SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB PARA EL CONTROL DE ACTIVIDADES ACADÉMICAS Y ADMINISTRATIVAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE NARIÑO

ALVARO DANIEL CORTÉS GUZMÁN

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2007

SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB PARA EL CONTROL DE ACTIVIDADES ACADÉMICAS Y ADMINISTRATIVAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE NARIÑO

ALVARO DANIEL CORTÉS GUZMÁN

Trabajo de grado presentado como requisito parcial para optar por el título de Ingeniero de Sistemas

Director:
Mg. NELSON ANTONIO JARAMILLO ENRIQUEZ

Asesor:
Ing. OSCAR ORLANDO CEBALLOS ARGOTE

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2007

NOTA DE ACEPTACIÓN

_____

_____

_____

_____
Jurado

_____
Jurado

_____
Director

_____
Asesor

San Juan de Pasto, Junio 4 de 2007

“Las ideas y conclusiones aportadas en el Trabajo de Grado son responsabilidad exclusiva del autor.”

Artículo 1º del Acuerdo Nº 324 de octubre 11 de 1966, emanado del Honorable Concejo Directivo de la Universidad de Nariño.

AGRADECIMIENTOS

Agradezco profundamente el apoyo incondicional a mi director y profesor NELSON ANTONIO JARAMILLO, sin sus ideas y colaboración este trabajo nunca se hubiera desarrollado. Agradezco también a mi asesor OSCAR CEBALLOS, por la confianza depositada en mi y sus valiosas enseñanzas. No puedo dejar por fuera a mi gran amigo JAVIER SANTACRUZ quien me brindo y compartió gran parte de su conocimiento, su ayuda fue una parte primordial en el desarrollo de este proyecto. Gracias Virgencita por estar a mi lado siempre, gracias Jesús mi señor por tu bendición, gracias Dios mio por darme la vida.

DEDICATORIA

Dedicado a los cuatro pilares de mi vida, ellos son mi familia: Mi Madre Fanny Guzmán por traerme a este mundo y estar siempre conmigo, Mi Novia Nayibe Erasso por comprenderme y apoyarme, ser mi amiga y compañera, por ser tan especial y por ser el amor de mi vida y mis pequeños, mi dos tesoros, mis hijos, Esteban y Sebastián, mi fuente de inspiración y motor para salir adelante.

RESUMEN

El Sistema de Información Orientado a la Web para el control de actividades académicas y administrativas de la Facultad de Ingeniería de la Universidad de Nariño, surge por la necesidad de implementar un sistema de información que se ajuste a las necesidades que se presentan en el manejo y control oportuno de todas las actividades que se desarrollan en la Facultad, y que se encargue de gestionar de manera confiable, ágil y oportuna la gran cantidad de información que se maneja en los procesos de registro y asignación de tareas mejorando así el desempeño y la calidad de servicio que se presta.

El sistema de información esta orientado a manejar la información de las diferentes dependencias y procesos de manera estándar, lo cual permitirá compartir y adaptar la información para diversos procesos académicos y administrativos sin ambigüedades y redundancias. Establece unas reglas claras para que el Director de Departamento o Decano tenga el control de todos los usuarios del sistema y a su vez permite una comunicación en doble sentido, informando avances de tareas o notas especiales.

En el desarrollo del proyecto se utilizó la metodología del Proceso Unificado de Desarrollo de Software, utilizando el Análisis y Diseño Orientado a Objetos con el Lenguaje de Modelado Unificado “UML”. Con esta metodología se garantiza que el software resultante se acerque mas a la realidad, además, es más fácil mantener y reutilizar, también permite su desarrollo por etapas y susceptible de futuras mejoras.

El sistema se desarrolló con la utilización del Lenguaje de programación orientado a objetos Java y su orientación a la web “JSP” (Java Server Pages), el sistema gestor de base de datos fue PostgreSQL 8.1, se utilizó tecnología ajax y javascript, con lo que se garantizan los resultados esperados en las etapas de análisis y diseño. .

ABSTRACT

The Guided System of Information to the Web for the Control of Academic and Administrative Activities of the Faculty of Engineering of the University of Nariño, it arises for the necessity of implementing a system of information that is adjusted to the necessities that are presented in the handling and opportune control of all the activities that are developed in the Faculty, and that it takes charge of negotiating in a reliable, agile and opportune way the great quantity of information that is managed in the registration processes and assignment of tasks improving this way the acting and the quality of service that it is lent.

The system of information this guided to manage the information of the different dependences and processes in a standard way, that which will allow to share and to adapt the information for diverse academic and administrative processes without ambiguities and redundancies. It establishes some clear rules so that each Director of Department or Dean has the control of all the users of the system and in turn it allows a communication in double sense, informing advances of tasks or special notes.

In the development of the project the methodology of the Unified Process of Development of Software was used, using the Analysis and Guided Design to Objects with the Unified Model Language “UML”. With this methodology it is guaranteed that the resulting software comes closer but to the reality it is also easier of maintaining and reuse, it also allows its development for stages and susceptible of future improvements.

The system development was created with the use of the programming Language oriented to objects "Java" and their orientation to the web "JSP" (Java Server Pages), the system database agent was PostgreSQL 8.1, was used technology ajax and javascript, with what the prospective results of the analysis stages and design are guaranteed.

CONTENIDO

pág.

LISTA DE TABLAS

LISTA DE FIGURAS

GLOSARIO

Agenda. Nombre con el cual se identifica a la aplicación por medio de la cual el usuario puede consultar fácil y eficientemente las tareas, citas y notas de su interés, registrar el avance de las tareas a su cargo e imprimir o guardar digitalmente información mostrada en la agenda.

Tarea. Trabajo que debe hacerse en tiempo limitado, por lo tanto incluye una fecha de inicio y una fecha de terminación, un responsable.

Cita. Señalamiento, asignación de día, hora y lugar para verse y hablarse dos o más personas. Reunión o encuentro entre dos o más personas, previamente acordado.

Nota. Marca o señal que se pone en algo para recordar o para dar a conocerlo. Es un recordatorio, una información sobre algo importante. Grabar en la memoria algo que se debe recordar.

INTRODUCCIÓN

Con el surgimiento de las aplicaciones Web se cambia el modelo de utilizar recursos estáticos de información controlados por el contenido, a usar entornos de aplicaciones dinámicos controlados por el usuario, siendo cada vez más factible emplearlas como una herramienta para registrar, actualizar y consultar información, procesos que siempre trabajan con datos variables en todo momento.

La Facultad de Ingeniería de la Universidad de Nariño, necesita un sistema de información para la mayoría de procesos que desarrolla, uno de estos es permitir optimizar y agilizar el control de las actividades que se desarrollan en su interior de forma manual, para prestar así un servicio eficiente a todas los miembros de la institución relacionados.

Teniendo en cuenta lo anterior, se desarrolló este proyecto, el cual implementa un Sistema de Información orientado a la Web que facilita y mejora el registro, actualización y consulta de las actividades y la información generada en los procesos académicos y administrativos de la Facultad de Ingeniería de la Universidad de Nariño, con el propósito de mejorar el desarrollo de todas las actividades que en el momento se realizan en forma manual.

Se presenta el desarrollo de este proyecto organizado de la siguiente forma: en la primera parte se describe el problema existente, el alcance y delimitaciones, para plantear tanto el objetivo general como los objetivos específicos; posteriormente, se determinan los antecedentes para continuar con la metodología a utilizar y por último, terminar con la elaboración del cronograma a seguir para el desarrollo del mismo.

1. DESCRIPCIÓN DEL PROBLEMA

1.1 PLANTEAMIENTO DEL PROBLEMA

En la Facultad de Ingeniería de la Universidad de Nariño se lleva a cabo procesos de carácter administrativos y académicos que generan diferentes actividades y responsabilidades, muchas de ellas recibidas y asignadas de manera verbal sin ningún tipo de soporte físico y otras que si tienen algún tipo de soporte en documentos como: actos administrativos, acuerdos, resoluciones, proposiciones, constancias, actas de entrega, entre otros.

El control de estas actividades se lleva a cabo de una manera ordenada pero no eficiente debido a la cantidad información que se maneja en la dependencia. En ocasiones se genera perdida del control y se da lugar a inconsistencias, fallas y errores en procesos.

Un ejemplo claro, es la búsqueda de una actividad específica para dar respuesta a la misma (tal vez el proceso más importante), actualmente solo la secretaria conoce cuales son las actividades que se desarrollan y donde se encuentran los soportes de cada una. Cuando se necesita información, es esta persona quien debe suspender su trabajo, realizar la búsqueda necesaria y demás. Esto implica dedicación de tiempo valioso del personal a la búsqueda, retrasando otros procesos que exigen atención inmediata.

También, cabe anotar, que la entrega de la información a tiempo en los procesos administrativos y académicos es parte del éxito de la Facultad de Ingeniería, sin embargo, la información no es transmitida oportunamente a las personas u organizaciones que están en continuo contacto.

1.2 FORMULACIÓN DEL PROBLEMA

¿Cómo mejorar la administración de la información y el control del desarrollo de los procesos de carácter administrativos y académicos en la Facultad de Ingeniería de la Universidad de Nariño?

