrol) {
		this.idrol = idrol;}
	@Column(name = "nombre")
```

```
        public String getNombre() {
                return this.nombre;}
        public void setNombre(String nombre) {
                this.nombre = nombre;}
        @Column(name = "descripcion", length = 100)
        public String getDescripcion() {
                return this.descripcion;}
        public void setDescripcion(String descripcion) {
                this.descripcion = descripcion;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "roles")
        public Set getUsurols() {
                return this.usurols;}
        public void setUsurols(Set usurols) {
                this.usurols = usurols;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "roles")
        public Set getRolpros() {
                return this.rolpros;}
        public void setRolpros(Set rolpros) {
                this.rolpros = rolpros;}
}
public class Rolpro implements java.io.Serializable {
        private int idrolpro;
        private Procesos procesos;
        private Roles roles;
        private Boolean estado;
        public Rolpro() {}
        public Rolpro(int idrolpro, Procesos procesos, Roles roles) {
                this.idrolpro = idrolpro;
```

```
        this.procesos = procesos;
        this.roles = roles;}
public Rolpro(int idrolpro, Procesos procesos, Roles roles, Boolean estado) {
        this.idrolpro = idrolpro;
        this.procesos = procesos;
        this.roles = roles;
        this.estado = estado;}
@Id
@Column(name = "idrolpro", unique = true, nullable = false)
public int getIdrolpro() {
        return this.idrolpro;}
public void setIdrolpro(int idrolpro) {
        this.idrolpro = idrolpro;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idpro", nullable = false)
public Procesos getProcesos() {
        return this.procesos;}
public void setProcesos(Procesos procesos) {
        this.procesos = procesos;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idrol", nullable = false)
public Roles getRoles() {
        return this.roles;}
public void setRoles(Roles roles) {
        this.roles = roles;}
@Column(name = "estado")
public Boolean getEstado() {
        return this.estado;}
public void setEstado(Boolean estado) {
        this.estado = estado;}}
```

```
public class Sacerdotes implements java.io.Serializable {
        private int idsacer;
        private String nombres;
        private String ap;
        private String am;
        private Date fnac;
        private String direccion;
        private Integer telefono;
        private Set parrsas = new HashSet(0);
        public Sacerdotes() {}
        public Sacerdotes(int idsacer) {
                this.idsacer = idsacer;}
        public Sacerdotes(int idsacer, String nombres, String ap, String am,
                Date fnac, String direccion, Integer telefono, Set parrsas) {
                this.idsacer = idsacer;
                this.nombres = nombres;
                this.ap = ap;
                this.am = am;
                this.fnac = fnac;
                this.direccion = direccion;
                this.telefono = telefono;
                this.parrsas = parrsas;}
        @Id
        @Column(name = "idsacer", unique = true, nullable = false)
        public int getIdsacer() {
                return this.idsacer;}
        public void setIdsacer(int idsacer) {
                this.idsacer = idsacer;}
        @Column(name = "nombres", length = 100)
        public String getNombres() {
```

```
        return this.nombres;}
public void setNombres(String nombres) {
        this.nombres = nombres;}
@Column(name = "ap", length = 100)
public String getAp() {
        return this.ap;}
public void setAp(String ap) {
        this.ap = ap;}
@Column(name = "am", length = 100)
public String getAm() {
        return this.am;}
public void setAm(String am) {
        this.am = am;}
@Temporal(TemporalType.DATE)
@Column(name = "fnac", length = 13)
public Date getFnac() {
        return this.fnac;}
public void setFnac(Date fnac) {
        this.fnac = fnac;}
@Column(name = "direccion")
public String getDireccion() {
        return this.direccion;}
public void setDireccion(String direccion) {
        this.direccion = direccion;}
@Column(name = "telefono")
public Integer getTelefono() {
        return this.telefono;}
public void setTelefono(Integer telefono) {
        this.telefono = telefono;}
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
```

```
mappedBy = "sacerdotes")
        public Set getParrsas() {
                return this.parrsas;}
        public void setParrsas(Set parrsas) {
                this.parrsas = parrsas;}
}
public class Sacramentos implements java.io.Serializable {
        private int idsacra;
        private String nombre;
        private Set libroses = new HashSet(0);
        private Set regmatrimonios = new HashSet(0);
        private Set regcomunions = new HashSet(0);
        private Set inscripcioneses = new HashSet(0);
        private Set regconfirmacions = new HashSet(0);
        private Set regbautismos = new HashSet(0);
        public Sacramentos() {}
        public Sacramentos(int idsacra) {
                this.idsacra = idsacra;}
        public Sacramentos(int idsacra, String nombre, Set libroses,
                        Set regmatrimonios, Set regcomunions, Set inscripcioneses,
                        Set regconfirmacions, Set regbautismos) {
                this.idsacra = idsacra;
                this.nombre = nombre;
                this.libroses = libroses;
                this.regmatrimonios = regmatrimonios;
                this.regcomunions = regcomunions;
                this.inscripcioneses = inscripcioneses;
                this.regconfirmacions = regconfirmacions;
                this.regbautismos = regbautismos;}
        @Id
```

```
        @Column(name = "idsacra", unique = true, nullable = false)
        public int getIdsacra() {
                return this.idsacra;}
        public void setIdsacra(int idsacra) {
                this.idsacra = idsacra;}
        @Column(name = "nombre", length = 100)
        public String getNombre() {
                return this.nombre;}
        public void setNombre(String nombre) {
                this.nombre = nombre;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
        public Set getLibroses() {
                return this.libroses;}
        public void setLibroses(Set libroses) {
                this.libroses = libroses;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
        public Set getRegmatrimonios() {
                return this.regmatrimonios;}
        public void setRegmatrimonios(Set regmatrimonios) {
                this.regmatrimonios = regmatrimonios;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
        public Set getRegcomunions() {
                return this.regcomunions;}
        public void setRegcomunions(Set regcomunions) {
                this.regcomunions = regcomunions;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
```

```
        public Set getInscripcioneses() {
                return this.inscripcioneses;}
        public void setInscripcioneses(Set inscripcioneses) {
                this.inscripcioneses = inscripcioneses;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
        public Set getRegconfirmacions() {
                return this.regconfirmacions;}
        public void setRegconfirmacions(Set regconfirmacions) {
                this.regconfirmacions = regconfirmacions;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "sacramentos")
        public Set getRegbautismos() {
                return this.regbautismos;}
        public void setRegbautismos(Set regbautismos) {
                this.regbautismos = regbautismos;}}
public class Usuarios implements java.io.Serializable {
        private int idusu;
        private Parroquia parroquia;
        private Integer ci;
        private String nombre;
        private String ap;
        private String am;
        private Date fnac;
        private String estado;
        private String direccion;
        private Integer telefono;
        private Set datoses = new HashSet(0);
        private Set usurols = new HashSet(0);
        public Usuarios() {}
```

```
public Usuarios(int idusu) {
        this.idusu = idusu;}
public Usuarios(int idusu, Parroquia parroquia, Integer ci, String nombre,
                String ap, String am, Date fnac, String direccion,
                Integer telefono, Set datoses, Set usurols) {
        this.idusu = idusu;
        this.parroquia = parroquia;
        this.ci = ci;
        this.nombre = nombre;
        this.ap = ap;
        this.am = am;
        this.fnac = fnac;
        this.direccion = direccion;
        this.telefono = telefono;
        this.datoses = datoses;
        this.usurols = usurols;}
@Id
@Column(name = "idusu", unique = true, nullable = false)
public int getIdusu() {
        return this.idusu;}
public void setIdusu(int idusu) {
        this.idusu = idusu;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idparr")
public Parroquia getParroquia() {
        return this.parroquia;}
public void setParroquia(Parroquia parroquia) {
        this.parroquia = parroquia;}
@Column(name = "ci")
public Integer getCi() {
```

```
        return this.ci;}
public void setCi(Integer ci) {
        this.ci = ci;}
@Column(name = "nombre", length = 100)
public String getNombre() {
        return this.nombre;}
public void setNombre(String nombre) {
        this.nombre = nombre;}
@Column(name = "ap", length = 100)
public String getAp() {
        return this.ap;}
public void setAp(String ap) {
        this.ap = ap;}
@Column(name = "am", length = 100)
public String getAm() {
        return this.am;}
public void setAm(String am) {
        this.am = am;}
@Temporal(TemporalType.DATE)
@Column(name = "fnac", length = 13)
public Date getFnac() {
        return this.fnac;}
public void setFnac(Date fnac) {
        this.fnac = fnac;}
@Column(name = "direccion", length = 100)
public String getDireccion() {
        return this.direccion;}
public void setDireccion(String direccion) {
        this.direccion = direccion;}
@Column(name = "telefono")
```

```
        public Integer getTelefono() {
                return this.telefono;}
        public void setTelefono(Integer telefono) {
                this.telefono = telefono;}
        @Column(name = "estado")
        public String getEstado() {
                return this.estado;}
        public void setEstado(String estado) {
                this.estado = estado;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "usuarios")
        public Set getDatoses() {
                return this.datoses;}
        public void setDatoses(Set datoses) {
                this.datoses = datoses;}
        @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "usuarios")
        public Set getUsurols() {
                return this.usurols;}
        public void setUsurols(Set usurols) {
                this.usurols = usurols;}
}
public class Usurol implements java.io.Serializable {
        private int idusurol;
        private Usuarios usuarios;
        private Roles roles;
        private Boolean estado;
        public Usurol() {}
        public Usurol(int idusurol, Usuarios usuarios, Roles roles) {
                this.idusurol = idusurol;
```

```
        this.usuarios = usuarios;
        this.roles = roles;}
public Usurol(int idusurol, Usuarios usuarios, Roles roles, Boolean estado) {
        this.idusurol = idusurol;
        this.usuarios = usuarios;
        this.roles = roles;
        this.estado = estado;}
@Id
@Column(name = "idusurol", unique = true, nullable = false)
public int getIdusurol() {
        return this.idusurol;}
public void setIdusurol(int idusurol) {
        this.idusurol = idusurol;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idusu", nullable = false)
public Usuarios getUsuarios() {
        return this.usuarios;}
public void setUsuarios(Usuarios usuarios) {
        this.usuarios = usuarios;}
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idrol", nullable = false)
public Roles getRoles() {
        return this.roles;}
public void setRoles(Roles roles) {
        this.roles = roles;}
@Column(name = "estado")
public Boolean getEstado() {
        return this.estado;}
public void setEstado(Boolean estado) {
        this.estado = estado;}.}
```

2.1.2.2.3.12 Modelo de Implementación

Este modelo es una colección de componentes y los subsistemas que los contienen. Estos componentes incluyen: ficheros ejecutables, ficheros de código fuente, y todo otro tipo de ficheros necesarios para la implantación y despliegue del sistema. (Este modelo es sólo una versión preliminar al final de la fase de Elaboración, posteriormente tiene bastante refinamiento).