1.3 SISTEMATIZACIÓN DEL PROBLEMA

¿Cómo mejorar la administración de la información relacionada con las actividades académicas y administrativas que se desarrollan en la Facultad de Ingeniería?

¿Cómo administrar la información personal de los integrantes de la Facultad que pueden ser responsable de una actividad especifica?

¿Cómo permitir que los integrantes de la Facultad se mantengan informados sobre asuntos de su interés y puedan hacer reportes del desarrollo de las actividades a su cargo?

1.4 ALCANCE Y DELIMITACIONES

Se pretende manejar las actividades de carácter administrativo y académico que se desarrollan en la Facultad utilizando tres conceptos claves: tareas, citas y notas. Una tarea es una actividad principal que tiene un tiempo definido, unos responsables y que necesita un registro permanente de su desarrollo, una cita es la convocatoria a una evento o suceso que se realizara en una fecha y hora especifica y por último, una nota simplemente es un recordatorio, un mensaje algo que no presenta mucha importancia pero que es conveniente informar.

La herramienta informática propuesta permite registrar, modificar y consultar datos relacionados con esta información, por parte de los interesados, facilitando la comunicación entre responsables y el jefe del departamento, el seguimiento de una actividad y la culminación exitosa de la misma.

Por lo tanto, se perfilan dos módulos en particular:

El módulo de administración de actividades. Este módulo se encargará del registro, modificación y consulta de actividades por los administradores de actividades, es decir, los que coordinan el trabajo en el programa, jefe de departamento y secretaria. Permitirá asignar responsables a una actividad dependiendo de la carga de cada uno, consultar el avance de actividades, y el resultado de las mismas.

Además, permitirá el control de usuarios responsables, es decir información de los docentes del departamento y otros que correspondan.

El módulo de la agenda. Este módulo se encargará de mostrar a los responsables, toda la información de las actividades relacionadas con el programa y en las cuales se encuentra involucrado, dependiendo de ciertos criterios y condiciones de búsqueda y tendrá acceso a información resumida y detallada de tareas, citas y notas.

También, permitirá por parte del responsable registrar el avance de una tarea específica.

El Sistema de Información esta orientado a la Web, necesita que este funcionando en Internet para mejorar la comunicación. Obviamente también implica que existan mecanismo para el control y posea la seguridad necesaria. La herramienta se desarrollará bajo software libre, el cual, trae consigo una autorización para que cualquiera pueda usarlo, copiarlo y distribuirlo, ya sea literal o con modificaciones, gratis o mediante una gratificación.

1.5 JUSTIFICACIÓN

Los Sistemas de Información se han convertido en una herramienta indispensable dentro de las organizaciones e instituciones aportando soluciones prácticas y rápidas a rutinas de trabajo habituales. En este sentido, el proyecto: “Sistema de Información orientado a la Web para el control de actividades académicas y administrativas de la Facultad de Ingeniería de la Universidad de Nariño” se convierte en una herramienta tecnológica necesaria para la Facultad de Ingeniería debido al volumen de información y actividades que se tratan en su interior.

La asignación y consulta de las actividades asignadas o recibidas en la secretaría de la Facultad de Ingeniería se agiliza en gran media, puesto que la creación, recepción y búsqueda se realizan en menor tiempo. Esto, gracias a la organización sistemática de las responsabilidades en base a unas reglas y prioridades que determinan su organización. El personal administrativo se beneficiará a través de la creación de una agenda virtual que ofrece un seguimiento a las actividades generadas por la información procesada en secretaría. De igual forma, docentes y estudiantes en general contarán indirectamente con una herramienta que facilita y agiliza la respuesta a diferentes solicitudes que se hagan a la secretaría de la Facultad.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Desarrollar un sistema de información orientado a la Web que permita procesar, clasificar y administrar actividades de carácter administrativo y académico de la Facultad de Ingeniería de la Universidad de Nariño.

2.2 OBJETIVOS ESPECÍFICOS

- Desarrollar un componente para administrar la información relacionada con las actividades académicas y administrativas que se desarrollan en la facultad, dentro del módulo administrativo.

- Implementar un componente para administrar la información personal de los integrantes de la Facultad que están incluidos en las actividades académicas y administrativas, como parte del módulo administrativo.

- Realizar un módulo para consulta por parte de los integrantes de la Facultad donde puedan tener acceso a información de asuntos interés general como particular y puedan realizar reportes sobre su desarrollo de actividades que llevan a cargo.

- Desarrollar un componente para manejar la seguridad de acceso a la información tanto en el módulo administrativo como en el módulo de consulta.

3. ANTECEDENTES

Anteriormente, se han desarrollado distintos sistemas de información que en cierta medida están relacionados con el presente proyecto, entre otros:

El ingeniero Vicente Aux, propuso un "Sistema de Información orientado a la Web para el manejo y archivo electrónico de documentos públicos de la Universidad de Nariño"[1], el cual facilita la transferencia de los documentos públicos que circulan en formato electrónico entre las distintas oficinas y dependencias de la Universidad de Nariño. Sin embargo, su objetivo principal se limita únicamente al transporte de documentos, y no a la creación, recepción, clasificación y búsqueda de los mismos, además no tiene en cuenta las actividades que estos generan.

De igual forma, el ingeniero Edwin Cabrera, propone el "Desarrollo e Implementación de un Sistema de Información para el manejo de información de Proyectos de Grado y Horarios de la Facultad de Ingeniería"[2], el cual ofrece un seguimiento al ciclo de vida de un trabajo de investigación, pasantía o extensión a la comunidad, al igual que, la creación sistematizada de los horarios de la Facultad de Ingeniería. El software esta desarrollado con Visual Basic y Microsoft Access, lo que impide orientarlo a la Web, además de solo estar dedicado a una actividad específica, los trabajos de grado.

Como se puede apreciar la necesidad de controlar las actividades de la Facultad siempre ha estado presente y buenos proyectos se han desarrollado, talvez el problema es que no ha existido un plan general que contenga la visión clara de hacia donde van encaminados todos estos esfuerzos, que los integre para buscar el mejoramiento global de la Facultad.

Como consecuencia de lo anterior, muchos proyectos han cumplido con su objetivo general, pero no se han implementado, solo han servido como requisitos para obtener el título profesional.

[1] Sistema de Información orientado a la Web para el manejo y archivo electrónico de documentos públicos de la Universidad de Nariño

[2] Desarrollo e Implementación de un Sistema de Información para el manejo de información de Proyectos de Grado y Horarios de la Facultad de Ingeniería

4. MARCO TEÓRICO

4.1 ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Los conceptos de análisis y diseño orientados a objetos (O-O) se originaron a partir de desarrollos en los lenguajes modernos de programación. Estos lenguajes O-O tienen nuevas estructuras que se sienten, que mejoran al mantenimiento del programa y hacen que grandes partes de los programas sean reutilizables. El consecuente reciclado de partes del programa, debe reducir el costo de desarrollo de los sistemas basados en computadora. Ya ha probado ser muy efectivo en el desarrollo de interfaces gráficas de usuario y bases de datos. Debido a que los lenguajes O-O tienen diferentes construcciones, se deben especificar los sistemas de computadora en forma tal que se maximice el uso efectivo de esas construcciones.

Se cree que las técnicas orientadas a objetos son mejores que los enfoques más antiguos para el manejo del ritmo de cambio cada vez más grande en muchas de las organizaciones actuales. Por ejemplo, muchos productos están siendo hechos cada vez más, bajo pedido o fabricados en lotes pequeños, conforme los fabricantes buscan mayor concentración sobre la satisfacción del cliente y la penetración de nichos de mercado. Esta tendencia significa cambios frecuentes al software que está integrado con estos productos. El cambio constante de personas y responsabilidades significa una necesidad cada vez mayor para el desarrollo y mantenimiento de sistemas. Se piensa que las técnicas O-O trabajan bien este tipo de situaciones, donde sistemas de información complicados están sufriendo mantenimiento, adaptación y rediseño continuos.

Los conceptos de análisis y diseño orientados a objetos fueron desarrollados para dar soporte a la tecnología de programación O-O. El desarrollo de esta tecnología de programación no fue una evolución instantánea, sino la evolución de un conjunto de conceptos algo desconectados que han sido puestos juntos para forma un nuevo paradigma para la ingeniería de software.[3]

[3] KENDALL K y KENDALL J. Análisis y Diseño de Sistemas. México: Prentice Hall, 1997. p 860.

4.2 PROGRAMACIÓN ORIENTADA A OBJETOS

Actualmente una de las áreas más importantes en el comercio y en el ámbito académico es la orientación a objetos. La programación orientada a objetos promete un amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo. Un objeto puede considerarse como una especie de cápsula dividida en tres partes: las relaciones, las propiedades y los métodos.