2.1.2.2.3.12.1 Modelado de Diagrama de Paquetes

2.1.2.2.3.12.1.1 Diagrama de Paquetes

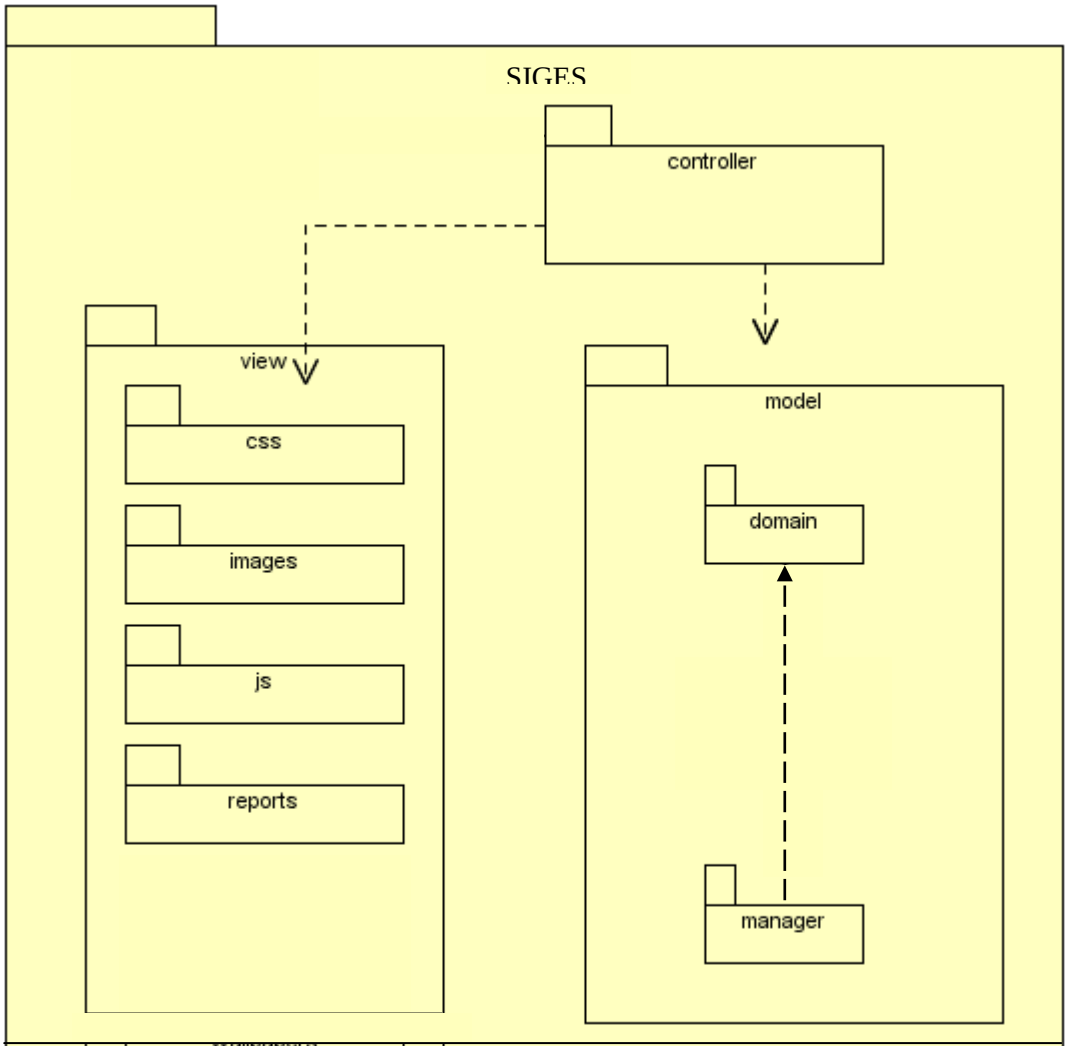


Figura Nº147. Diagrama de paquete

2.1.2.2.3.12.2 Modelado de Diagrama de Componentes

2.1.2.2.3.12.2.1 Diagrama de Componentes

2.1.2.2.3.12.2.1.1 Diagrama de Componente: Acceder al Sistema

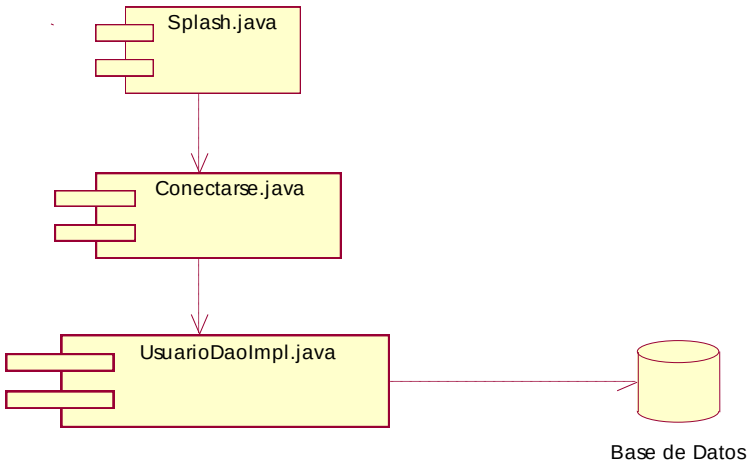


Figura Nº148. Diagrama de Componente: Acceder al Sistema

2.1.2.2.3.12.2.1.2 Diagrama de Componente: Guardar y Restaurar Bases de Datos

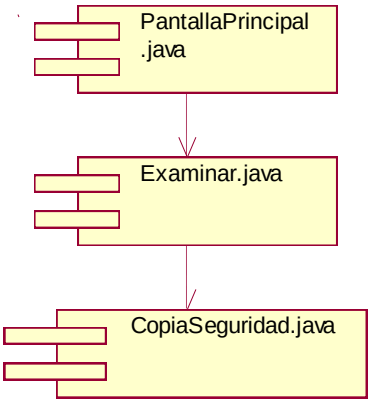


Figura Nº149. Diagrama de Componente: Guardar y restaurar Bases de Datos

2.1.2.2.3.12.2.1.3 Diagrama de Componente: Gestionar Usuarios

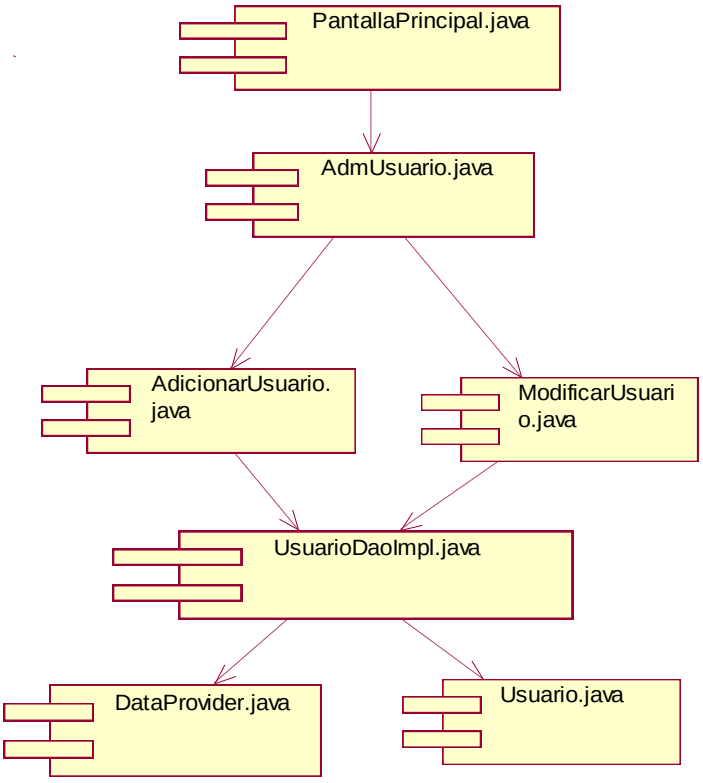


Figura Nº150. Diagrama de Componente: Gestionar Usuarios

2.1.2.2.3.12.2.1.4 Diagrama de Componente: Rol - Acceso

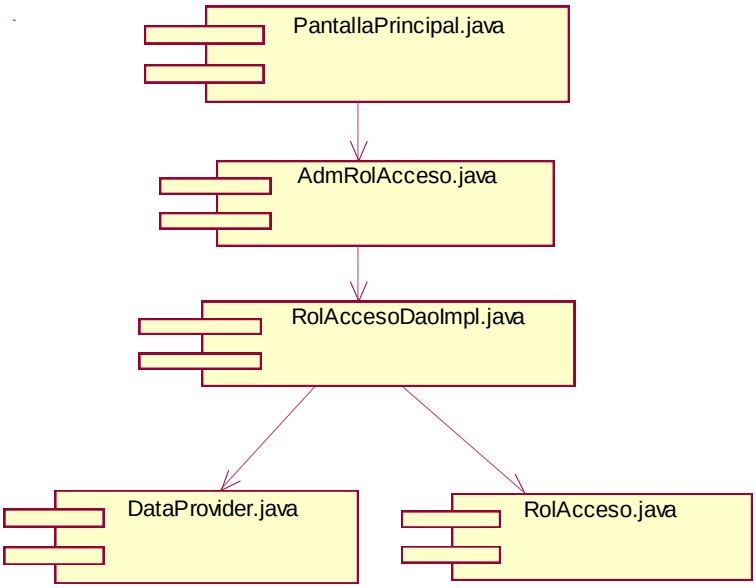


Figura Nº151. Diagrama de Componente: Rol - Acceso

2.1.2.2.3.13 Modelo de Despliegue

Este modelo muestra el despliegue la configuración de tipos de nodos del sistema, en los cuales se hará el despliegue de los componentes.

2.1.2.2.3.13.1 Modelado de Diagrama de Despliegue

2.1.2.2.3.13.1.1 Diagrama de despliegue

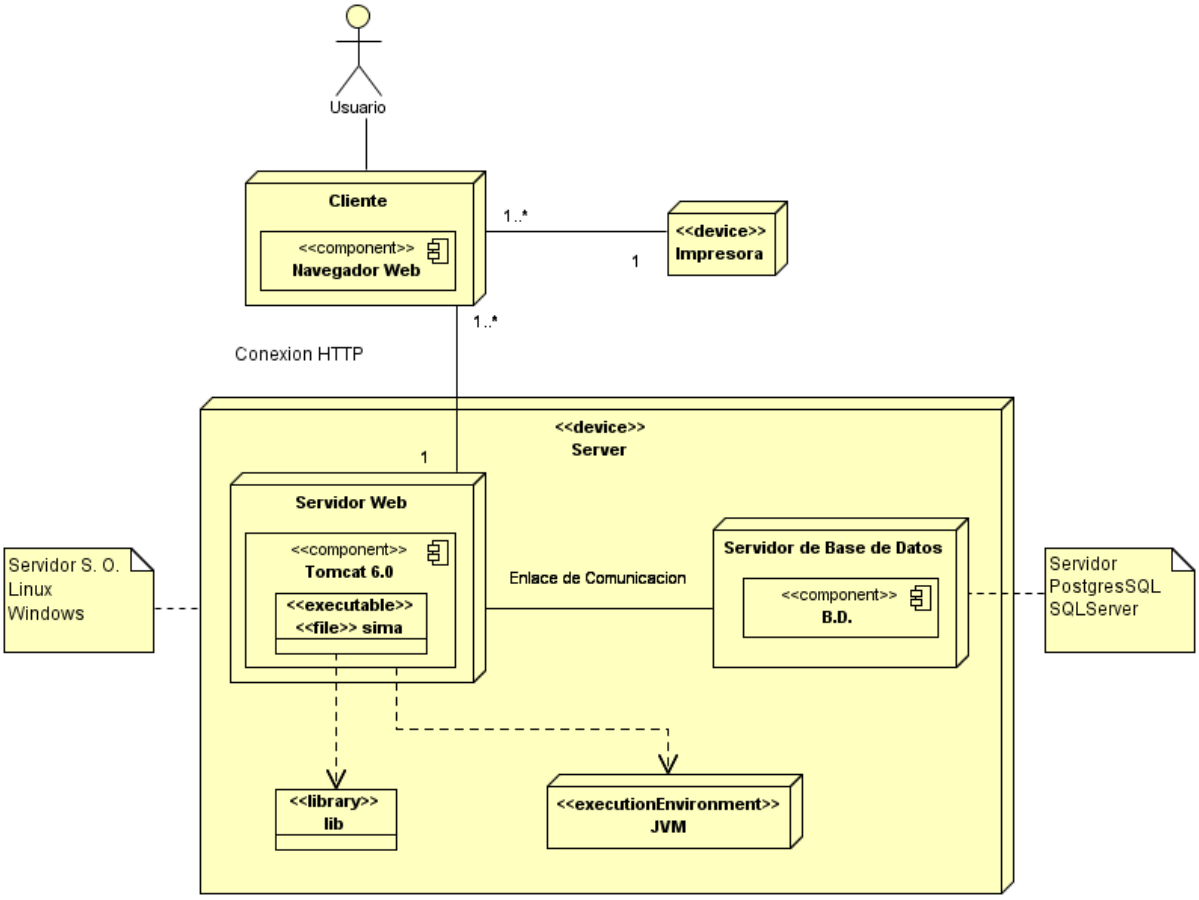


Figura Nº152. Diagrama de despliegue

2.1.2.2.3.14 Casos de Prueba

Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba, y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevará asociado un procedimiento de prueba con las instrucciones para realizar la prueba, y dependiendo del tipo de prueba dicho procedimiento podrá ser automatizable mediante un script de prueba.

2.1.2.2.3.14.1 Plan de Pruebas

2.1.2.2.3.14.1.1 Descripción de Aspectos Generales

Esta sección establece el alcance y el objetivo del Plan de Pruebas. Es aquí donde se describen los aspectos fundamentales del esfuerzo que se hará para probar cada uno de los módulos que conforman el sistema descrito en este Plan de Desarrollo de Software, independiente las características y tamaño que ésta pueda tener.

2.1.2.2.3.14.1.1.1 Objetivo

Este Plan de Pruebas fue desarrollado con el fin de guiar el proceso de pruebas al proyecto SIMA Sistema Informático para el Mejoramiento de Afiliación desarrollado por nuestro grupo de trabajo, esto con el fin de asegurar una excelente calidad del software desarrollado, encontrando para ello errores que puedan perjudicar en el funcionamiento de este, los cuales una vez corregidos, se podrá elaborar un documento que presente las evaluaciones correspondientes en el plazo más corto posible.

2.1.2.2.3.14.1.1.2 Entorno o Marco

La necesidad de aplicar este plan de pruebas radica en la necesidad de obtener un software de calidad, que cumpla con los requerimientos de la Caja Nacional de Salud Regional Tarija, además que sea aprobado por la comunidad de usuarios finales des sistema.

2.1.2.2.3.14.1.1.3 Arquitectura Técnica

Para la realización de las pruebas se tomará en cuenta los módulos desarrollados en la fase de construcción y los diagramas realizados en la fase de elaboración.

2.1.2.2.3.14.1.1.4 Especificaciones del Software y Hardware

Corresponde a una lista individualizada de todo el hardware y el software que utiliza la aplicación, incluyendo proveedores y versiones.

HARDWARE
Computadora con la suficiente capacidad de soportar el software

Tabla Nº33. Hardware

SOFTWARE		
Aplicación	Versión	Descripción
Sistema Informático de Gestión Sacramental	0.3	Sistema desarrollado por el director de proyecto.
Rational Rose Enterprise Edition	7.6	Herramienta de Diseño y Desarrollo de Diagramas UML para RUP
Visual Paradigm for UML Enterprise Edition	6.4	Herramienta de Diseño y Desarrollo de Diagramas UML para RUP
PostgreSQL	8.3	Motor de Base de Datos Relacional.

Tabla Nº34. Software

OTROS		
Descripción	Versión	Descripción
Documentación	0.3	Documentación de Desarrollo de Software

Tabla Nº35. Otros

2.1.2.2.3.14.1.1.5 Alcance

Este plan describe las pruebas del sistema, que serán aplicadas a los componentes del Sistema Informático para el Mejoramiento de Afiliación.

Se asume que antes de probar cada módulo habrá una revisión informal y solo el código que ha sido revisado como exitoso será probado.

Las unidades de prueba serán realizadas a través del programa manejador de pruebas que ejecuta el chequeo de los límites y las pruebas básicas de caja negra.

Las siguientes medidas de desempeño serán probadas:

Validación correcta de las Entradas de Datos.

Tiempo de respuesta de carga del Sitio Web.

Tiempo de respuesta y Pool de conexiones a la Base de Datos.

2.1.2.2.3.14.1.1.6 Referencias

Glosario del Proyecto.

Plan de Desarrollo de Software.

2.1.2.2.3.14.2 Ejecución de las Pruebas

2.1.2.2.3.14.2.1 Partición Equivalente : Casos de Prueba Caja Negra

Interfaz: Acceder al Sistema

Usuario: Alfanumérico 50 caracteres

Clave: Alfanumérico 50 caracteres

Condición de Entrada	Clases Equivalentes Válidas	Clases Equivalentes Inválidas
Tipo: Usuario Tamaño: Usuario	1_ Alfanumérico 2_ 1<=Caracteres<=50	3_ Vacío 4_ Caracteres >50
Tipo: Clave	4_ Alfanumérico	6_ Vacío

Tamaño: Clave	5_ 1<=Caracteres<=50	7_ Caracteres >50

Tabla Nº36. Partición Equivalente. Interfaz: Ingresar al Sistema

Casos de Prueba válidas

CP1	
Usuario	Clave
Administrador	administrador

Cubre las clases de equivalencias válidas: 1-2-4-5

Casos de Pruebas Inválidas

CP2	
Usuario	Clave
Administrador	

Cubre las clases de equivalencias inválidas: 3

CP3	
Usuario	Clave
	administrador

Cubre las clases de equivalencias inválidas: 6

Interfaz: Asignar Login y Clave

Username Alfanumérico 50 Caracteres

Password Alfanumérico 50 Caracteres

Password Verificar Alfanumérico 50 Caracteres

Condición de Entrada	Clases Equivalentes Válidas	Clases Equivalentes Inválidas
Tipo: Username Tamaño: Username	1_ Alfanumérico 2_ 1<=Caracteres<=50	3_ Vacío 4_ Caracteres >50
Tipo: Password Tamaño: Password	5_ Alfanumérico 6_ 1<=Caracteres<=50	7_ Vacío 8_ Caracteres >50

Tipo: Password Verificar Tamaño: Password Verificar	9_ Alfanumérico 10_ 1<=Caracteres<=50	11_ Distinto de Password 12_ Caracteres >50

Tabla Nº37. Partición Equivalente. Interfaz: Asignar Login y Clave

Casos de Prueba válidas

CP1		
Userrname	Password	Password Verificar
*******	*****	*****

Cubre las clases de equivalencias válidas: 1-5-9

Casos de Pruebas Inválidas

CP		
Userrname	Password	Password Verificar
	*****	*****

Cubre las clases de equivalencias inválidas: 3

CP3		
Userrname	Password	Password Verificar
*******		*****

Cubre las clases de equivalencias inválidas: 5

CP4		
Userrname	Password	Password Verificar
*******	*****	********

Cubre las clases de equivalencias inválidas: 9

2.1.2.2.3.14.2.1.1 Partición Equivalente : Casos de Prueba Caja Blanca

El objetivo de la prueba de software es descubrir errores, es ver una forma de ahorrar código, depurar iteración para contabilizar el mayor porcentaje del esfuerzo técnico del proceso de desarrollo de software.