Las **relaciones** permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

Las **propiedades** distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de la cual se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los **métodos** son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

4.3 PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE

El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo software iterativo e incremental. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational o simplemente RUP. El Proceso Unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. De la misma forma, el Proceso Unificado de Rational, también es un marco de trabajo extensible, por lo que muchas veces resulta imposible decir si un refinamiento particular del proceso ha sido derivado del Proceso Unificado o del RUP. Por dicho motivo, los dos nombres suelen utilizarse para referirse a un mismo concepto.

El nombre Proceso Unificado se usa para describir el proceso genérico que incluye aquellos elementos que son comunes a la mayoría de los refinamientos existentes. También permite evitar problemas legales ya que Proceso Unificado de Rational o RUP son marcas registradas por IBM (desde su compra de Rational Software Corporation en 2003). El primer libro sobre el tema se denominó, en su versión española, El Proceso Unificado de Desarrollo de Software (ISBN 84-7829-036-2) y fue publicado en 1999 por Ivar Jacobson, Grady Booch y James Rumbaugh, conocidos también por ser los desarrolladores del UML, el Lenguaje Unificado de Modelado. Desde entonces los autores que publican libros sobre el tema y que no están afiliados a Rational utilizan el término Proceso Unificado, mientras que los autores que pertenecen a Rational favorecen el nombre de Proceso Unificado de Rational.

4.3.1 Características del proceso unificado. Iterativo e Incremental. El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas: Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio sólo consta de varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

Figura 1. Diagrama Proceso Unificado de Desarrollo de Software

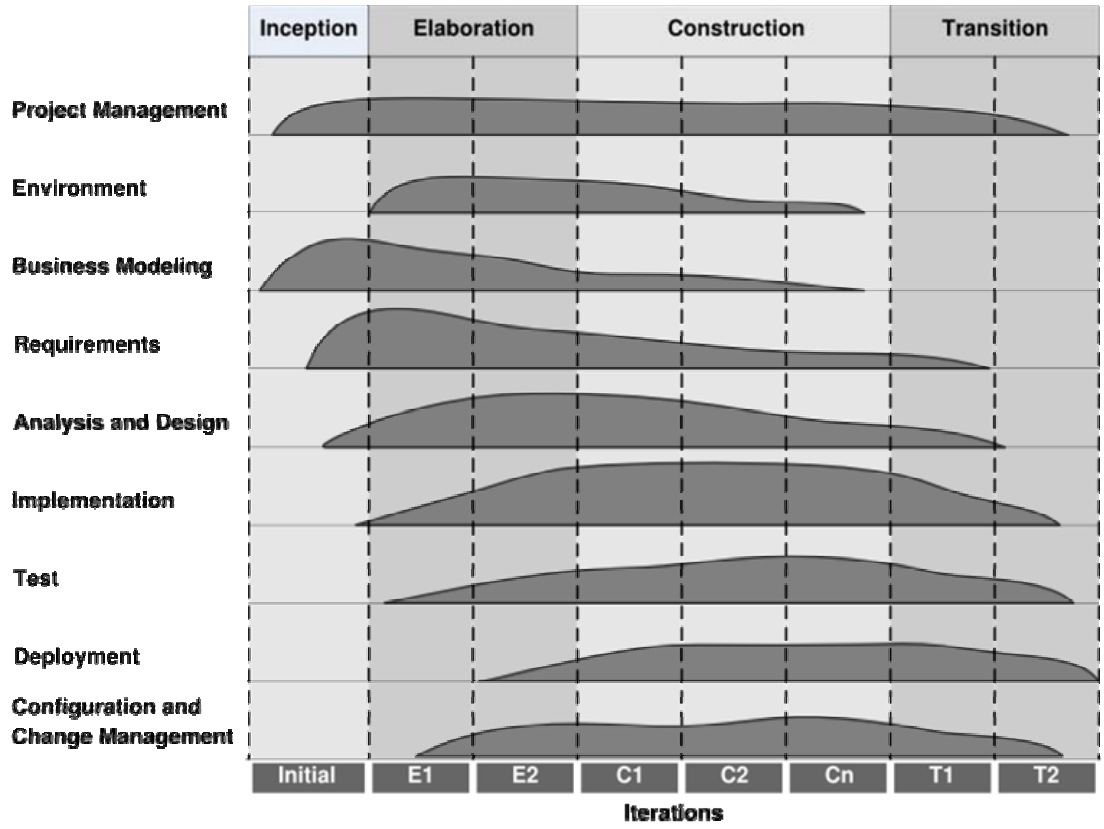


Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto.

Dirigido por los casos de uso. En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc. el proceso dirigido por casos de uso es el rup.

Centrado en la arquitectura. El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura software de un sistema. La analogía con la construcción es clara, cuando se construye un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería y otros.

Enfocado en los riesgos. El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

4.4 LENGUAJE UNIFICADO DE MODELADO UML

UML es una especificación de notación orientada a objetos. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representa la arquitectura del proyecto. Con UML se debe olvidar del protagonismo excesivo que se le da al diagrama de clases, éste representa una parte importante del sistema, pero solo representa una vista estática, es decir muestra al sistema parado. Se sabe su estructura pero no se sabe que le sucede a sus diferentes partes cuando el sistema empieza a funcionar. UML introduce nuevos diagramas que representa una visión dinámica del sistema. Es decir, gracias al diseño de la parte dinámica del sistema, puede darse cuenta en la fase de diseño de problemas de la estructura al propagar errores o de las partes que necesitan ser sincronizadas, así como del estado de cada una de las instancias en cada momento.

El diagrama de clases continua siendo muy importante, pero se debe tener en cuenta que su representación es limitada, y que ayuda a diseñar un sistema robusto con partes reutilizables, pero no a solucionar problemas de propagación de mensajes, ni de sincronización o recuperación ante estados de error. En resumen, un sistema debe estar bien diseñado, pero también debe funcionar bien.
UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos, se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

UML es ahora un estándar, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otro ramo.
UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento

forzado de una clase o relación, es decir mediante la restricción se fuerza el comportamiento que debe tener el objeto al que se aplica.

El Lenguaje Unificado de Modelado (Unified Modeling Language, UML), es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

UML es sólo un lenguaje y por tanto es tan sólo una parte de un método de desarrollo de software. UML es independiente del proceso, aunque para utilizarlo óptimamente se debe usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

El Lenguaje Unificado de Modelado fue desarrollado por Grady Booch, Jim Rumbaugh e Ivar Jacobson a partir de octubre de 1994. Las organizaciones que contribuyeron a la definición de 1.0 de UML fueron Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments y Unisys.

Los objetivos primarios que se persiguen al diseñar UML, son:

- Modelar sistemas, desde el concepto hasta los artefactos ejecutables, utilizando técnicas orientadas a objetos.

- Ser independiente de cualquier lenguaje de programación y de cualquier proceso de desarrollo.

- Fomentar el crecimiento de las herramientas OO (Orientadas a Objetos).

- Crear un lenguaje de modelado utilizable tanto por las personas como por las máquinas.

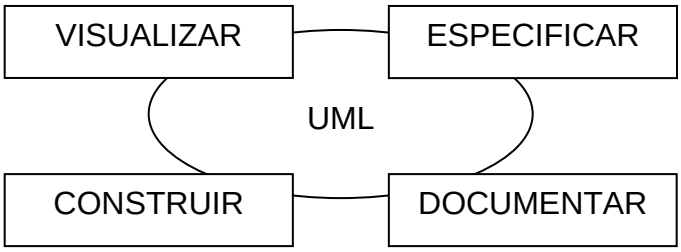
4.4.1 El Lenguaje Uml y los procesos de desarrollo. El lenguaje UML estandariza los artefactos y la notación, pero no define un proceso oficial de desarrollo. Esto es explicado, por:

- Aumentar las probabilidades de una aceptación generalizada de la notación estándar del modelado, sin la obligación de adoptar un proceso oficial.

- La esencia de un proceso apropiado admite mucha variación y depende e las habilidades del personal, de la razón investigación-desarrollo, de la naturaleza del problema, de las herramientas y de muchos otros factores.

Los diagramas permiten expresar modelos, los modelos son representaciones de la realidad. UML es un lenguaje para hacer diagramas (planos) de software estándar. Es un lenguaje para hacer diagramas (planos) de software estándar.

UML es un lenguaje para:



- **Visualizar.** "Lo piensas, lo codificas", hay algunas cosas que se modelan mejor directamente en el código, hay otras que necesitan ser expresadas textualmente y otras que se pueden expresar gráficamente. UML es un lenguaje gráfico estándar, bien definido. Un modelo escrito con UML permite que otro desarrollador pueda interpretarlo sin ambigüedad.

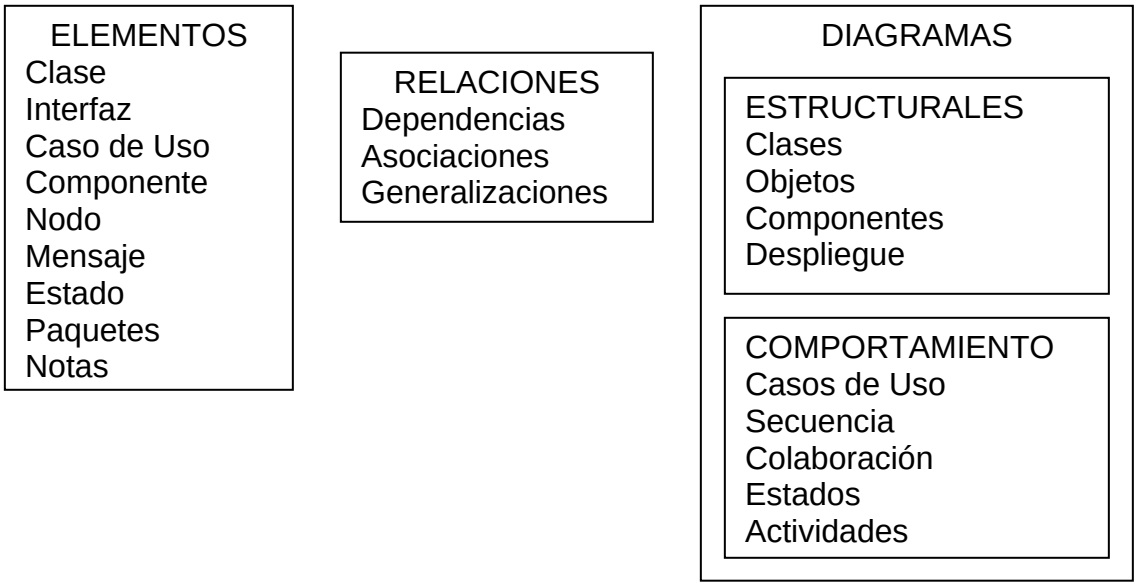
- **Especificar.** Los modelos deben ser precisos y completos.

- **Construir.** Los modelos pueden conectarse con gran variedad de lenguajes. Ingeniería directa. Generación de código a partir de un modelo UML en un lenguaje de programación. Ingeniería inversa. Reconstruir un modelo a partir de la implementación. También conocidas como Ingeniería de ida y vuelta.

- **Documentar.** En el desarrollo de software se producen diferentes artefactos. Un artefacto es el término general para cualquier producto del trabajo: código, gráficos, esquema de base de datos, documentos texto, diagramas, modelos,

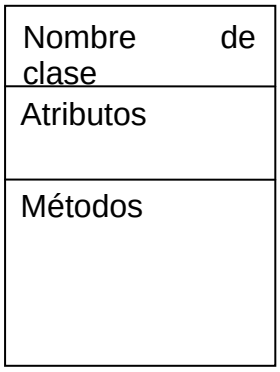
etc. Los artefactos no son solo para entregar al finalizar un proyecto, sino que son importantes en el desarrollo de los mismos.

Los bloques básicos de construcción de UML.



- **Elementos.** Componentes abstractos básicos de los diagramas, representan partes estáticas (conceptuales o materiales), partes dinámicas (comportamiento), partes organizativas y partes de anotación (explicativas).

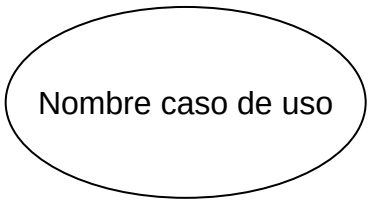
Clase. Descripción de un conjunto de objetos que comparten atributos, operaciones y relaciones. Un rectángulo con nombre, atributos y operaciones. Implementan interfaces.



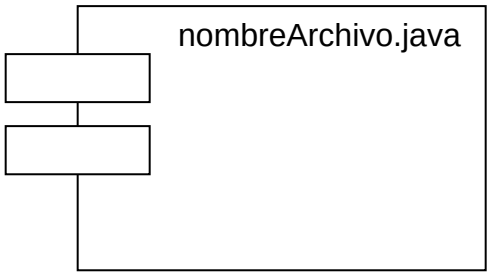
Interfaz. Colección de operaciones que especifican un comportamiento de una clase. Define un conjunto operaciones (sus nombres) pero no como se realizan. Un círculo junto con su nombre, conectada a la clase que la implementa.

Nombre de interfaz

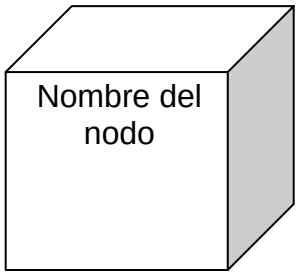
Caso de uso. Descripción de conjunto de secuencias de acciones que el sistema ejecuta.



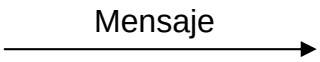
Componente. Es una representación física de algo lógico.



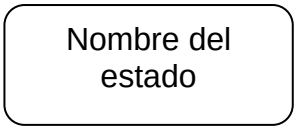
Nodo. Recurso físico, computacional.



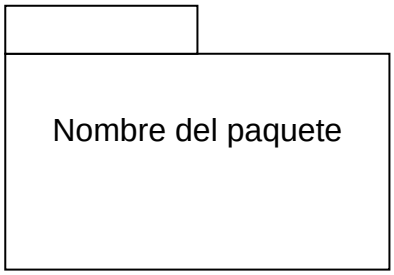
Mensaje. Enlaces o conexiones entre objetos, cuando se dan interacciones entre ellos



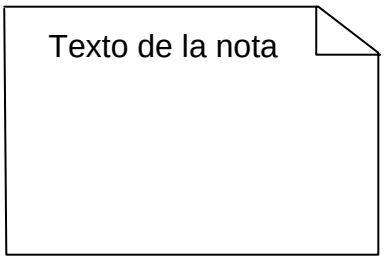
Estados. Etapa por la cual pasa un objeto durante su vida, respondiendo a eventos.



Paquete. Organizar elementos en grupos.



Notas. Mostrar comentarios junto a elementos.



Relaciones. Como se relacionan los elementos

Dependencia. Relación entre dos elementos, uno independiente y otro dependiente. Un cambio en uno afecta al otro.

Asociación. Relaciones entre objetos (Conexiones)

Generalización. Relación entre un elemento general (Padre) y otro especializado (Hijo) Herencia

Realización. Relación donde un elemento especifica que otro realizará una operación.

Diagramas. Representación gráfica de un conjunto de elementos y sus relaciones.

Permiten visualizar el sistema desde diferentes puntos de vista.
Por ejemplo:
La vista de casos de uso
La vista de diseño
La vista de implementación
La vista de despliegue

Un elemento puede aparecer en uno o más diagramas.

Diagramas de clases. Representa las clases, interfaces y relaciones entre ellas. Se muestran los conceptos.

Diagramas de objetos. Representa un conjunto de objetos y sus relaciones.

Diagramas de componentes. Representa un conjunto de componentes y sus relaciones.

Diagramas de despliegue. Representa los nodos y sus relaciones, con los componentes que incluye.

De Comportamiento. Representan aspectos dinámicos. Las partes cambiantes.

Diagramas de casos de uso. Representan los actores y los casos de uso con sus relaciones. Casos de uso ayudan a organizar y modelar el sistema.

Diagramas de secuencia. Diagrama de Interacción, que representa un conjunto de objetos y los mensajes enviados y recibidos entre ellos, en el tiempo.

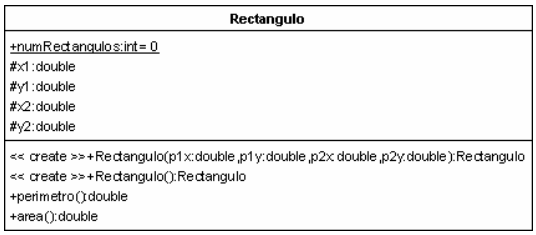
Diagrama de colaboración. Diagrama de Interacción, que representa un conjunto de objetos y los mensajes enviados y recibidos entre ellos, en la estructura.

Diagrama de estados. Máquina de estados constituida por estados, transiciones, eventos y actividades. Modelan el comportamiento de una interfaz, clase o colaboración.

Diagrama de actividades. Representa un flujo secuencial de actividades y los objetos que actúan

- **Mecanismos comunes que se aplican a lo largo del lenguaje.**

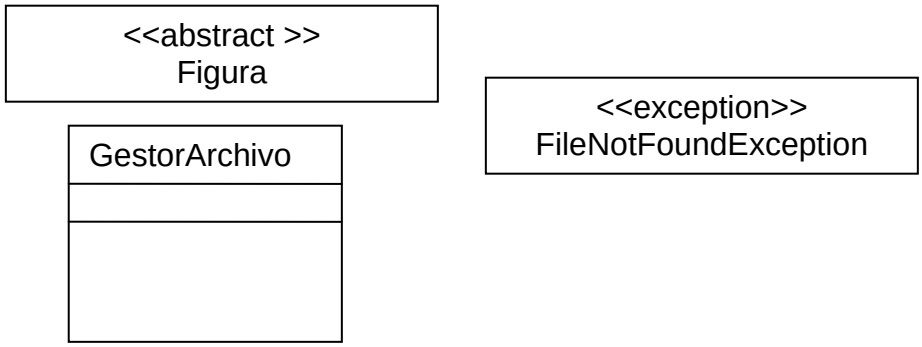
Adornos. Notación clara de aspectos importantes de los elementos. En una clase, la visibilidad de atributos y métodos.