Para ello existe una serie de herramienta que ayudan a verificar este tipo de pruebas en este caso para nuestro proyecto se ha utilizado la herramienta PMD

PMD es una herramienta de auditoría y verificación de código desarrollada por SourceForge.

PMD escanea el código Java y busca posibles problemas en potencia como pueden ser:

• Possible bugs (sentences try /catch/finally/switch vacias)

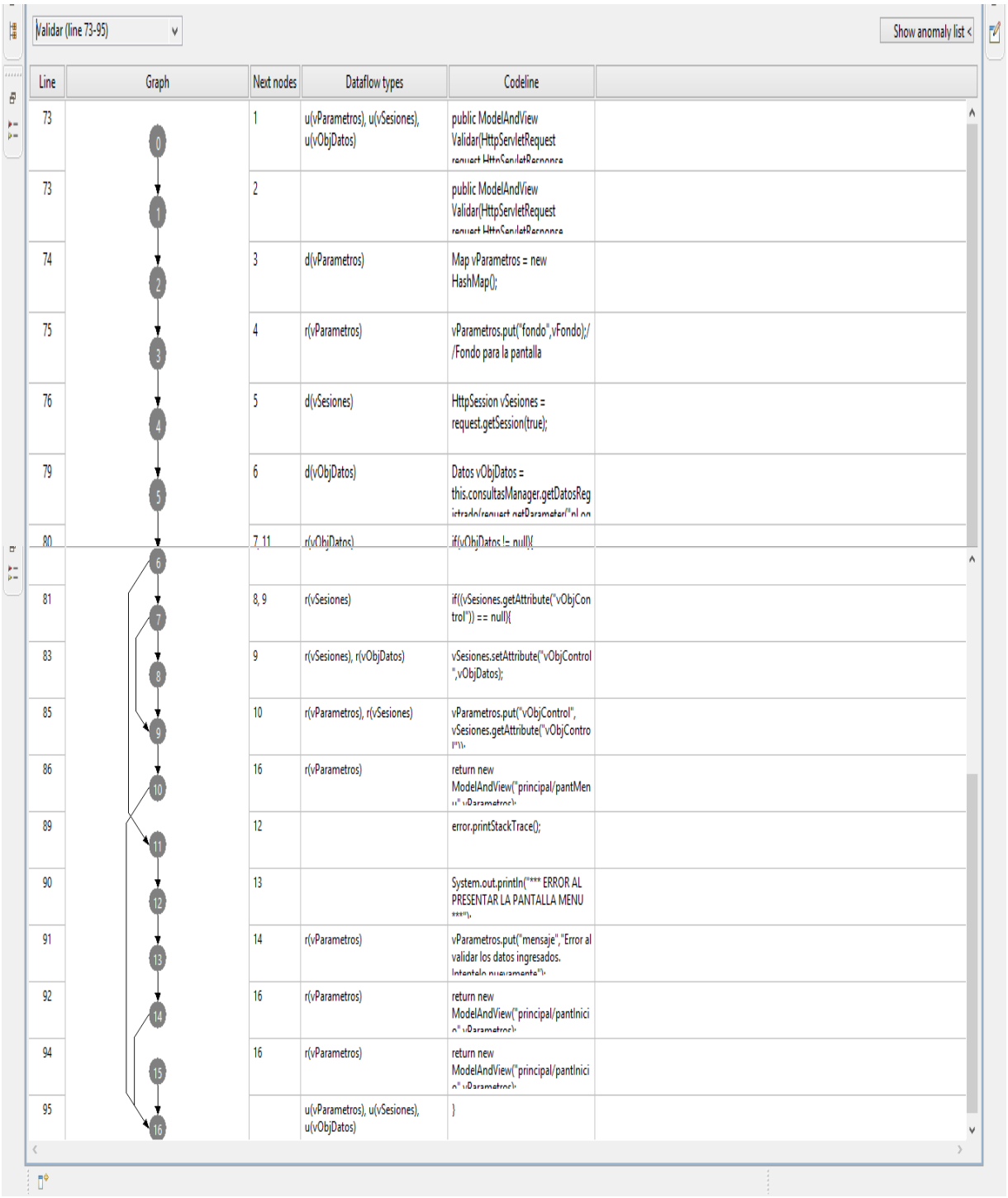
• Código muerto (variables locales, parámetros y métodos privados que no se usan)

• Código no óptimo (derroches en el uso de String/StringBuffer)

• Expresiones excesivamente complicadas (sentencias if innecesarias, implementaciones con bucles while)

• Código duplicado (código copiar-pegar que significa errores copiados-pegados)

2.1.2.2.3.14.2.1.2 Casos de Prueba Caja Blanca: Validar



Line	Graph	Next nodes	Dataflow types	Codeline
73	0	1	u(vParametros), u(vSesiones), u(vObjDatos)	public ModelAndView Validar(HttpServletRequest request,HttpServletResponse
73	1	2		public ModelAndView Validar(HttpServletRequest request,HttpServletResponse
74	2	3	d(vParametros)	Map vParametros = new HashMap();
75	3	4	r(vParametros)	vParametros.put("fondo",vFondo);//Fondo para la pantalla
76	4	5	d(vSesiones)	HttpSession vSesiones = request.getSession(true);
79	5	6	d(vObjDatos)	Datos vObjDatos = this.consultasManager.getDatosRegistrado(request.getParameter("pLog
80	6	7, 11	r(vObjDatos)	if(vObjDatos != null){
81	7	8, 9	r(vSesiones)	if((vSesiones.getAttribute("vObjControl")) == null){
83	8	9	r(vSesiones), r(vObjDatos)	vSesiones.setAttribute("vObjControl",vObjDatos);
85	9	10	r(vParametros), r(vSesiones)	vParametros.put("vObjControl", vSesiones.getAttribute("vObjControl"));
86	10	16	r(vParametros)	return new ModelAndView("principal/pantMenu",vParametros);
89	11	12		error.printStackTrace();
90	12	13		System.out.println("*** ERROR AL PRESENTAR LA PANTALLA MENU ***");
91	13	14	r(vParametros)	vParametros.put("mensaje","Error al validar los datos ingresados. Intentelo nuevamente");
92	14	16	r(vParametros)	return new ModelAndView("principal/pantInicio",vParametros);
94	15	16	r(vParametros)	return new ModelAndView("principal/pantInicio",vParametros);
95	16		u(vParametros), u(vSesiones), u(vObjDatos)	}

Figura Nº153. Caja Blanca Validar

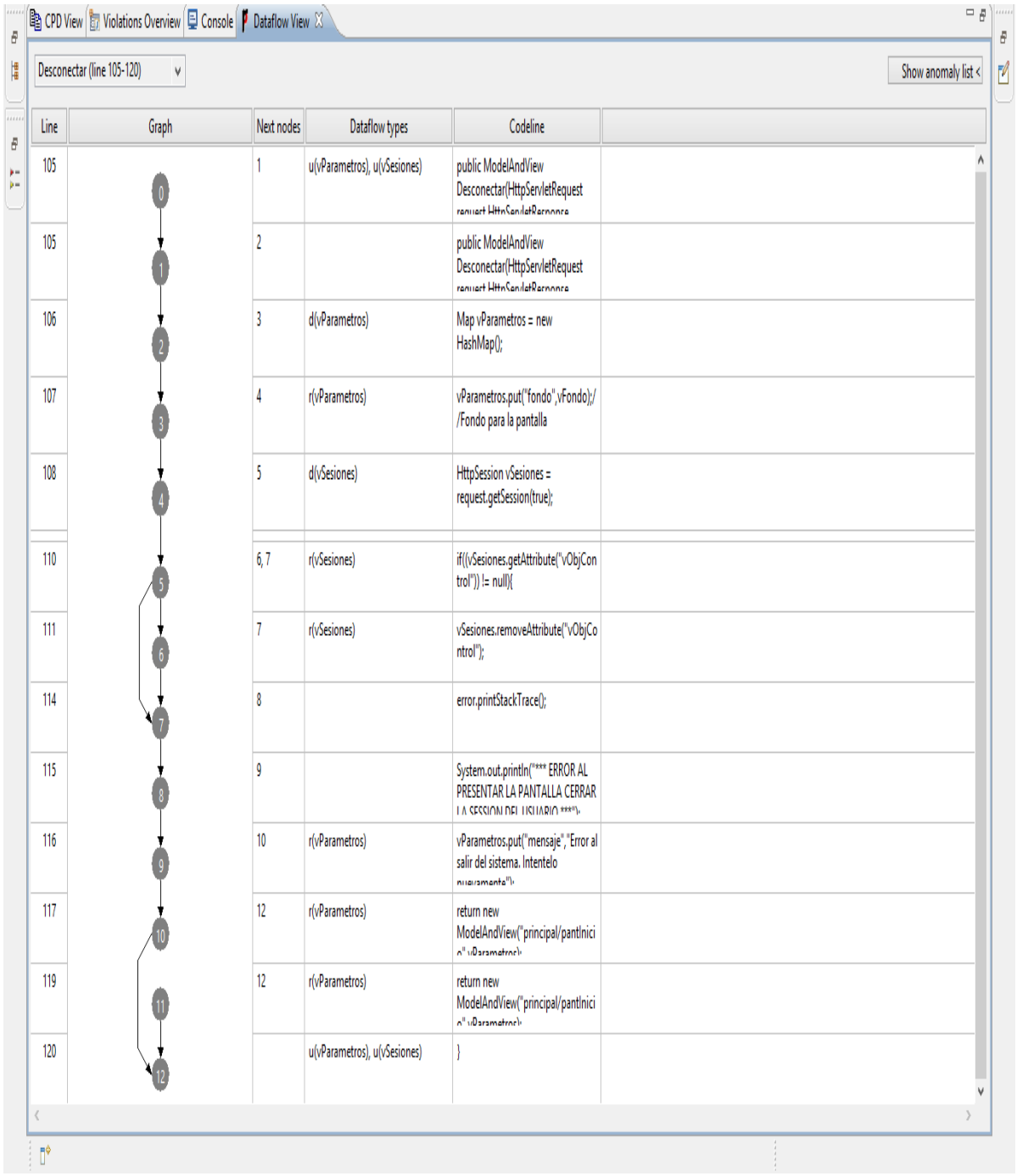


Figura Nº154. Caja Blanca

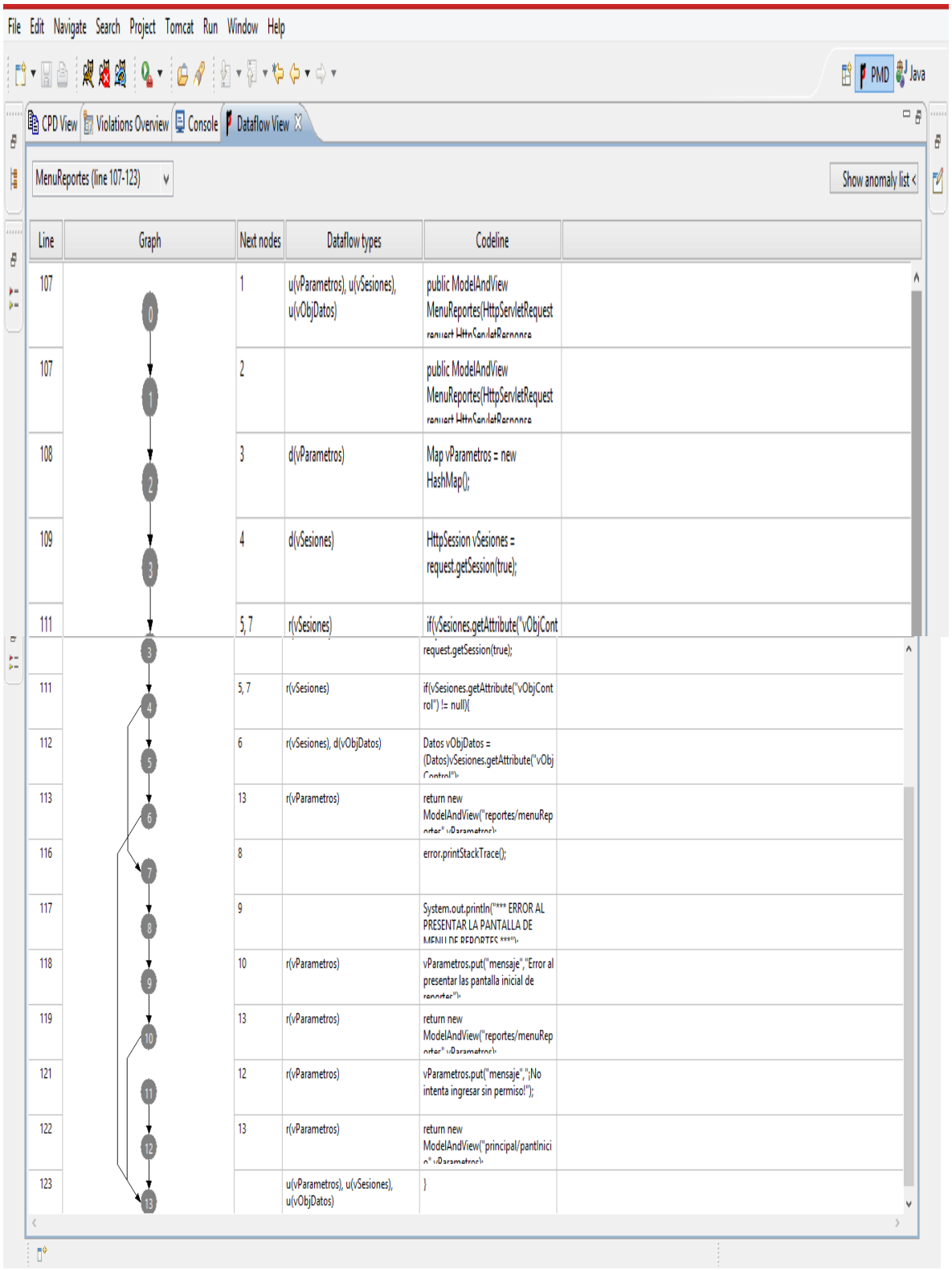


Figura Nº155. Caja Blanca Menú Reportes

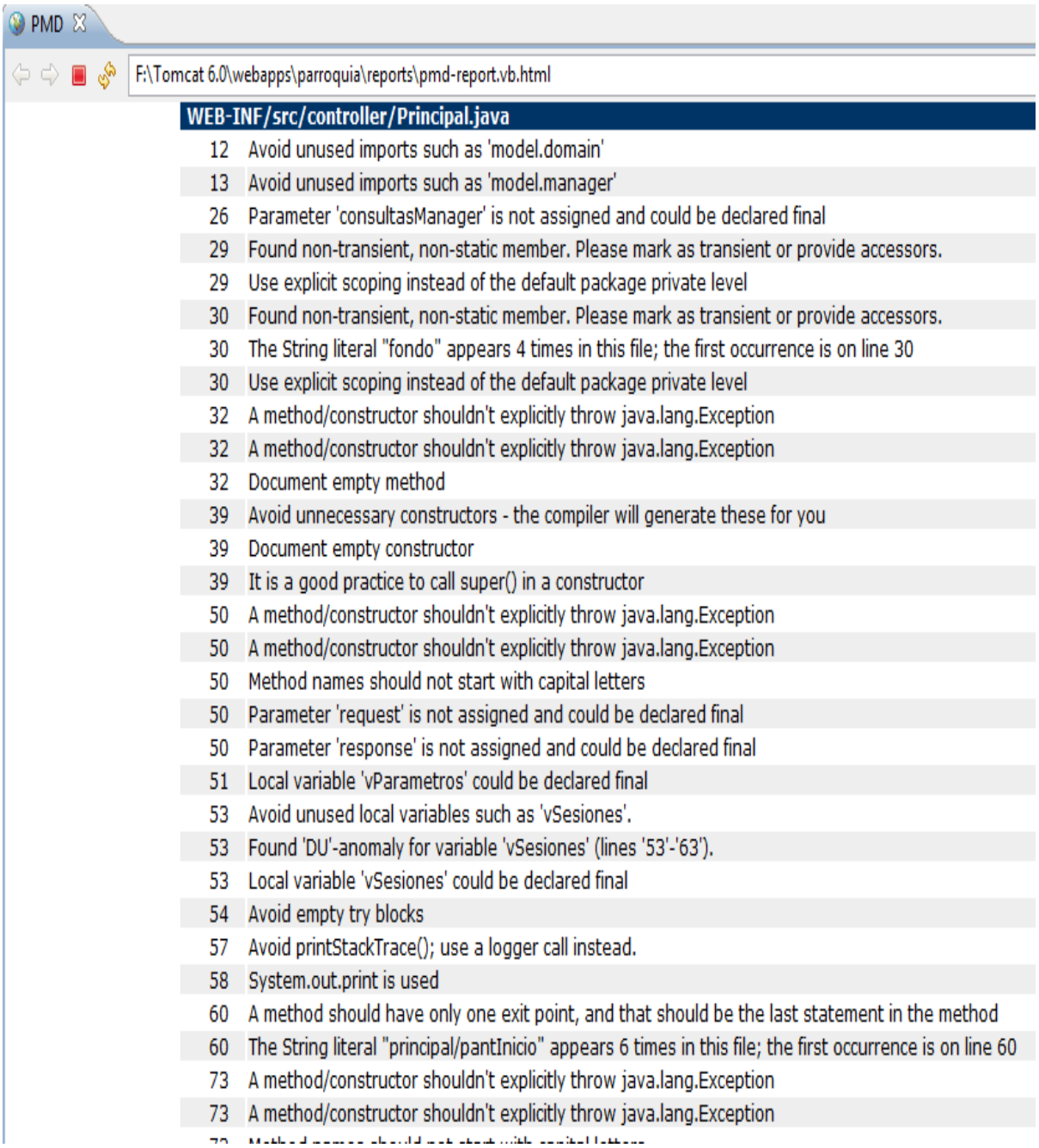
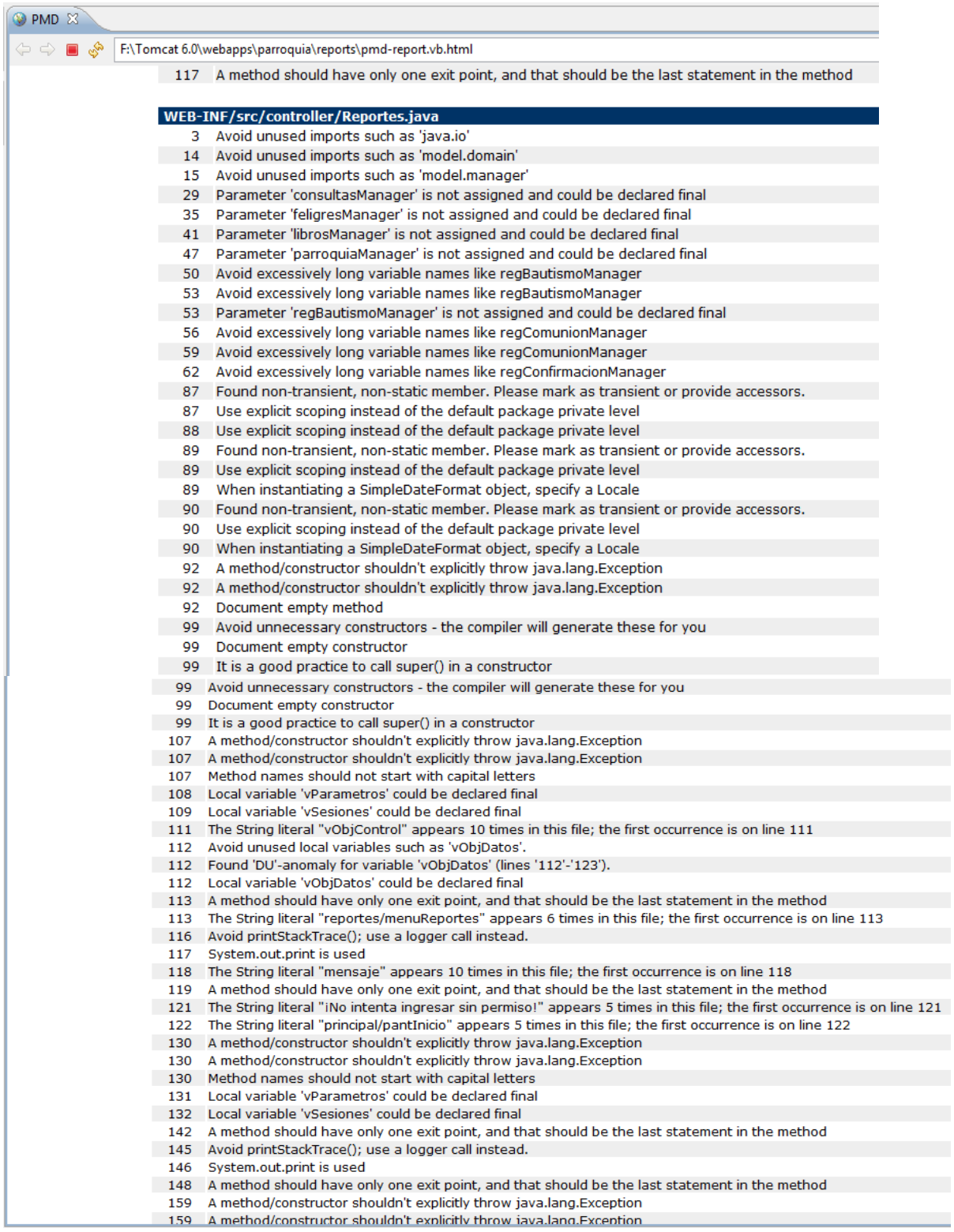


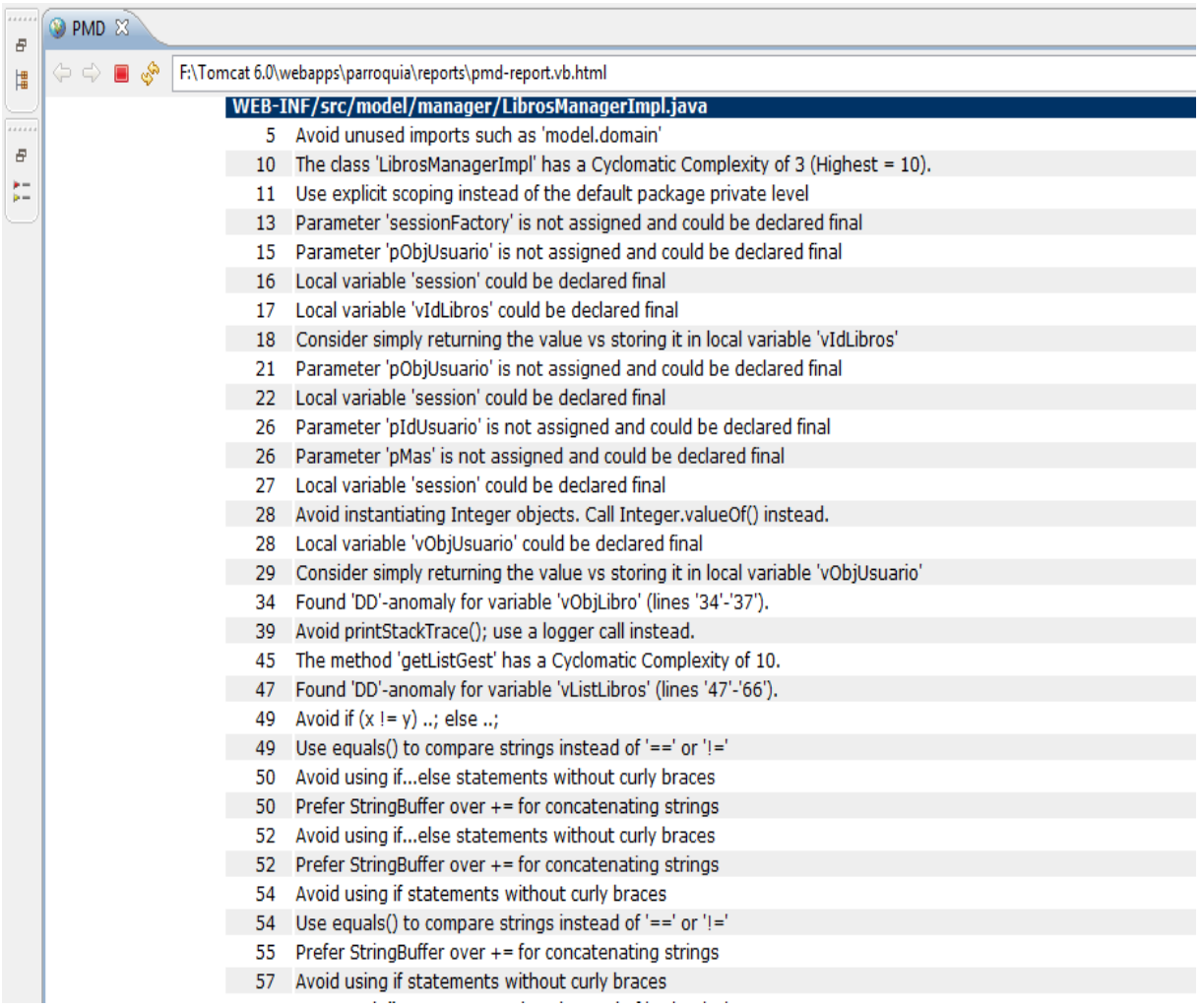
Figura N°156. Caja Blanca Menú Reportes

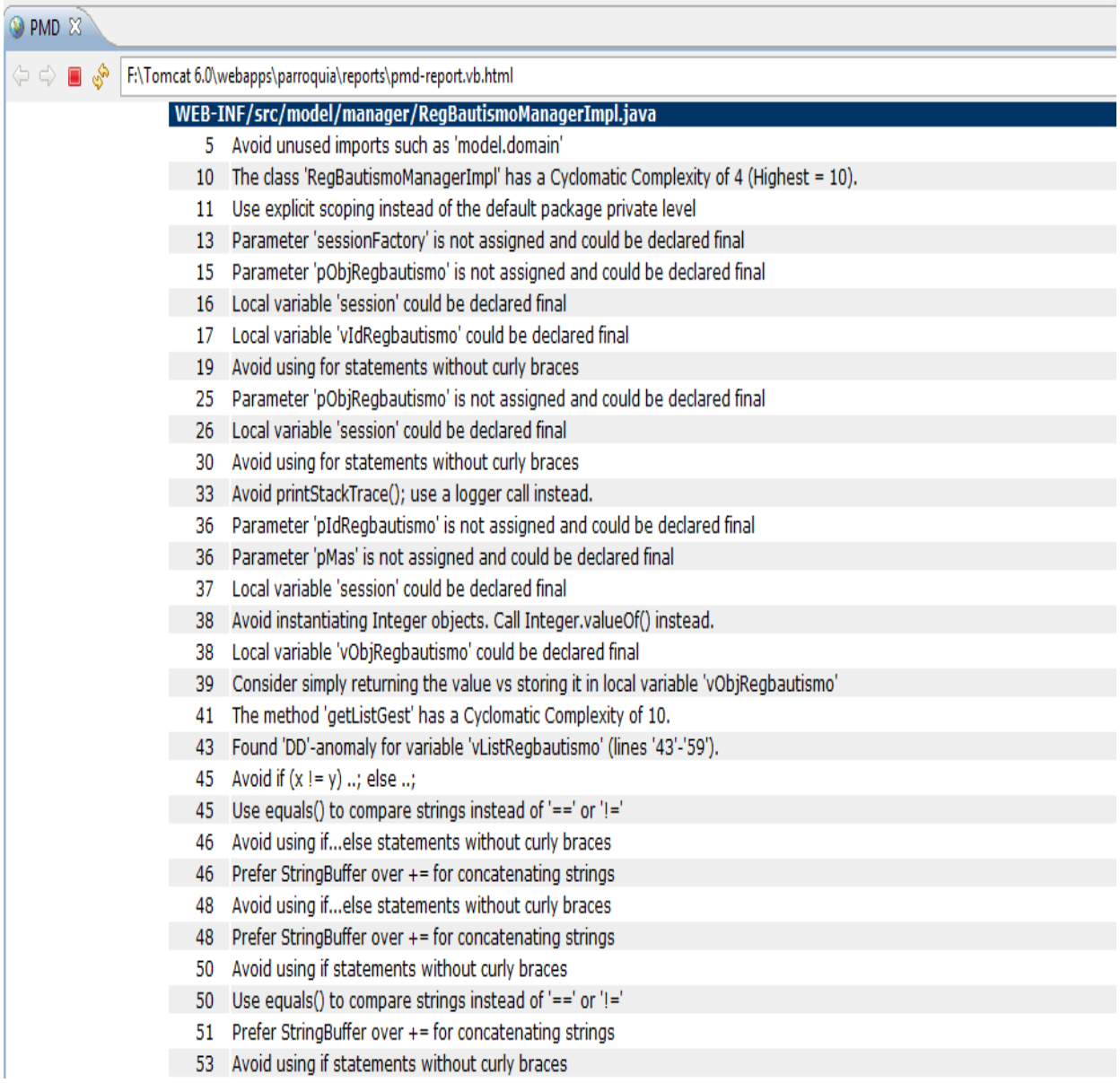


117	A method should have only one exit point, and that should be the last statement in the method

WEB-INF/src/controller/Reportes.java