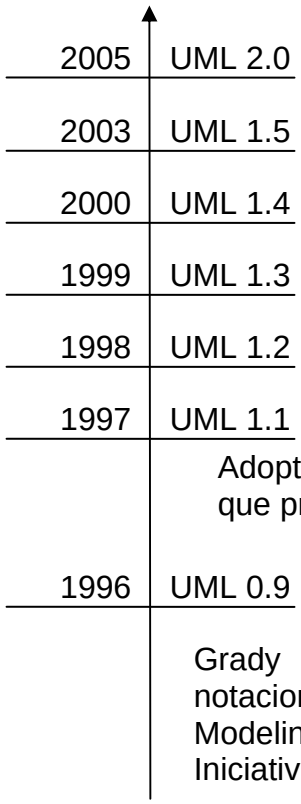
Extensibilidad. No es un lenguaje cerrado y se pueden expresar variaciones de los elementos iniciales. La extensibilidad se manifiesta a través de los estereotipos.

Estereotipos. Nuevos tipos de bloques Ejemplo: <<abstract>>, <<exception>>.

Notas. En general los comentarios sobres restricciones y demás expresados a través de notas.



- **Historia.**



4.5 Aplicación de UML y patrones en el análisis y diseño orientado a objetos

4.5.1 ¿Qué es el análisis y diseño? El análisis pone énfasis en una investigación del problema y los requisitos, en vez de ponerlo en una solución. Por ejemplo, si se desea un nuevo sistema de información informatizado para una biblioteca. ¿Cómo se utilizará?

"Análisis" es un término amplio, es mas adecuado calificarlo, como el análisis de requisitos (un estudio de los requisitos) o análisis de objetos (un estudio de los objetos del dominio).

El Diseño enfatiza en una solución conceptual que satisface los requisitos, en vez de situarse en la implementación. Por ejemplo, una descripción del esquema de una base de datos y objetos software. Finalmente, los diseños pueden ser implementados.

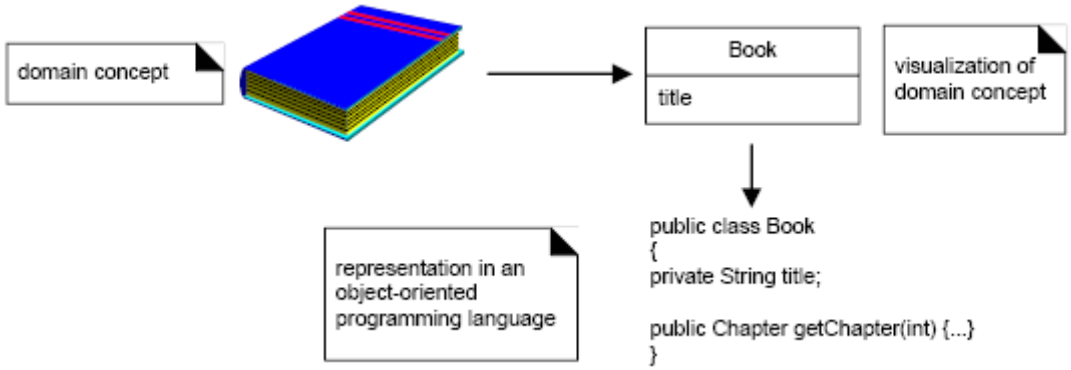
Como con el análisis, es mas apropiado calificar el término como diseño de objetos o diseño de base de datos.

El análisis y el diseño se han resumido en la frase hacer lo correcto (análisis), y hacerlo correcto (diseño).

4.5.2 ¿Qué son el análisis y diseño orientado a objetos? Durante el análisis orientado a objetos, se presta especial atención encontrar y describir los objetos –o conceptos- en el dominio del problema. Por ejemplo, en el caso del sistema de información de la biblioteca, algunos conceptos son Libro, Biblioteca y Socio.

Durante el diseño orientado a objetos, se presta especial atención a la definición de los objetos software y en como colaboran para satisfacer los requisitos. Por ejemplo, en el sistema de la biblioteca, un objeto software Libro podría tener un atributo titulo y un método obtenerCapitulo.

Figura 2. Objetos en el diseño y clases



Por último, durante la implementación o programación orientada a objetos, los objetos de diseño se implementan, como la clase Java Libro.

Un ejemplo:

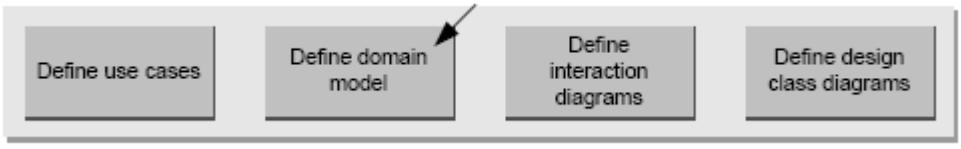
Se presenta de manera superficial, unos pocos pasos y diagramas claves, utilizando un ejemplo sencillo, un “juego de dados” el el que un jugador lanza dos dados. Si el total es siete, gana; en otro caso, pierde.

Definición de los casos de uso. El análisis de requisitos podría incluir una descripción de los procesos del dominio relacionados, que podrían representarse como casos de uso.

Los casos de uso no son artefactos orientados a objetos, son simplemente historias escritas. Sin embargo, son una herramienta muy popular en análisis de requisitos y son una parte importante del Proceso Unificado. Por ejemplo, aquí esta una versión breve del caso de uso Jugar una Partida de Dados:

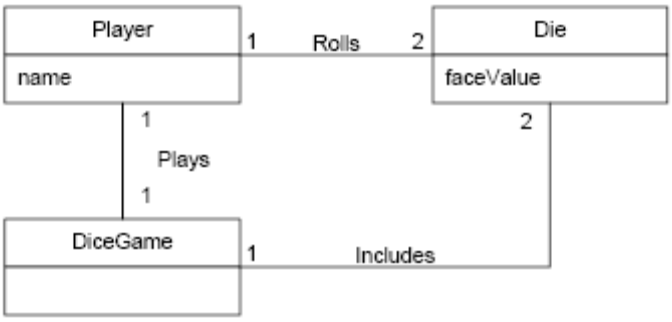
Jugar una partida de dados: Un jugador recoge y lanza los dados. Si el valor de las caras de los dados suman siete, gana; en otro caso pierde.

Definición de un modelo del dominio. La finalidad del análisis orientado a objetos es crear una descripción del dominio desde la perspectiva de la clasificación de objetos. Una descomposición del dominio conlleva una identificación de los conceptos, atributos y asociaciones que se consideran significativas. El resultado se puede expresar en un modelo del dominio, que se ilustra mediante un conjunto de diagramas que muestran los objetos o conceptos del dominio.



Por ejemplo, la figura muestra un modelo del dominio parcial.

Figura 3. Modelo dominio parcial

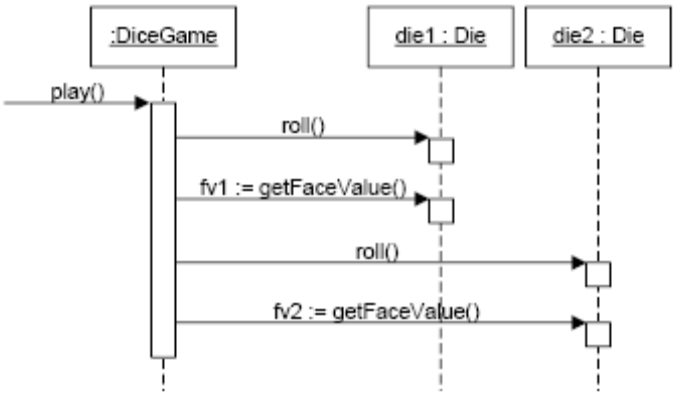


Este modelo ilustra los conceptos importantes Jugador, Dado y JuegoDados, con sus asociaciones y atributos. Nótese que un modelo del dominio no es una descripción de los objetos software, es una visualización de los conceptos en el dominio del mundo real.

Definición de los diagramas de interacción. La finalidad del diseño orientado a objetos es definir los objetos software y sus colaboraciones. Una notación habitual para ilustrar estas colaboraciones es el diagrama de interacción. Muestra el flujo de mensajes entre los objetos software y, por tanto, la invocación de métodos

Por ejemplo, se supone que desea la implementación de un juego de dados. El diagrama de interacción de la siguiente figura ilustra los pasos esenciales del juego, enviando mensajes a las clases JuegoDados y Dado.

Figura 4. Diagramas de interacción juego de dados



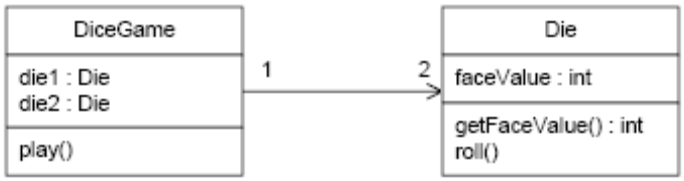
Nótese que aunque en el mundo real un jugador lanza los dados, en el diseño software el objeto JuegoDados tira los dados (es decir, envía mensajes a los objetos Dado). Los diseños de los objetos software y los programas se inspiran en los dominios del mundo real, pero no son modelos directos o simulaciones del mundo real.