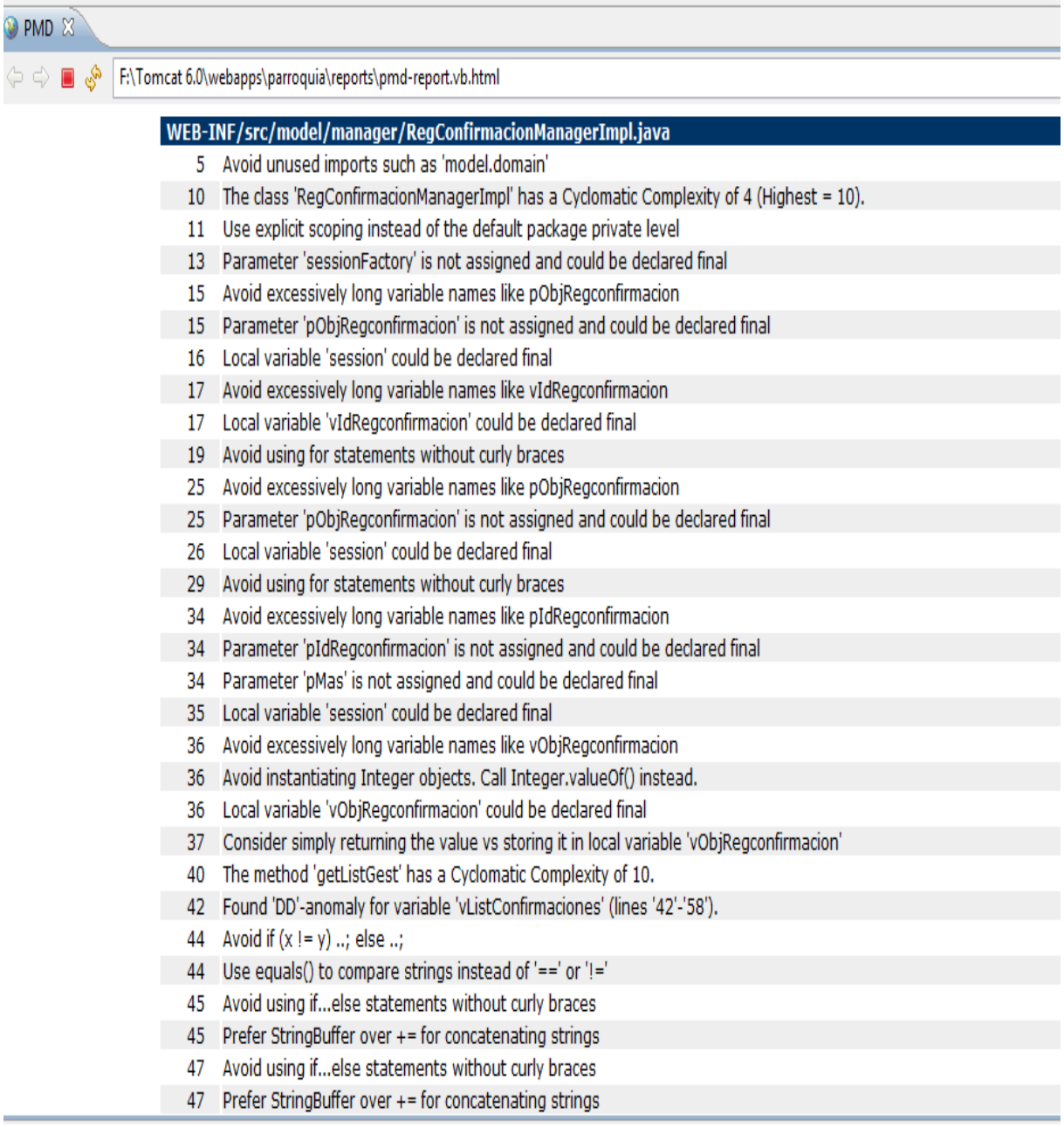
3	Avoid unused imports such as 'java.io'
14	Avoid unused imports such as 'model.domain'
15	Avoid unused imports such as 'model.manager'
29	Parameter 'consultasManager' is not assigned and could be declared final
35	Parameter 'feligresManager' is not assigned and could be declared final
41	Parameter 'librosManager' is not assigned and could be declared final
47	Parameter 'parroquiaManager' is not assigned and could be declared final
50	Avoid excessively long variable names like regBautismoManager
53	Avoid excessively long variable names like regBautismoManager
53	Parameter 'regBautismoManager' is not assigned and could be declared final
56	Avoid excessively long variable names like regComunionManager
59	Avoid excessively long variable names like regComunionManager
62	Avoid excessively long variable names like regConfirmacionManager
87	Found non-transient, non-static member. Please mark as transient or provide accessors.
87	Use explicit scoping instead of the default package private level
88	Use explicit scoping instead of the default package private level
89	Found non-transient, non-static member. Please mark as transient or provide accessors.
89	Use explicit scoping instead of the default package private level
89	When instantiating a SimpleDateFormat object, specify a Locale
90	Found non-transient, non-static member. Please mark as transient or provide accessors.
90	Use explicit scoping instead of the default package private level
90	When instantiating a SimpleDateFormat object, specify a Locale
92	A method/constructor shouldn't explicitly throw java.lang.Exception
92	A method/constructor shouldn't explicitly throw java.lang.Exception
92	Document empty method
99	Avoid unnecessary constructors - the compiler will generate these for you
99	Document empty constructor
99	It is a good practice to call super() in a constructor
99	Avoid unnecessary constructors - the compiler will generate these for you
99	Document empty constructor
99	It is a good practice to call super() in a constructor
107	A method/constructor shouldn't explicitly throw java.lang.Exception
107	A method/constructor shouldn't explicitly throw java.lang.Exception
107	Method names should not start with capital letters
108	Local variable 'vParametros' could be declared final
109	Local variable 'vSesiones' could be declared final
111	The String literal "vObjControl" appears 10 times in this file; the first occurrence is on line 111
112	Avoid unused local variables such as 'vObjDatos'.
112	Found 'DU'-anomaly for variable 'vObjDatos' (lines '112'-'123').
112	Local variable 'vObjDatos' could be declared final
113	A method should have only one exit point, and that should be the last statement in the method
113	The String literal "reportes/menuReportes" appears 6 times in this file; the first occurrence is on line 113
116	Avoid printStackTrace(); use a logger call instead.
117	System.out.print is used
118	The String literal "mensaje" appears 10 times in this file; the first occurrence is on line 118
119	A method should have only one exit point, and that should be the last statement in the method
121	The String literal "¡No intenta ingresar sin permiso!" appears 5 times in this file; the first occurrence is on line 121
122	The String literal "principal/pantInicio" appears 5 times in this file; the first occurrence is on line 122
130	A method/constructor shouldn't explicitly throw java.lang.Exception
130	A method/constructor shouldn't explicitly throw java.lang.Exception
130	Method names should not start with capital letters
131	Local variable 'vParametros' could be declared final
132	Local variable 'vSesiones' could be declared final
142	A method should have only one exit point, and that should be the last statement in the method
145	Avoid printStackTrace(); use a logger call instead.
146	System.out.print is used
148	A method should have only one exit point, and that should be the last statement in the method
159	A method/constructor shouldn't explicitly throw java.lang.Exception
159	A method/constructor shouldn't explicitly throw java.lang.Exception


PMD
F:\Tomcat 6.0\webapps\parroquia\reports\pmd-report.vb.html
WEB-INF/src/model/manager/LibrosManagerImpl.java
5 Avoid unused imports such as 'model.domain'
10 The class 'LibrosManagerImpl' has a Cyclomatic Complexity of 3 (Highest = 10).
11 Use explicit scoping instead of the default package private level
13 Parameter 'sessionFactory' is not assigned and could be declared final
15 Parameter 'pObjUsuario' is not assigned and could be declared final
16 Local variable 'session' could be declared final
17 Local variable 'vIdLibros' could be declared final
18 Consider simply returning the value vs storing it in local variable 'vIdLibros'
21 Parameter 'pObjUsuario' is not assigned and could be declared final
22 Local variable 'session' could be declared final
26 Parameter 'pIdUsuario' is not assigned and could be declared final
26 Parameter 'pMas' is not assigned and could be declared final
27 Local variable 'session' could be declared final
28 Avoid instantiating Integer objects. Call Integer.valueOf() instead.
28 Local variable 'vObjUsuario' could be declared final
29 Consider simply returning the value vs storing it in local variable 'vObjUsuario'
34 Found 'DD'-anomaly for variable 'vObjLibro' (lines '34'-'37').
39 Avoid printStackTrace(); use a logger call instead.
45 The method 'getListGest' has a Cyclomatic Complexity of 10.
47 Found 'DD'-anomaly for variable 'vListLibros' (lines '47'-'66').
49 Avoid if (x != y) ..; else ..;
49 Use equals() to compare strings instead of '==' or '!='
50 Avoid using if...else statements without curly braces
50 Prefer StringBuffer over += for concatenating strings
52 Avoid using if...else statements without curly braces
52 Prefer StringBuffer over += for concatenating strings
54 Avoid using if statements without curly braces
54 Use equals() to compare strings instead of '==' or '!='
55 Prefer StringBuffer over += for concatenating strings
57 Avoid using if statements without curly braces

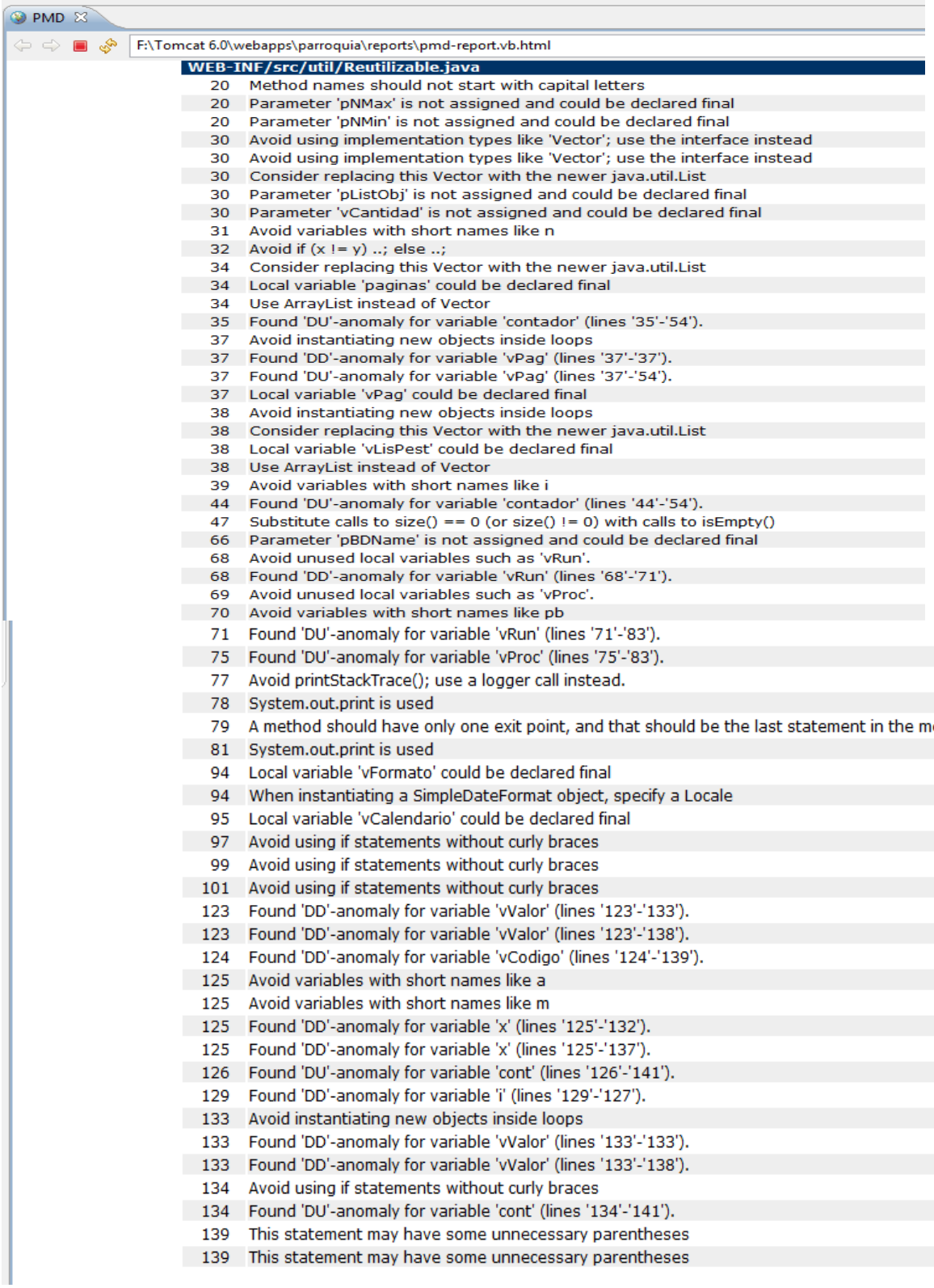

PMD

F:\Tomcat 6.0\webapps\parroquia\reports\pmd-report.vb.html

WEB-INF/src/model/manager/RegBautismoManagerImpl.java	
5	Avoid unused imports such as 'model.domain'
10	The class 'RegBautismoManagerImpl' has a Cyclomatic Complexity of 4 (Highest = 10).
11	Use explicit scoping instead of the default package private level
13	Parameter 'sessionFactory' is not assigned and could be declared final
15	Parameter 'pObjRegbautismo' is not assigned and could be declared final
16	Local variable 'session' could be declared final
17	Local variable 'vIdRegbautismo' could be declared final
19	Avoid using for statements without curly braces
25	Parameter 'pObjRegbautismo' is not assigned and could be declared final
26	Local variable 'session' could be declared final
30	Avoid using for statements without curly braces
33	Avoid printStackTrace(); use a logger call instead.
36	Parameter 'pIdRegbautismo' is not assigned and could be declared final
36	Parameter 'pMas' is not assigned and could be declared final
37	Local variable 'session' could be declared final
38	Avoid instantiating Integer objects. Call Integer.valueOf() instead.
38	Local variable 'vObjRegbautismo' could be declared final
39	Consider simply returning the value vs storing it in local variable 'vObjRegbautismo'
41	The method 'getListGest' has a Cyclomatic Complexity of 10.
43	Found 'DD'-anomaly for variable 'vListRegbautismo' (lines '43'-'59').
45	Avoid if (x != y) ..; else ..;
45	Use equals() to compare strings instead of '==' or '!='
46	Avoid using if...else statements without curly braces
46	Prefer StringBuffer over += for concatenating strings
48	Avoid using if...else statements without curly braces
48	Prefer StringBuffer over += for concatenating strings
50	Avoid using if statements without curly braces
50	Use equals() to compare strings instead of '==' or '!='
51	Prefer StringBuffer over += for concatenating strings
53	Avoid using if statements without curly braces



WEB-INF/src/model/manager/RegConfirmacionManagerImpl.java	
5	Avoid unused imports such as 'model.domain'
10	The class 'RegConfirmacionManagerImpl' has a Cyclomatic Complexity of 4 (Highest = 10).
11	Use explicit scoping instead of the default package private level
13	Parameter 'sessionFactory' is not assigned and could be declared final
15	Avoid excessively long variable names like pObjRegconfirmacion
15	Parameter 'pObjRegconfirmacion' is not assigned and could be declared final
16	Local variable 'session' could be declared final
17	Avoid excessively long variable names like vIdRegconfirmacion
17	Local variable 'vIdRegconfirmacion' could be declared final
19	Avoid using for statements without curly braces
25	Avoid excessively long variable names like pObjRegconfirmacion
25	Parameter 'pObjRegconfirmacion' is not assigned and could be declared final
26	Local variable 'session' could be declared final
29	Avoid using for statements without curly braces
34	Avoid excessively long variable names like pIdRegconfirmacion
34	Parameter 'pIdRegconfirmacion' is not assigned and could be declared final
34	Parameter 'pMas' is not assigned and could be declared final
35	Local variable 'session' could be declared final
36	Avoid excessively long variable names like vObjRegconfirmacion
36	Avoid instantiating Integer objects. Call Integer.valueOf() instead.
36	Local variable 'vObjRegconfirmacion' could be declared final
37	Consider simply returning the value vs storing it in local variable 'vObjRegconfirmacion'
40	The method 'getListGest' has a Cyclomatic Complexity of 10.
42	Found 'DD'-anomaly for variable 'vListConfirmaciones' (lines '42'-'58').
44	Avoid if (x != y) ..; else ..;
44	Use equals() to compare strings instead of '==' or '!='
45	Avoid using if...else statements without curly braces
45	Prefer StringBuffer over += for concatenating strings
47	Avoid using if...else statements without curly braces
47	Prefer StringBuffer over += for concatenating strings



WEB-INF/src/util/Reutilizable.java

20	Method names should not start with capital letters
20	Parameter 'pNMax' is not assigned and could be declared final
20	Parameter 'pNMin' is not assigned and could be declared final
30	Avoid using implementation types like 'Vector'; use the interface instead
30	Avoid using implementation types like 'Vector'; use the interface instead
30	Consider replacing this Vector with the newer java.util.List
30	Parameter 'pListObj' is not assigned and could be declared final
30	Parameter 'vCantidad' is not assigned and could be declared final
31	Avoid variables with short names like n
32	Avoid if (x != y) ..; else ..;
34	Consider replacing this Vector with the newer java.util.List
34	Local variable 'paginas' could be declared final
34	Use ArrayList instead of Vector
35	Found 'DU'-anomaly for variable 'contador' (lines '35'-'54').
37	Avoid instantiating new objects inside loops
37	Found 'DD'-anomaly for variable 'vPag' (lines '37'-'37').
37	Found 'DU'-anomaly for variable 'vPag' (lines '37'-'54').
37	Local variable 'vPag' could be declared final
38	Avoid instantiating new objects inside loops
38	Consider replacing this Vector with the newer java.util.List
38	Local variable 'vLisPest' could be declared final
38	Use ArrayList instead of Vector
39	Avoid variables with short names like i
44	Found 'DU'-anomaly for variable 'contador' (lines '44'-'54').
47	Substitute calls to size() == 0 (or size() != 0) with calls to isEmpty()
66	Parameter 'pBDName' is not assigned and could be declared final
68	Avoid unused local variables such as 'vRun'.
68	Found 'DD'-anomaly for variable 'vRun' (lines '68'-'71').
69	Avoid unused local variables such as 'vProc'.
70	Avoid variables with short names like pb
71	Found 'DU'-anomaly for variable 'vRun' (lines '71'-'83').
75	Found 'DU'-anomaly for variable 'vProc' (lines '75'-'83').
77	Avoid printStackTrace(); use a logger call instead.
78	System.out.print is used
79	A method should have only one exit point, and that should be the last statement in the method
81	System.out.print is used
94	Local variable 'vFormato' could be declared final
94	When instantiating a SimpleDateFormat object, specify a Locale
95	Local variable 'vCalendario' could be declared final
97	Avoid using if statements without curly braces
99	Avoid using if statements without curly braces
101	Avoid using if statements without curly braces
123	Found 'DD'-anomaly for variable 'vValor' (lines '123'-'133').
123	Found 'DD'-anomaly for variable 'vValor' (lines '123'-'138').
124	Found 'DD'-anomaly for variable 'vCodigo' (lines '124'-'139').
125	Avoid variables with short names like a
125	Avoid variables with short names like m
125	Found 'DD'-anomaly for variable 'x' (lines '125'-'132').
125	Found 'DD'-anomaly for variable 'x' (lines '125'-'137').
126	Found 'DU'-anomaly for variable 'cont' (lines '126'-'141').
129	Found 'DD'-anomaly for variable 'i' (lines '129'-'127').
133	Avoid instantiating new objects inside loops
133	Found 'DD'-anomaly for variable 'vValor' (lines '133'-'133').
133	Found 'DD'-anomaly for variable 'vValor' (lines '133'-'138').
134	Avoid using if statements without curly braces
134	Found 'DU'-anomaly for variable 'cont' (lines '134'-'141').
139	This statement may have some unnecessary parentheses
139	This statement may have some unnecessary parentheses

2.1.2.2.4 Evolución del Plan de Desarrollo del Software

El Plan de Desarrollo del Software se revisará semanalmente y se refinará antes del comienzo de cada iteración.

2.1.2.3 Organización el Proyecto

2.1.2.3.1 Participantes en el Proyecto

Director del Proyecto: Maritza Tatiana Vedia Mollo

2.1.2.3.2 Interfaces Externas

El entorno gráfico del sistema está pensado en los usuarios, se tomará en cuenta la comodidad y la facilidad de manejo.

El sistema cuenta con una interfaz diseñada en base a módulos debido a que el sistema es muy complejo, requiere bastante precisión, y sobre todo para otorgar una mejor comunicación en el proceso de desarrollo entre el personal responsable del mismo.

Todas las funcionalidades del sistema requieren de una validación de seguridad debido a que se maneja información de afiliados. La tediosa carga de datos que se realiza bajo una conexión de Intranet, debe ser totalmente transparente para el Usuario, para esto la interfaz debe simular a una Interfaz de un Sistema de Escritorio.

El rendimiento de las interfaces nombradas con anterioridad recae sobre todo en el lenguaje a utilizar y la versión del mismo.

El sistema contara de manera general con:

- Pantalla de Acceso.
- Pantalla de Menú Principal.

2.1.2.3.3 Roles y Responsabilidades

A continuación se describen las principales responsabilidades de cada uno de los puestos en el equipo de desarrollo durante las fases de Inicio y Elaboración, de acuerdo con los roles que desempeñan en RUP.

Puesto	Responsabilidad
Jefe de Proyecto	El jefe de proyecto asigna los recursos, gestiona las prioridades, coordina las interacciones con los clientes y usuarios, y mantiene al equipo del proyecto enfocado en los objetivos. El jefe de proyecto también establece un conjunto de prácticas que aseguran la integridad y calidad de los artefactos del proyecto. Gestión de riesgos. Planificación y control del proyecto.
Analista de Sistemas	Captura, especificación y validación de requisitos, interactuando con el cliente y los usuarios mediante entrevistas. Elaboración del Modelo de Análisis y Diseño.
Programador	Construcción de prototipos. Colaboración en la elaboración de las pruebas funcionales, modelo de datos y en las validaciones con el usuario.
Ingeniero de Software	Gestión de requisitos, gestión de configuración y cambios, elaboración del modelo de datos, preparación de las pruebas funcionales, elaboración de la documentación.

Tabla Nº38. Roles y Responsabilidades

2.1.2.4 Gestión del Proceso

2.1.2.4.1 Estimaciones del Proyecto

2.1.2.4.1.1 Introducción

El objetivo principal del plan de medidas es generar información que sea útil para gestionar el proyecto y tomar decisiones en base a datos objetivos.

2.1.2.4.1.2 Propósito

Se tomarán medidas sobre el proyecto de gestión **"SIGES"** Sistema Informático Gestión Sacramental que se está desarrollando.

2.1.2.4.1.3 Alcance

Este plan se limita al proyecto de gestión **"SIGES"** Sistema Informático Gestión Sacramental.

2.1.2.4.1.4 Visión General

Se van a identificar la información que se desea obtener y qué métricas miden esa información, describiendo con detalle algunas de ellas.

2.1.2.4.2 Objetivos y sub-objetivos de la gestión

Se deben identificar las necesidades de información para el proyecto de gestión "SIGES" Sistema Informático Gestión Sacramental. Varias de estas medidas son necesarias para gestionar riesgos, otras para gestionar el proyecto midiendo tiempos y recursos, otras para comprobar los costes y otras para comprobar la calidad, por ejemplo del diseño realizado antes de comenzar la implementación. Puede haber más casos en los que se necesite hacer medidas, pero como ejemplo, en este documento sólo veremos algunas métricas representativas.

2.1.2.4.3 Métricas

Categoría de información	**Concepto medible**	**Métrica**
Tiempo y progreso	Tiempo y esfuerzo de desarrollo	COCOMO
Tamaño del producto	Tamaño físico	Número de líneas de código
	Tamaño Funcional	Puntos de función Número de casos de uso

Tabla Nº39. Métricas

2.1.2.4.4 Descripción de las métricas

Nombre	COCOMO
Definición	Métrica de tiempo y esfuerzo de desarrollo
Objetivo	Ayudar a realizar la planificación temporal y de recursos, así como a realizar la gestión del proyecto.
Proceso de Análisis	Aplicar tablas y formulas del método COCOMO para obtener estimaciones de esfuerzo y tiempo.

Tabla Nº40. Cocomo

Nombre	Basado en Casos de Uso
Definición	Métrica de tamaño, Esfuerzo y Funcionalidad
Objetivo	Medir el tamaño de Producto y Funcional de la aplicación y Estimar Esfuerzo
Proceso de Análisis	Aplicar tablas con valores y hacer cálculos siguiendo el método de análisis de estimación basada en Casos de Uso.

Tabla Nº41. Basado en casos de uso

Nombre	Puntos de Función
Definición	Métrica de tamaño y complejidad
Objetivo	Medir el tamaño y complejidad de la aplicación
Proceso de Análisis	Aplicar tablas con valores y hacer cálculos siguiendo el método de análisis de puntos de función.

Tabla Nº42. Puntos de función

2.1.2.4.5 Utilización de las Métricas

2.1.2.4.5.1 COCOMO

El tamaño del software varía de unos pocos miles de líneas (tamaño pequeño) a unas decenas de miles de líneas (medio).

Se utilizan dos ecuaciones para determinar el esfuerzo de personal y el tiempo de desarrollo. El coste es:

$K_m = 2.4\ S_k^{1.05}$

Dónde:

K_m se expresa en personas-mes.

S_k es el tamaño expresado en miles de líneas de código fuente.

El tiempo de desarrollo se da por:

$t_d = 2.5\ K_m^{0.38}$

Donde K_m se obtiene de la ecuación anterior y t_d es el tiempo de desarrollo en meses.

El Nº de Personas Medio se da por:

$P_e = K_m / t_d$

Donde K_m se obtiene de la ecuación anterior y t_d se obtiene de la ecuación anterior y Pe es el número de personas promedio.

Km=(2.4)(15)^1.05 = 41.22 personas-mes

Td= 2.5 (41.22)^0.38 = 10.27 mes

Pe=41.22 / 10.27 = 4.01 personas

Lo cual se estima que el proyecto se llevara a cabo en 10 meses con 4 personas, Por tanto, el proyecto se realizará en 250 días sin considerar los días domingos y feriados. En este proyecto sin embargo lo desarrolla solo una persona que es el Jefe de Proyecto que tiene que realizar la función de las cuatro personas.

2.1.2.4.5.2 Basado en Casos de Uso

Se cuenta con los puntos de Casos de Uso ajustados y se calcula que para cada punto de casos de uso se requieren 2 horas-hombre. Por lo tanto el esfuerzo se obtiene aplicando la siguiente fórmula:

UCP = Punto de Caso de Uso

E = UCP * 2 = horas-hombre

E = 132 * 2 = 264 horas-hombre

Usar la contabilización de los Factores ambientales para ajustar las horas-hombre que se requieren por punto de caso de uso, este valor de esfuerzo según la siguiente tabla representa a la actividad de programación, por tanto se debe realizar un nuevo cálculo agregando las actividades restantes relacionadas con el desarrollo del software.

Por lo tanto debería calcularse el esfuerzo de la siguiente manera:

Actividad	Porcentaje	Horas-hombre
Análisis	10%	10%*264=26.4
Diseño	20%	20%*264=52.8
Programación	40%	40%*264=105.6
Pruebas	15%	15%*264=39.6
Sobrecarga (otras actividades)	15%	15%*264=39.6
TOTAL de esfuerzo	100%	∑ = 264

Tabla Nº43. Calculo de esfuerzo

El tiempo es de 264 horas – hombre por cada Caso de Uso.

En la cual Si realizamos el siguiente calculo

264 horas-hombre / 24 horas = 11 días por cada Caso de Uso

Es una Estimación que no tiene mucha relevancia por considerar solo los requisitos del cliente capturados en los casos de uso.

2.1.2.4.5.3 Puntos de Función

Los puntos de función que obtienen utilizando una función empírica basando en medidas cuantitativas del dominio de información del software y valoraciones subjetivos de la complejidad del software.

Los puntos de función se calculan mediante la siguiente fórmula:

$$PF=CT*(0.65+0.1*\sum Fi)$$

Dónde:

PF= Punto de función

CT= Cuenta Total

Fi= Cuenta de los valores de ajuste de complejidad

Los Puntos de Función se derivan de medidas Directas del dominio de la información

Calculo de la cuenta total:

Parámetro	**Cuenta**	**Factor de Ponderación**			**Subtotal**
		Simple	**Medio**	**Complejo**	
Número de entradas de usuario	15	3	4	6	60
Número de salidas de usuario	20	4	5	7	140
Número de peticiones de usuario	15	3	4	6	90
Numero de archivos	3	7	10	15	2
Numero de interfaces externas	2	5	7	10	10
Cuenta Total					351

Tabla Nº44. Factor de ponderación

Valores de Ajuste de Complejidad.

0	1	2	3	4	5
Sin influencia	Incidental	Moderado	Medio	Significativo	Esencial

Tabla Nº45. Valores de ajuste de complejidad

Nº	Parámetros a Evaluar	Valor
1	Comunicación de datos	5
2	Funciones de procesamiento distribuidos	2
3	Objetivos de Performance	4
4	Ejecución del sistema en un entorno operativo utilitario	4
5	Transacciones de datos sobre múltiples entradas	1
6	Entrada de datos	5
7	Copia de seguridad y recuperación de datos fiable	5
8	Actualización de archivos en forma interactiva	5
9	Complejidad de procesamiento interno	3
10	Reusabilidad de código	4
11	Facilidad de instalación	2
12	Facilidad operacional	5
13	Soporte de múltiples instalaciones	4
14	Facilidad de cambio y manejo	5
	S-Total	54

Tabla Nº46. Parámetros a evaluar

PF = Cuenta-Total * [0.65 + (0.01 * S-Total)]

PF = 351 * [0.65 + (0.01 * 54)]

PF = 417.69

PF = 418

La siguiente tabla proporciona estimaciones del número de líneas de código que se necesitan para construir un punto de función en varios lenguajes de programación:

Lenguaje	**LDC/PF**
Ensamblador	320
C	128
Java	120
Fortran	105
Ada	70
4GL	20
Lenguajes de Iconos	6

Tabla Nº47. Estimación de cantidad líneas de código en función al lenguaje de programación

En base a esta tabla se pueden establecer las líneas de código (LDC) que sería la estimación para nuestro proyecto.

El lenguaje de programación que desarrollamos para nuestro proyecto es Java entonces tendríamos unas 120 LDC por PF:

LDC = 120 * 418

LDC = 50160

Para simplificar el proceso de estimación y utilizar una forma más común para su modelo de estimación, Putman y Myers sugieren un conjunto de ecuaciones obtenidas de la ecuación del software:

$t = 8.14 (LDC / P)^{0.43}$ (1)

$E = 180 B t^3$ (2)

Donde

E = esfuerzo en personas – mes

t = duración del proyecto en meses a años.

B = factor especial de destrezas. Para programas mayores a 60 KLDC, B > 0.39.

P = parámetro de productividad. Para aplicaciones comerciales de sistemas, P > 28000

Aplicando las ecuaciones (1) y (2) a nuestro proyecto obtenemos:

Dónde: B = 1.03 y P = 28000 para nuestro caso.