Definición de los diagramas de clases de diseño

Además de la vista dinámica de las colaboraciones entre los objetos que se muestra mediante los diagramas de interacción, es útil crear una vista estática de las definiciones de las clases mediante un diagrama de clases de diseño.

Por ejemplo, en el juego de dados, un estudio del diagrama de interacción conduce al diagrama de clases de diseño parcial que se muestra en la siguiente figura. Puesto que se envía el mensaje jugar al objeto JuegoDados, JuegoDados requiere un método jugar, mientras la clase Dado requiere los métodos lanzar y obtenerValorCara.

Figura 5. Diagramas de clase de diseño parcial juego de dados



A diferencia del modelo de dominio, este diagrama no muestra conceptos del mundo real, sino clases software.

4.6 LENGUAJE DE PROGRAMACIÓN JAVA

Java es un lenguaje de propósito general, orientado a objetos y neutral a la arquitectura. El compilador toma un archivo fuente y genera un archivo objeto el cual es interpretado y ejecutado por una máquina virtual JVM (Java Virtual Machine), por lo tanto, es portable al nivel de código objeto. Java es totalmente orientado a objetos. La tecnología de objetos se presenta como el siguiente paso lógico de la programación procedural. La programación procedural enfatiza la especificación de tareas (procedimientos) para realizar una tarea. Cuando los sistemas se vuelven grandes y distribuidos, ya no es tan fácil definir estas tareas. Un método más natural es definir objetos y acciones entre objetos. Java soporta los mecanismos estándar de la orientación a objetos tales como encapsulamiento, herencia y polimorfismo.

Java es seguro. El chequeo de datos es muy estricto. Antes de que un código byte sea ejecutado es revisado por la JVM para detectar accesos no autorizados a la memoria. El manejo de la memoria es sencillo. Cada objeto se crea con el operador new y el programador no es responsable de liberar la memoria. Cuando un objeto ya no se utiliza, el recolector de basura automáticamente reclama ese espacio. Java es distribuido. Tiene librerías para programación con TCP/IP (el protocolo de comunicaciones estándar de Internet).

Java es concurrente. Tiene librerías para hacer programas con múltiples hilos de ejecución simultánea.

4.7 JAVASCRIPT

Se trata de un lenguaje interpretado que pese a su nombre no tiene nada que ver con Java, son dos cosas distintas, principalmente porque Java sí que es un lenguaje de programación completo, lo único que comparten es la misma sintaxis. No es un lenguaje de programación propiamente dicho, es un lenguaje script u orientado a documento y tan solo se puede usar para mejorar una página Web con algunas cosas sencillas (revisión de formularios, efectos en la barra de estado, etc.), es decir sólo pueden escribirse 'scripts' que funcionarán en el entorno de una página Web, interpretado por un explorador Con Javascript se puede dar respuesta a eventos iniciados por el usuario tales como la entrada de una forma o algún enlace. Esto sucede sin ningún tipo de transmisión de tal forma que cuando un usuario escribe algo en una forma, no es necesario que sea transmitido hacia el servidor, verificado y devuelto. Las entradas son verificadas por la aplicación cliente y pueden ser transmitidas después de esto.

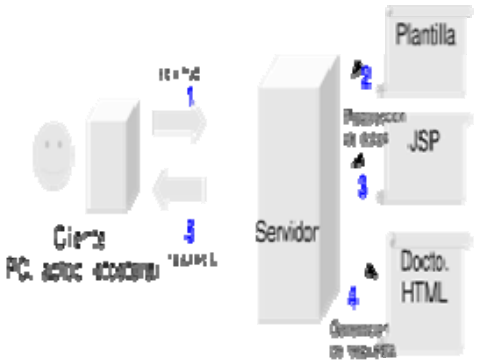
JavaScript es un lenguaje sin tipo de datos. Es decir, no necesita declarar el tipo de datos de las variables explícitamente. Es más, en muchos casos JavaScript realiza la conversión automáticamente cuando sea necesario. JavaScript, al igual que Java, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML. Al ser la más sencilla, es por el momento la más extendida.

4.8 JAVA SERVER PAGES

Java Server Pages (JSP), en el campo de la informática, es una tecnología para crear aplicaciones web. Es un desarrollo de la compañía Sun Microsystems, y su funcionamiento se basa en scripts, que utilizan una variante del lenguaje java. La JSP es una tecnología Java que permite a los programadores generar contenido dinámico para web, en forma de documentos HTML, XML, o de otro tipo. Las JSP's permite al código Java y a algunas acciones predefinidas ser incrustadas en el contenido estático del documento web.

En las JSP se escribe el texto que va a ser devuelto en la salida (normalmente código HTML) incluyendo código java dentro de él para poder modificar o generar contenido dinámicamente. El código java se incluye dentro de las marcas de etiqueta <% y %>, a esto se le denomina scriptlet.

Figura 6. Funcionamiento de JSP



En una posterior especificación, se incluyeron taglib; esto es, la posibilidad de definir etiquetas nuevas que ejecuten código de clases java. La asociación de las etiquetas con las clases java se declaran en archivos de configuración en XML.

La principal ventaja de JSP frente a otros lenguajes es que permite integrarse con clases Java (.class) lo que permite separar en niveles las aplicaciones web, almacenando en clases java las partes que consumen más recursos así como las que requieren más seguridad, y dejando la parte encargada de formatear el documento html en el archivo jsp. La idea fundamental detrás de este criterio es el de separar la lógica del negocio de la presentación de la información. Independientemente de la certeza de la aseveración, Java es conocido por ser un lenguaje muy portable (su lema publicitario reza: escríbelo una vez, córrelo donde sea), y sumado a las capacidades de JSP se hace una combinación muy atractiva. Sin embargo JSP no se puede considerar un script al 100% ya que antes de ejecutarse el servidor web compila el script y genera un servlet, por lo tanto, se puede decir que aunque este proceso sea transparente para el programador no deja de ser una aplicación compilada. La ventaja de esto es algo más de rapidez y disponer del API de Java en su totalidad.

Debido a esto la tecnología JSP, así como Java está teniendo mucho peso en el desarrollo web profesional (sobre todo en intranets).
Microsoft, la más directa competencia de Sun, ha visto en esta estrategia de Sun una amenaza, lo que le ha llevado a que su plataforma .NET incluya su lenguaje de scripts ASP.NET que permite ser integrado con clases .NET (ya estén hechas en C++, VisualBasic o C#) del mismo modo que jsp se integra con clases Java.

Para ejecutar las páginas JSP, se necesita un servidor Web con un contenedor Web que cumpla con las especificaciones de JSP y de Servlet. Tomcat 5 es una completa implementación de referencia para las especificaciones Java Servlet 2.2 y JSP 1.1.

JSP 2.0. A partir de la versión 2.0 de JSP se incluyó nuevas funcionalidades con el fin de aumentar la productividad, reutilización de código y separar de manera clara la presentación y la lógica de negocio (normalmente implementada en clases Java). Se desarrolló Expression Language (EL) para hacer referencia a componentes Java de una manera limpia y elegante evitando el uso de scriptlets. Se simplificó la creación de librerías de etiquetas personalizadas.

4.9 TECNOLOGÍA AJAX

Acrónimo de Asynchronous JavaScript And XML (JavaScript y XML asíncronos, donde XML es un acrónimo de eXtensible Markup Language), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma, es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

AJAX es una combinación de tres tecnologías ya existentes: XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información. Document Object Model (DOM) accedido con un lenguaje de scripting por parte el usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.

El objeto XMLHttpRequest para intercambiar datos asincrónicamente con el servidor web. En algunos frameworks y en algunas situaciones concretas, se usa un objeto iframe en lugar del XMLHttpRequest para realizar dichos intercambios. XML es el formato usado comúnmente para la transferencia de vuelta al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Como el DHTML, LAMP o SPA, AJAX no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

Historia de Ajax. A pesar de que el término "Ajax" fuese creado en 2005, la historia de las tecnologías que permiten Ajax se remonta a una década antes con la iniciativa de Microsoft en el desarrollo de Scripting Remoto. Sin embargo, las técnicas para la carga asíncrona de contenidos en una página existente sin requerir recarga completa remontan al tiempo del elemento iframe (introducido en Internet Explorer 3 en 1996) y el tipo de elemento layer (introducido en Netscape 4 en 1997, abandonado durante las primeras etapas de desarrollo de Mozilla). Ambos tipos de elemento tenían el atributo src que podía tomar cualquier dirección URL externa, y cargando una página que contenga javascript que manipule la página paterna, pueden lograrse efectos parecidos al Ajax.

El Microsoft's Remote Scripting (o MSRS, introducido en 1998) resultó un sustituto más elegante para estas técnicas, con envío de datos a través de un applet Java el cual se puede comunicar con el cliente usando JavaScript. Esta técnica funcionó en ambos navegadores, Internet Explorer versión 4 y Netscape Navigator versión 4. Microsoft la utilizó en el Outlook Web Access provisto con la versión 2000 de Microsoft Exchange Server.