$t = 8.14 (50160 / 28000)^{0.43}$

t = 10,46 meses.

El tiempo estimado el proyecto es de 10 meses.

$E = 180 * 1.03 * (0.82)^3$

E = 102.22 personas – mes.

Entonces 102.22 personas – mes. / 10 mes.= 10.22 personas.

El esfuerzo estimado el proyecto según el análisis de Punto de Función es de 10 personas por mes. Pero vale la pena mencionar que solo una persona desarrolla el sistema.

2.1.2.4.6 Plan del Proyecto

En esta sección se presenta la organización en fases e iteraciones y el calendario del proyecto.

2.1.2.4.6.1 Plan de las Fases

El desarrollo se llevará a cabo en base a fases con una o más iteraciones en cada una de ellas. La siguiente tabla muestra una la distribución de tiempos y el número de iteraciones de cada fase (para las fases de Construcción y Transición es sólo una aproximación muy preliminar).

Fase	Nro. Iteraciones	Duración
Fase de Inicio	1	50
Fase de Elaboración	1	64
Fase de Construcción	1	61
Fase de Transición	1	53

Tabla Nº48. Plan de las faces

Los hitos que marcan el final de cada fase se describen en la siguiente tabla.

Descripción	Hito
Fase de Inicio	En esta fase desarrollarán los requisitos del producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto Visión. Los principales casos de uso serán identificados y se hará un refinamiento del Plan de Desarrollo del Proyecto. La aceptación del cliente /usuario del artefacto Visión y el Plan de Desarrollo marcan el final de esta fase.
Fase de Elaboración	En esta fase se analizan los requisitos y se desarrolla un prototipo de arquitectura (incluyendo las partes más relevantes y / o críticas del sistema). Al final de esta fase, todos los casos de uso correspondientes a requisitos que

	serán implementados en la primera reléase de la fase de Construcción deben estar analizados y diseñados (en el Modelo de Análisis / Diseño). La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase. En nuestro caso particular, por no incluirse las fases siguientes, la revisión y entrega de todos los artefactos hasta este punto de desarrollo también se incluye como hito. La primera iteración tendrá como objetivo la identificación y especificación de los principales casos de uso, así como su realización preliminar en el Modelo de Análisis / Diseño, también permitirá hacer una revisión general del estado de los artefactos hasta este punto y ajustar si es necesario la planificación para asegurar el cumplimiento de los objetivos. Ambas iteraciones tendrán una duración de una semana.
Fase de Construcción	Durante la fase de construcción se terminan de analizar y diseñar todos los casos de uso, refinando el Modelo de Análisis / Diseño. El producto se construye en base a 2 iteraciones, cada una produciendo una reléase a la cual se le aplican las pruebas y se valida con el cliente / usuario. Se comienza la elaboración de material de apoyo al usuario. El hito que marca el fin de esta fase es la versión de la reléase 2.0, con la capacidad operacional parcial del producto que se haya considerado como crítica, lista para ser entregada a los usuarios para pruebas beta.
Fase de Transición	En esta fase se prepararán dos raleases para distribución, asegurando una implantación y cambio del sistema previo

	de manera adecuada, incluyendo el entrenamiento de los usuarios. El hito que marca el fin de esta fase incluye, la entrega de toda la documentación del proyecto con los manuales de instalación y todo el material de apoyo al usuario, la finalización del entrenamiento de los usuarios y el empaquetamiento del producto.

Tabla Nº49. Hitos

2.1.2.4.6.2 Calendario del Proyecto

A continuación se presenta un calendario de las principales tareas del proyecto incluyendo sólo las fases de Inicio y Elaboración. Como se ha comentado, el proceso iterativo e incremental de RUP está caracterizado por la realización en paralelo de todas las disciplinas de desarrollo a lo largo del proyecto, con lo cual la mayoría de los artefactos son generados muy tempranamente en el proyecto pero van desarrollándose en mayor o menor grado de acuerdo a la fase e iteración del proyecto. La siguiente figura ilustra este enfoque, en ella lo ensombrecido marca el énfasis de cada disciplina (workflow) en un momento determinado del desarrollo.

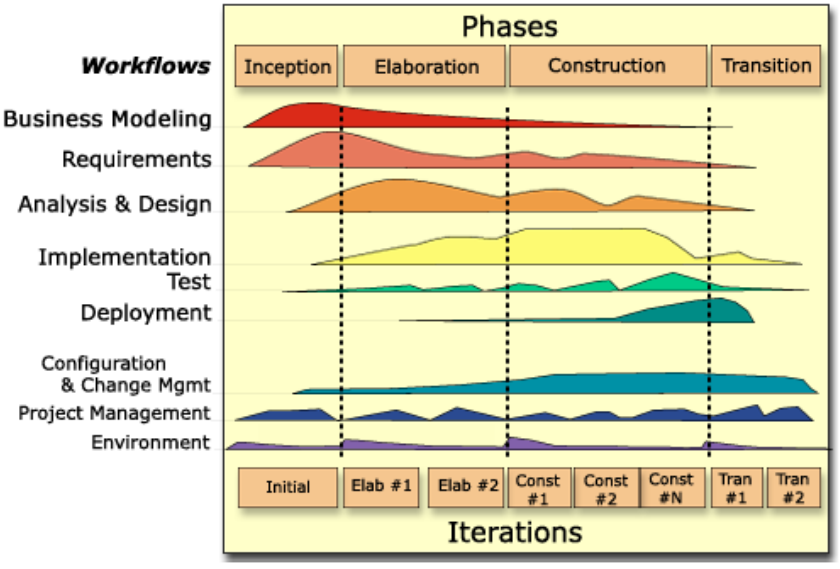


Figura Nº146. Calendario del proyecto

Para este proyecto se ha establecido el siguiente calendario. La fecha de aprobación indica cuándo el artefacto en cuestión tiene un estado de completitud suficiente para someterse a revisión y aprobación, pero esto no quita la posibilidad de su posterior refinamiento y cambios.

Disciplinas / Artefactos generados o modificados durante la Fase de Inicio	**Comienzo**	**Aprobación**
Modelado del Negocio		
Modelo de Casos de Uso del Negocio y Modelo de Objetos del Negocio		
Requisitos		
Glosario		
Visión		
Modelo de Casos de Uso		siguiente fase
Especificación de Casos de Uso		siguiente fase
Especificaciones Adicionales		siguiente fase
Análisis/Diseño		
Modelo de Análisis/Diseño		siguiente fase
Modelo de Datos		siguiente fase
Implementación		
Prototipos de Interfaces de Usuario		siguiente fase
Modelo de Implementación		siguiente fase

Pruebas		
Casos de Pruebas Funcionales		siguiente fase
Despliegue		
Modelo de Despliegue		siguiente fase
Gestión de Cambios y Configuración	Durante todo el proyecto	
Gestión del proyecto		
Plan de Desarrollo del Software en su versión 1.0 y planes de las Iteraciones		
Ambiente	Durante todo el proyecto	

Tabla Nº50. Disciplina / Artefactos fase de inicio

Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración	**Comienzo**	**Aprobación**
Modelado del Negocio		
Modelo de Casos de Uso del Negocio y Modelo de Objetos del Negocio		Aprobado
Requisitos		
Glosario		Aprobado
Visión		Aprobado
Modelo de Casos de Uso		
Especificación de Casos de Uso		
Especificaciones Adicionales		

Análisis / Diseño		
Modelo de Análisis / Diseño		Revisar en cada iteración
Modelo de Datos		Revisar en cada iteración
Implementación		
Prototipos de Interfaces de Usuario		Revisar en cada iteración
Modelo de Implementación		Revisar en cada iteración
Pruebas		
Casos de Pruebas Funcionales		Revisar en cada iteración
Despliegue		
Modelo de Despliegue		Revisar en cada iteración
Gestión de Cambios y Configuración	Durante todo el proyecto	
Gestión del proyecto		
Plan de Desarrollo del Software en su versión 2.0 y planes de las Iteraciones		Revisar en cada iteración
Ambiente	Durante todo el proyecto	

Tabla Nº51. Disciplina / Artefactos fase de elaboración

2.1.2.4.7 Seguimiento y Control del Proyecto

2.1.2.4.7.1 Gestión de Requisitos

Ver Anexo: Especificación de Requisitos de Software Norma IEEE830

2.1.2.4.7.2 Control de Plazos

Fase	Nº de Iteración	Inicio	Fin
Inicio	Primera	14-04-2013	31-05-2013
Elaboración	Primera	01-06-2013	03-08-2013
Construcción	Primera	04-08-2013	04-10-2013
Transición	Primera	05-10-2013	29-11-2013

Tabla Nº52. Control de plazos

2.1.2.4.7.3 Control de Calidad

2.1.2.4.7.3.1 Introducción

El control de calidad es un modelo planeado y sistemático de todas las acciones necesarias para proporcionar la confianza de que el artículo o producto se ajuste a los requisitos técnicos establecidos (IEE83).

La preparación de un plan de control de calidad del software para cada proyecto de software es una de las principales responsabilidades del grupo de control de calidad del software.

El control de calidad realizará las siguientes funciones:

Durante el análisis y diseño, se presentaran un plan de verificación del software y un plan de prueba de aceptación. El plan de verificación describe los métodos que se ocuparan para revisar que los documentos de diseño satisfagan los requisitos, y que el código fuente sea consistente con las especificaciones de requisitos y con la documentación del diseño.

El plan de prueba del código fuente es un componente importante del plan de verificación del software.

El plan de prueba de aceptación incluye casos de prueba, resultados esperados y capacidades demostradas por cada caso de prueba. A menudo, el personal de control de calidad trabajara con el cliente para desarrollar un solo plan de prueba de aceptación. En otros casos el cliente desarrollara un plan de prueba de aceptación independiente del plan de control de calidad. De cualquier forma, el personal de control de calidad debe desarrollar un plan de prueba de aceptación doméstico.

Al terminar los planes de verificación y de aceptación se realizara una revisión de verificación del software para evaluar cuan adecuados son los planes.

Durante la evolución del producto, se realizaran Auditorias en el proceso para verificar que los productos de trabajo sean consistentes y estén completos.

Los elementos que sufrirán auditoria por consistencia incluyen especificaciones de interfaces para hardware, software y personas; diseño interno contra especificaciones funcionales; código fuente contra documentación. En la práctica, solo ciertas porciones criticas del sistema pueden someterse a auditorias intensivas.

Antes de la entrega del sistema, se realizan una auditoria funcional y una auditoria física.

La primera reconfirma el cumplimiento de todos los requisitos. La auditoría física verifica que el código fuente y todos los documentos asociados estén completos, sean consistentes tanto internamente, como uno con otro, y que estén listos para enviarse. El resumen de verificación del software se repara para describir los resultados de todas las revisiones, auditorias y pruebas efectuadas por el personal de control de calidad, a través del ciclo de desarrollo.

Dicho personal, a veces se encarga de los acuerdos para los recorridos, inspecciones, y revisiones de logros principales. Además, el personal de control de calidad conduce el proyecto póstumo, escribe es documento del legado del proyecto, y proporciona una custodia a largo plazo de los registros del proyecto.

El grupo de control de calidad trabajara con el grupo de desarrollo para obtener el plan de pruebas y el código fuente, que especifica los objetivos de las pruebas, los criterios para la terminación de las pruebas, el plan de integración del sistema, los

métodos que se usaran en módulos particulares, además, entradas de prueba particulares y resultados esperados.

El plan de pruebas de código fuente tiene cuatro tipos de pruebas: pruebas de función, de desempeño, de tensión, y estructuradas. Las dos primeras se basan en las especificaciones de requisitos y se diseñaron para demostrar que el sistema satisface sus requisitos los que a su vez se deben redactar en términos cuantificables y que se puedan probar.

Los casos de prueba funcional especifican condiciones operativas comunes valores de entradas comunes y resultados esperados comunes, también prueban el comportamiento dentro, sobre, y más allá de las fronteras funcionales.

Las pruebas de desempeño se proyectan para verificar el tiempo de respuesta, rendimiento, la utilización de memorias primarias y secundarias y las tasas de tráfico en los canales de datos y los enlaces de comunicación.

Las pruebas de tensión se diseñan para sobrecargar un sistema de varias maneras.

Las pruebas de estructura se relacionan con el examen de la lógica interna de procesamiento de un sistema de software. Las rutinas particulares llamadas y las rutas lógicas recorridas a lo largo de las rutinas son los objetos importantes.

2.1.2.4.7.3.2 Propósito

- Detectar problemas.
- Delimitar el área problemática.
- Estimar factores que probablemente provoquen el problema.
- Determinar si el efecto tomado como problema es verdadero o no.
- Prevenir errores debido a omisión, rapidez o descuido.
- Confirmar los efectos de mejora.
- Detectar desfases.
- Realizar pruebas en cada versión.

2.1.2.4.7.3.3 Objetivos

- Aumentar la satisfacción del Cliente.
- Equilibrar el esfuerzo en múltiples demandas.
- Obtener el mejor producto.
- Disfrutar de una ventaja competitiva.
- Disponer de métricas objetivas de valoración.
- Ahorrar tiempo y dinero.

2.1.2.4.7.3.4 Resumen de las actividades de garantía de calidad

Listado de las actividades de garantía de la calidad que se llevarán a cabo durante el proyecto.

2.1.2.4.7.3.4.1 Seguimiento de la administración del SQA

ID	Propósito	Sub Actividad	Detalle / Indicación
1	Criterios de Entrada	- Ninguna.	Plan de Desarrollo y proceso de Software y Personal.
2	Revisión	- Examinar estructura gerencial de la organización. - Identificar tareas de cada integrante de la gerencia. - Definir responsabilidades a cada integrante de la gerencia	Verificar consistencia de la estructura organizacional con las responsabilidades asignadas en Plan de desarrollo de Software.
3	Criterios de Salida	- Estructura de la administración revisada.	Estructura organizacional de la gerencia óptima para el proyecto.

Tabla Nº53. Seguimiento de la administración del SQA

2.1.2.4.7.3.4.2 Seguimiento de la Documentación

ID	Propósito	Sub Actividad	Detalle / Indicación
1	Criterios de Entrada	- Ninguna.	Plan de Desarrollo de Software.
2	Revisión	- Revisión y análisis del plan de documentación. - Buscar discrepancias. - Discutir discrepancias con el gerente de proyecto.	Reportar discrepancias según documento presentado por el docente y estándares.
3	Criterios de Salida	- Documentación revisada.	Documentos de acuerdo a Estándar, y sin discrepancias.

Tabla Nº54. Seguimiento de la documentación

2.1.2.4.7.3.4.3 Seguimiento de la adherencia a los Estándares

ID	Propósito	Sub Actividad	Detalle / Indicación
1	Criterios de Entrada	- Ninguna.	Documentos, plan de desarrollo de Software.
2	Documentación	- Monitorear adherencias de los documentos a los estándares.	Chequear documento desarrollado basado en la metodología RUP.
3	Diseño	- Monitorear adherencias del diseño a los estándares.	Chequear documento UML del sistema.
4	Codificación	- Monitorear adherencias de la codificación a los estándares.	Revisar de acuerdo a Patrones de Diseño MVC.
5	Métricas	- Revisar la métrica definida.	Revisar de acuerdo al estándar Puntos de Función y otros.
6	Criterios de Salida	- Proceso de Documentación revisado. - Proceso de Diseño revisado.	Discrepancias reportadas y solucionadas. Documentos de acuerdo a estándares.

Tabla Nº55. Seguimiento de la adherencia a los Estándares

2.1.2.4.7.4 Gestión de Riesgos

2.1.2.4.7.4.1 Relacionados con el tamaño del producto.

- Es posible no poder llegar a terminar todos los componentes del proyecto debido a que estos son muchos o grandes.

2.1.2.4.7.4.2 Relacionados con el impacto en la organización.

- No entregar el proyecto en el tiempo estimado.
- El posible no uso del software debido a los requerimientos de equipos.
- Poco uso del software.
- Gran cantidad de documentación a entregar al Cliente.
- Los límites legales y gubernamentales en cuanto al desarrollo y funciones del proyecto.
- Posibles errores en el producto y costos asociados.

2.1.2.4.7.4.3 Relacionados con el tipo de Cliente.

- Si se tiene la disponibilidad de tiempo para la especificación formal de requerimientos.
- Si están dispuestos a participar de las pruebas o revisiones.
- Si se relacionará de forma ágil con el grupo de desarrollo.

2.1.2.4.7.4.4 Relacionados con la definición del proceso de producción.

- El software a ser utilizado en el control del desarrollo del proyecto.
- Las herramientas a ser utilizadas en el análisis y diseño.

2.1.2.4.7.4.5 Relacionados con el entorno de desarrollo.

- Si hay herramientas de gestión de proyectos.
- Hay herramientas de prueba apropiadas.
- Generadores de código para la aplicación.

2.1.2.4.7.4.6 Relacionados con la tecnología.

- Es una nueva tecnología.
- El hardware con el que debe interactuar es nuevo o cumple las expectativas.
- La base de datos a ser utilizada ha sido probada y tiene la funcionalidad y rendimiento.
- Las interfaces son especializadas.
- La necesidad de nuevos componentes.

2.1.2.4.7.4.7 Relacionados con la experiencia y tamaño del equipo.

- Es el mejor personal disponible.
- Los miembros tienen las técnicas apropiadas.
- Hay suficiente gente disponible.
- El personal está comprometido a lo largo de desarrollo del proyecto.
- Tiene el personal las expectativas correctas del trabajo.

2.1.2.4.7.5 Gestión de Configuración

Se llevara a cabo una gestión de configuración para llevar un registro de los artefactos generados y sus versiones. También se incluirían la gestión de las solicitudes de cambio y de las modificaciones que estas produzcan, informando y publicando dichos cambios a todos los participantes del proyecto.

2.2 Componente 2: Capacitación del Secretario y administrador del Sistema de la Parroquia San Martin de Porres.

2.2.1 Introducción

De acuerdo al modo de vida actual del mundo, el término Capacitación y Sistemas de Información están modificando la forma de trabajo de las empresas, los sistemas de Información ayudan a acelerar los procesos, por tanto, las organizaciones que los implantan logran ventajas competitivas al adoptarlos en sus funciones. Pero si el personal no está debidamente capacitado la producción de las organizaciones será deficiente y en mucho casos incompleto.

La capacitación se refiere a las metodologías que se usan para proporcionar a las personas dentro de la organización, las habilidades que necesitan para realizar su trabajo de una manera más eficiente, esto contempla desde pequeños cursos que le permitan al usuario entender el funcionamiento básico del sistema nuevo, hasta capacitación más profunda y avanzadas a bases de prácticas y material didáctico como libros (Tutoriales) y otros.

La capacitación es un proceso que lleva a la mejora continua de la producción y con esto implantar nuevas formas de trabajo, como en este caso el manejo adecuado del Sistema Informático desarrollado y Conocimientos en el área de las TIC’s.

2.2.2 Aspectos Pedagógicos de la Capacitación

2.2.2.1 Definición de Capacitación

La capacitación es un proceso educacional de carácter estratégico aplicado de manera organizada y sistémica, mediante el cual los colaboradores adquieren o desarrollan conocimientos y habilidades específicas relativas al trabajo, y modifica sus actitudes frente a los quehaceres de la organización, el puesto o el ambiente laboral.

2.2.2.2 Objetivos de la Capacitación

- Preparar a los colaboradores para la ejecución de las diversas tareas y responsabilidades de la parroquia.
- Manipular el sistema agregar, modificar datos

- Manipular información de los feligreses
- Tener una información ordenada.

2.2.3 Metodología

La metodología utilizada para esta capacitación i/o taller se basa en el modelo de experiencia "Aprender Haciendo", logrando de esta manera la motivación de ser constructores de su propio conocimiento. Involucra trabajos de completa actividad en Talleres de construcción de contenidos, Laboratorios, Guías, donde se incentiva el trabajo creativo y práctico, experimentándose con procesos o fenómenos a partir de ideas o propuestas teóricas previamente formuladas.

Es una metodología busca desarrollar su sensibilidad frente a problemas reales, estudiar alternativas de solución y evaluar sus implicancias en conjunto con la utilización de tecnologías.

2.2.3.1 Estrategias Didácticas

2.2.3.1.1 Descubriendo la Tecnología

2.2.3.1.1.1 Tipo: Sesiones Individuales y Grupales.

2.2.3.1.1.1.1 Descripción:

Se desarrolla Actividades Guiadas donde los capacitadores hacen uso del material guía y van descubriendo por sí solos las potencialidades de los programas y herramientas y que luego ellos mismos documentan haciendo énfasis en cuáles de las experiencias descubiertas son de principal apoyo para su práctica pedagógica.

2.2.3.2 Aspectos Prácticos

2.2.3.2.1 Actividad teórico- práctica:

Propicia la modalidad del curso teórico con una actividad de la práctica en relación a la temática de estudio. Lo teórico y lo práctico se dan simultáneamente en forma conjunta e interrelacionada y es dirigida directamente por el capacitador participante.

2.2.3.3 Mecanismos de Trabajo durante la capacitación:

2.2.3.3.1 Fase Inicial:

Análisis de la Realidad Social de la Región: donde los Facilitadores intercambian diferentes criterios con los docentes logrando tener una visión clara de las necesidades pedagógicas de los participantes enfocando la capacitación a la Presentación de Temas Individuales.

2.2.3.3.2 Fase de Capacitación:

Investigación y documentación: se promueve en todas las estrategias didácticas aplicadas a la investigación continua y descubrimiento de la tecnología reconociendo su valor e identificando su potencial para ser utilizados en su preparación de material, todo esto desarrollado por el mismos capacitador.

2.2.3.3.3 Fase de Finalización:

Medición de aprovechamiento: donde los Facilitadores realizan las evaluaciones de trabajos en forma individualizada como a su vez de término de grupo capacitado.

2.2.3.4 Modalidades Prácticas de Capacitación

El Plan de capacitación podrá usar otras modalidades que se incorporen según las necesidades:

2.2.3.4.1 Inducción

Su objeto es la ambientación inicial al medio social y físico donde trabaja y se programa para todo colaborador nuevo.

2.2.3.4.2 Capacitación en el Puesto de Trabajo

Se desarrollara en el propio puesto de trabajo y mientras el interesado ejecuta sus tareas.

2.2.3.5 Medios de Capacitación

Dentro de los más principales tenemos:

2.2.3.5.1 Manuales de Capacitación

Manuales de capacitación u otros impresos, diagramas que permiten la exposición repetida, es útil aplicación de secuencias largas o procedimientos complicados que no pueden retenerse en una sola presentación. Pueden combinarse con conferencias y prácticas de tareas reales.

2.2.4 Determinación de la Estructura y la Metodología del Componente Capacitación

Para determinar la estructura y la metodología de la capacitación, existe una cantidad de cuestiones interrelacionadas que requiere solución como pueden ser:

- Cuál será la mejor estructura para el programa de capacitación y cuándo se deberá instrumentar. (Momento Adecuado para Impartir la Capacitación);
- Qué se necesitará para llevar a cabo una presentación exitosa. (Materiales de Referencia con Propósitos de Capacitación);
- Qué temas deberán cubrir las sesiones de capacitación y cómo deberán organizarse (Estilo para Impartir la Capacitación);
- Qué instalaciones y recursos adicionales se necesitarán. (Entorno para la Capacitación);
- Cómo se medirán los resultados de la capacitación. (Evaluación del Conocimiento).

2.2.5 Ingeniería de Requerimientos

2.2.5.1 Propuesta de Solución para el curso de Capacitación.

2.2.5.1.1 Introducción

La efectividad de la capacitación puede ser considerablemente influenciada por el estilo de impartición y los métodos de presentación empleados para este fin.

El entorno físico en el cual se llevará a cabo la capacitación tiene una importante repercusión en la eficacia de la misma. Es por lo mismo que se llevará a cabo en las Instalaciones de la Parroquia San Martin de Porres ya que el grado de respuesta a la capacitación puede disminuirse significativamente si las necesidades básicas no se

han organizado satisfactoriamente.

Las personas que se motivan y desean ser mejores a través del proceso de capacitación requieren de una herramienta que los apoye a lograr su nueva actitud en la organización con procedimientos rápidos y fáciles.

Con lo expuesto anteriormente se realizaran una sesión de capacitación,

La sesión, del 02 de Diciembre al 06 de Diciembre, será la capacitación en el manejo del sistema informático desarrollado denominado “Gestión Sacramental” Sistema Informático Gestión de Sacramental, en la que asistirá el secretario de la Parroquia San Martin de Porres y encargado del Administración del Sistema.

2.2.5.1.2 Objetivos de la Capacitación

En cuanto al objetivo, éste apunta a: Capacitar al Secretario de la Parroquia, Encargado de Sistemas en el manejo correcto del Sistema Web para el Mejoramiento en los Procesos de Registros de Datos de personas que reciben un Sacramento en la Parroquia San Martin de Porres de Tarija.

2.2.5.1.3 Medios A utilizar

2.2.5.1.3.1 Aspectos Técnicos

- 1 Computador con el Sistema Desarrollado.
- Diapositivas de Exposición hechas en PowerPoint.
- Contenido en Documento Impreso.

2.2.5.1.3.2 Aspectos Logísticos

- Refrigerios

2.2.5.1.4 Estructura del Curso de Capacitación

2.2.5.1.4.1 Capacitación en el uso del Sistema Informático Desarrollado

Fecha a Realizar	Módulos de Aprendizaje del Curso	Horas C/Módulo	Nº de Sesiones	Dirigido A
06/12/2013	Módulo 1. Presentación y Manejo del Sistema Informático Gestión Sacramental.	1 hrs.	1	Encargado de Administración del Sistema y Secretario de la Parroquia San Martin de Porres
07/12/2013	Capacitación en el Manejo del Sistema Informático Gestión Sacramental	1 hrs.	1	Encargado de Administración del Sistema y Secretario de la Parroquia San Martin de Porres
08/11/2013	Capacitación en el Manejo del Sistema Informático Gestión Sacramental	1 hrs.	1	Encargado de Administración del Sistema y Secretario de la Parroquia San Martin de Porres
09/11/2013	Capacitación en el Manejo del Sistema Informático Gestión Sacramental	1 hrs.	1	Encargado de Administración del Sistema y Secretario de la Parroquia San Martin de Porres
10/11/2013	Capacitación en el Manejo del Sistema Informático Gestión Sacramental	1 hrs.	1	Encargado de Administración del Sistema y Secretario de la Parroquia San Martin de Porres
	Totales	**5 Horas**	**5 Sesiones**	

Tabla Nº56. Capacitación en el uso del Sistema Informático Desarrollado

2.2.5.1.5 El Contenido de la Capacitación se Realizó de Acuerdo al Siguiente Cronograma:

Fecha	Hora	Actividad	**Expositor**
06 de Diciembre del 2013	18:00 – 19:00	Clase 1: Ingresar al Sistema y funcionalidades según el tipo de Usuario.	Maritza Tatiana Vedia Mollo
07 de Diciembre del 2013	18:00 – 19:00	Clase 2: Gestionar Usuarios, Roles, parroquias, Sacerdotes.	Maritza Tatiana Vedia Mollo
08 de Diciembre del 2013	18:00 – 19:00	Clase 3: Gestionar Feligreses.	Maritza Tatiana Vedia Mollo
09 de Diciembre del 2013	18:00 – 19:00	Clase 4: Gestionar Sacramentos.	Maritza Tatiana Vedia Mollo
10 de Diciembre del 2013	18:00 – 19:00	Clase 5: Gestionar Reportes	Maritza Tatiana Vedia Mollo

Tabla Nº57. Cronograma de Contenido de capacitación.

2.2.5.1.6 Medios de Verificación para el Componente Capacitación

Fotos de la Capacitación

III. Capítulo 3: Conclusiones y Recomendaciones

3.1 Conclusiones

Luego de haber concluido el presente proyecto denominado "**Mejorar la gestión de la Información Sacramental de la parroquia San Martin de Porres"** se llegó a las siguientes conclusiones.

- En relación al primer componente se logró desarrollar un sistema que va a mejorar la gestión de la información de los sacramentos recibidos por un feligrés.

  Con la Metodología RUP se realizó el análisis y requerimientos del sistema porque la metodología nos permite realizar las iteraciones y modificaciones necesarias para alcanzar mayor calidad en el desarrollo del sistema.

  La utilización del Lenguaje UML permitió diseñar un sistema seguro, confiable y mantenible.

  Comenzando por los casos de uso que nos marcaron las necesidades que el sistema debería cumplir para cada uno de los actores pasando por el diagrama de secuencia que nos ayudó a implementar en el sistema los métodos y mensajes que se implementarían en el sistema.

  Pasando al modelo de datos se hizo el modelo de entidad-relación usando como herramienta a postgres para posteriormente modelar el diagrama de clases con la herramienta Enterprise Architec.

  Para la programación se empleó el modelo vista controlador con este modelo se reduce el tiempo que se empleará en la programación. En nuestro caso hicimos uso de Eclipse Europa porque facilita la conexión con la base de datos y presenta una interfaz cómoda para la programación.

- En relación al segundo componente se realizó la capacitación al personal en base al cronograma preparado para su realización, donde se hacer conocer la importancia y ventajas que tienen hoy en día el uso de sistemas en cualquier campo de aplicación.

3.2 Recomendaciones

- En base al primer componente se recomienda realizar la implementación del sistema en red para así poder brindar una mejor información a los feligreses y a las parroquias.

  Para el desarrollo de futuros proyectos con características similares a éste se recomienda hacer uso de las metodologías utilizadas como UML, RUP, ya que éstas son estándares para el desarrollo de éste tipo de aplicaciones, ya que permitieron llegar a la conclusión del Proyecto.

  Se recomienda llevar a cabo una planificación para salvaguardar la información backups, con el motivo de evitar pérdidas de información.

  Al momento de desarrollar un software es conveniente tomar en cuenta la consistencia entre modelos, ya que si estamos programando con código orientado a objetos, nuestro motor de base de datos también debe ser orientado a objetos y documentación. Para lo cual se re recomienda profundizar más sobre esta nueva tecnología en base de datos.

- En base al segundo componente se recomienda consultar el Manual de Usuario para el uso adecuado del sistema.