4.10 SERVIDOR DE APLICACIONES WEB

Los Servidores Web suministran páginas Web a los navegadores (como por ejemplo, Netscape Navigator, Internet Explorer de Microsoft) que lo solicitan. En términos más técnicos, los servidores Web soportan el Protocolo de Transferencia de Hipertexto conocido como HTTP (HyperText Transfer Protocol), el estándar de Internet para comunicaciones Web. Usando HTTP, un servidor Web envía páginas Web en HTML, así como otros tipos de scripts a los navegadores o browsers cuando éstos lo requieren. Cuando un usuario hace clic sobre un enlace (link) a una página Web, se envía una solicitud al servidor Web para localizar los datos nombrados por ese enlace. El servidor Web recibe esta solicitud y suministra los datos que han sido solicitados (una página HTML, un script interactivo, una página Web generada dinámicamente desde una base de datos,...) o bien devuelve un mensaje de error.

4.10.1 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 Pages (JSP) de Sun Microsystems. Se considera un servidor de aplicaciones.

Tomcat es un servidor web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Historia. Tomcat empezó siendo una implementación de la especificación de los servlets comenzada por James Duncan Davidson, que trabajaba como arquitecto de software en Sun y que posteriormente ayudó a hacer el proyecto open source y en su donación a la Apache Software Foundation. Duncan Davidson inicialmente esperaba que el proyecto se convirtiese en open source y dado que la mayoría de los proyectos open source tienen libros de O'Reilly asociados con un animal en la portada, quiso ponerle al proyecto nombre de animal. Eligió Tomcat (gato), pretendiendo representar la capacidad de cuidarse por sí mismo, de ser independiente.

4.11 SISTEMA GESTOR DE BASE DE DATOS

El sistema manejador de bases de datos es la porción más importante del software de un sistema de base de datos. Un DBMS (Data Base Management System) es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de alguna tarea específica. Las funciones Principales de un DBMS son:

- Crear y organizar la Base de datos.

- Establecer y mantener las trayectorias de acceso a la base de datos de tal forma que los datos puedan ser accesados rápidamente.

- Manejar los datos de acuerdo con las peticiones de los usuarios.

- Registrar el uso de las bases de datos. Interacción con el manejador de archivos. Esto a través de las sentencias en DML al comando del sistema de archivos. Así el Manejador de base de datos es el responsable del verdadero almacenamiento de los datos.

- Respaldo y recuperación. Consiste en contar con mecanismos implantados que permitan la recuperación fácilmente de los datos en caso de ocurrir fallas en el sistema de base de datos.

- Control de concurrencia. Consiste en controlar la interacción entre los usuarios concurrentes para no afectar la inconsistencia de los datos.

- Seguridad e integridad. Consiste en contar con mecanismos que permitan el control de la consistencia de los datos evitando que estos se vean perjudicados por cambios no autorizados o previstos.

4.11.1 PostgreSQL. PostgreSQL es un motor de base de datos, es servidor de base de datos relacional libre, liberado bajo la licencia BSD.

Características. Algunas de sus principales características, son:

Funciones. Bloques de código que se ejecutan en el servidor. Pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos da, desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientación a objetos o la programación funcional.

Los disparadores (triggers en inglés) son funciones enlazadas a operaciones sobre los datos.

Algunos de los lenguajes que se pueden usar son los siguientes:

* Un lenguaje propio llamado PL/pgSQL (similar al PL/SQL de oracle).

* Lenguajes de "Scripting", tales como:
    * PL/Perl.

    * plPHP.
    * PL/Python.
    * PL/Ruby.
    * PL/sh.
    * PL/Tcl.
    * PL/Scheme. PL/Scheme.
* Lenguajes compilados como:
    * C.
    * C++.
    * Java (via PL/Java).
* Lenguajes de dominios de conocimiento específico:
    * Lenguaje para aplicaciones estadísticas R through PL/R.

PostgreSQL soporta funciones que retornan "filas", donde la salida puede tratarse como un conjunto de valores que pueden ser tratados igual a una fila retornada por un consulta (query en inglés).

Las funciones pueden se definidas para ejecutarse con los derechos del usuario ejecutor o con los derechos de un usuario previamente definido. El concepto de funciones, en otros DBMS, son muchas veces referidas como "procedimientos almacenados" (stored procedures en inglés).

Alta concurrencia. Mediante un sistema denominado MVCC (Acceso concurrente multiversión) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Amplia variedad de tipos nativos. PostgreSQL provee nativamente soporte para:

Números de precisión arbitraria.
Texto de largo ilimitado.
Figuras geómetricas (con una variedad de funciones asociadas)
Direcciones IP (IPv4 e IPv6).
Bloques de direcciones estilo CIDR.
Direcciones MAC.
Arrays.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

Otras características. Claves ajenas también denominadas Llaves ajenas o Llaves Foráneas (foreign keys).

Disparadores (triggers).
Vistas.
Integridad transaccional.
Herencia de tablas.
Tipos de datos y operaciones geométricas.

Historia. PostgreSQL es el último resultado de una larga evolución comenzada con el proyecto Ingres en la Universidad de Berkeley. El líder del proyecto, Michael Stonebraker abandonó Berkeley para comercializar Ingres en 1982, pero finalmente regresó a la academia. Tras su retorno a Berkeley en 1985, Stonebraker comenzó un proyecto post-Ingres para resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos.

El proyecto resultante, llamado Postgres, era orientado a introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones - las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En Postgres la base de datos "comprendía" las relaciones y podía obtener información de tablas relacionadas utilizando reglas.

Comenzando en 1986, el equipo liberó una serie de documentos describiendo la base del sistema y en 1988 poseían un prototipo funcional. La versión 1 fue liberada a un pequeño grupo de usuarios en junio de 1989, seguido por la versión 2 con un sistema de reglas reescrito en junio de 1990. Para la versión 3, liberada en 1991, el sistema de reglas fue reescrito nuevamente, pero también agregó soporte para múltiples administradores de almacenamiento y un sistema de consultas mejorado. Hacia 1993, Postgres había crecido inmensamente en popularidad y poseía una demanda asfixiante de nuevas funcionalidades. Tras

liberar la versión 4, la cual era una simple versión de limpieza, el proyecto fue abandonado. A pesar de que el proyecto Postgres hubiese finalizado oficialmente, la licencia BSD bajo la cual Postgres había sido liberado permitió a desarrolladores de código abierto el obtener una copia del código para continuar su desarrollo.

5. METODOLOGÍA

Para el proceso de desarrollo de esta herramienta informática se tomó como referencia dos modelos de desarrollo de software el Proceso Unificado y la Ingeniería Web. Debido a que un modelo solo representa en que orden y frecuencia se desarrollan las actividades comunes a todo proceso de desarrollo de software, es necesario también especificar el como se va a desarrollar cada una de esas actividades, esto es lo que se conoce como metodología.

La metodología a utilizar en este proyecto es el Análisis y Diseño Orientado a Objetos, apoyado en la utilización UML (Lenguaje Unificado de Modelado), y tomando como referencia la metodología expuesta en el libro UML y Patrones[4].

El modelo de desarrollo de software llamado Proceso Unificado UP, tiene las siguientes características:

Es dirigido por los requisitos de los usuarios. Por lo tanto, se debe hacer un estudio para conocer que necesitan y desean los usuarios en el nuevo sistema.

Es iterativo e incremental. Divide el sistema en partes más pequeñas, las primeras iteraciones cubren de forma rápida los aspectos más importantes de las funcionalidades del sistema, las demás se enfocan a aspectos menos trascendentales pero que generan incrementos funcionales necesarios al sistema.

Permite obtener una vista completa del diseño del sistema. Luego de obtener una vista general de los requisitos que los usuarios presentan se bosqueja el sistema de acuerdo con los mismos.

Además, el UP, maneja cuatro fases importantes, que son:

Fase de inicio. Donde se describe el producto final, y como será el plan de desarrollo.

[4] LARMAN, Craig. UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Madrid: Pearson Educación S.A., 2da. Edición. 2003. 624 p.

Fase de elaboración. En la cual se detalla que aspectos debe cubrir el software y se crean los primeros modelos del sistema.

Fase de construcción. Donde se desarrolla el software y terminan de definirse los modelos.

Fase de transición. Donde se realizan las pruebas convenientes y se corrigen algunos defectos.

6. RESULTADOS OBTENIDOS

6.1 MODELO DE CASOS DE USO

6.1.1 Listado de características. Los requisitos son capacidades o condiciones con la cuales debe ser conforme el sistema a desarrollar. Para registrar los requisitos de este proyecto, se utilizo un listado de características de alto nivel, esto significa que los requisitos están considerados dentro de un contexto amplio de uso del sistema, orientados a obtener un resultado de valor o cumplir con un objetivo del usuario.

El siguiente es el listado de características que se tuvieron en cuenta para este proyecto, se adjunta el estado de la característica al finalizar el proyecto, de acuerdo con la tabla que se específica a continuación: (Tabla 1)

Tabla 1. Tabla de estados para una característica del proyecto

Estado (E)		Descripción
P	Propuesto	Estado usado para describir características que han sido discutidas pero que no se han revisado o aceptado oficialmente.
A	Aprobado	Capacidades que se juzgan como útiles y factibles, y ha sido aceptadas para implementarlas en la aplicación.
I	Incorporada	Características incorporadas al producto en un momento específico.

Tabla 2. Tabla de listado de características del proyecto

Características relacionadas con la administración de la agenda

ID	**CARACTERÍSTICA1**
Descripción	El sistema debe permitir manejar a los usuarios de la agenda, es decir, puede ingresar a nuevos usuarios, modificar los datos de usuarios, activar o desactivar usuarios, dependiendo de su vinculación con el programa.
Estado	Incorporada

ID	CARACTERÍSTICA2
Descripción	El sistema debe permitir gestionar las tareas, citas y notas de los usuarios, permitir el ingreso, modificación y estado de cada una.
Estado	Incorporada

ID	CARACTERÍSTICA3
Descripción	El sistema debe permitir consultar las tareas, de forma que permita saber el estado de una tarea en cuanto a su desarrollo, o el resultado final de una tarea.
Estado	Incorporada

Características relacionadas con el uso de la agenda por el usuario

ID	CARACTERÍSTICA4
Descripción	La agenda debe permitir consultar las tareas, citas y notas que están asignadas al usuario correspondiente. Debe tener dos alternativas de los últimos tres días (ayer, hoy y mañana) y consulta por calendario (día, semana, mes y semestre)
Estado	Incorporada

ID	CARACTERÍSTICA5
Descripción	La agenda debe permitir al usuario registrar el avance de las tareas asignadas.
Estado	Incorporada

ID	CARACTERÍSTICA6
Descripción	La agenda debe permitir al usuario imprimir y guardar en un formato electrónico, información sobre una o más tareas, citas y notas a través de la agenda.
Estado	Incorporada

ID	CARACTERÍSTICA7
Descripción	La agenda debe permitir que se notifique por correo electrónico que debe revisar la agenda.
Estado	Propuesta

ID	CARACTERÍSTICA8
Descripción	La agenda permite cargar los archivos de resultados de una tarea específica.
Estado	Propuesta

6.1.2 Listado de reglas. Las reglas del dominio especifican la forma como se van a desarrollar las características del sistema. Son políticas, leyes, entre otras que influyen sobre los requisitos de la aplicación. Estas son las reglas que se consideraron para el desarrollo del proyecto.

Tabla 3. Tabla de listado de reglas del proyecto

ID	Descripción
REGLA1	Se necesita tener información sobre los usuarios de la agenda que permita localizarlos rápidamente, como es: el teléfono fijo, el teléfono móvil, la dirección de correo electrónico.

ID	Descripción
REGLA2	Se debe acceder fácilmente a la información de los usuarios registrados, para manejo y consulta de los mismos.

ID	Descripción
REGLA3	Los usuarios de la agenda son profesores, estudiantes o funcionarios relacionados con el Programa de Ingeniería de Sistemas de la Universidad de Nariño. El director del departamento e incluso la secretaria pueden tener su propia agenda.

ID	Descripción
REGLA4	Las tareas citas y notas solo son manejadas por la secretaria, el director del departamento, ellos también las registran aunque el monitor pude intervenir en la trascripción de las mismas.

ID	Descripción
REGLA5	La consulta en la agenda de las tareas citas y notas debe ser rápida y eficiente, es decir, clasificar las tareas, citas y notas de acuerdo a criterios como periodos de tiempo: los últimos tres días (ayer, hoy y mañana) o por calendario (día, semana, mes y semestre)

ID	Descripción
REGLA6	Las tareas y citas deben manejar una prioridad que se refiere a la importancia de la misma con relación a su cumplimiento.

ID	Descripción
REGLA7	Se debe asignar por lo menos un responsable para una tarea, aunque pueden existir otros. La asignación del responsable se puede hacer explícitamente cuando un documento generado en la dependencia asigna a una persona como responsable de una

	tarea o según el criterio de la Secretaria y en última instancia del Director del Programa.

ID	Descripción
REGLA8	Una tarea puede encontrarse en el tiempo en diferentes estados, entre ellos, registrado cuando arranca, suspendido (opcional), cancelada (opcional). Además tiene un periodo de cumplimiento que puede ser horas o días.

ID	Descripción
REGLA9	Las citas no se pueden asignar en una misma fecha y hora.

ID	Descripción
REGLA10	Una nota no implica una obligación, simplemente se trata de un recordatorio. Por lo tanto las notas tienen un tiempo de duración.

6.1.3 Listado de actores. Los actores son generalmente personas (usuarios) que necesitan satisfacer sus necesidades de información a través del sistema que se esta desarrollando, es decir, que utilizan los servicios del sistema, por lo regular se clasifican de acuerdo con los roles que desempeñan en el contexto del uso del sistema. También se pueden considerar como actores a otros sistemas que interactúan con el sistema en desarrollo. Para este proyecto se identificaron los siguientes actores:

Tabla 4. Tabla de listado de actores del sistema

Nombre	Administrador de actividad
Descripción	Se encarga de realizar todas las acciones relacionadas con las actividades (Citas, Notas, Tareas) de los usuarios. Puede ser la secretaria y el director. ✓ Necesita controlar las tareas que se generan en el Departamento, y si es posible asignar al responsable del desarrollo de esa actividad. ✓ Necesita saber que tareas se están desarrollando en este momento y a quien están asignadas. ✓ Necesita registrar y saber en que concluyó una determinada actividad (Resultado).

Nombre	UsuarioAgenda (Responsable)
Descripción	Es la persona que esta asignada como responsable de una actividad. ✓ Necesita saber cuales son las tareas en las cuales figura como responsable. ✓ Necesita saber cuales son las tareas asignadas a él y más próximas que debe cumplir. ✓ Necesita registrar cual fue el resultado de dicha actividad.

6.1.4 Listado de casos de uso. Un caso de uso, representa una forma de usar el sistema (dar soporte a un usuario durante un proceso). Se constituyen como segmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. El siguiente es el listado de casos de uso que se identificaron para el desarrollo del proyecto:

Tabla 5. Tabla de listado de casos de uso

MÓDULO DE ADMINISTRACIÓN DE LA AGENDA

ID	**CASO DE USO 1**
Caso de uso	Iniciar sesión administrativa
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad debe estar registrado en el sistema
Poscondiciones	El administrador de actividad inicia sesión en el sistema
Flujo Básico	
1. El administrador de actividad introduce el nombre y la clave. 2. El sistema verifica que el nombre existe. 3. El sistema inicia una sesión en el sistema. 4. El sistema carga la información relacionada con la administración del sistema.	
Flujos Alternativos	
2a. Si el sistema no encuentra un usuario que corresponda al nombre y a la clave introducida 1. El sistema indica que no existe un usuario con ese nombre y clave 2b. Si el sistema encuentra un usuario que corresponde al nombre introducido pero no coincide la clave 1. El sistema indica que la clave es incorrecta.	

ID	**CASO DE USO 2**
Caso de uso	Cerrar sesión administrativa
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad debe estar registrado en el

	sistema
Poscondiciones	El administrador de actividad cierra la sesión en la agenda
Flujo Básico	
1. El administrador de actividad inicia el cierre de la sesión 2. El sistema cierra la sesión del usuario. 3. El sistema muestra al usuario un mensaje confirmando que la sesión se cerró correctamente.	
Flujos Alternativos	

ID	**CASO DE USO 3**
Caso de uso	Adicionar usuario de la agenda
Actor principal	Administrador de actividad
Actores relacionados	Usuario-agenda: quiere que sus datos personales queden registrados correctamente, y obtener acceso a su agenda para consultar las tareas, citas y notas
Precondiciones	El administrador de actividad ha ingresado al sistema Se necesita crear al usuario-agenda para poder asignarle tareas.
Poscondiciones	Se crea el usuario-agenda. Se le asigna un nombre y clave para que pueda acceder al sistema.
Flujo Básico	
1. El administrador de actividad comienza el registro del nuevo Usuario-agenda. 2. El administrador de actividad introduce los datos del nuevo usuario como son: identificación, nombre completo, dirección de domicilio, dirección de trabajo, número del teléfono fijo y número del celular, e-mail. 3. El sistema verifica que no exista otro usuario usando la misma identificación. 4. Si no existe otro usuario con la misma identificación, el sistema asigna al nuevo usuario, el nombre de usuario que es la misma identificación, la contraseña que es la misma identificación y el estado de *activo*. 5. El sistema registra el nuevo usuario-agenda y muestra un mensaje confirmando la creación del usuario y muestra el nombre del usuario y la clave asignada.	
Flujos Alternativos	
3a. Si el usuario existe 1. El sistema informa que ya existe un usuario con esa identificación. 2. El administrador de actividad puede cambiar los datos del nuevo usuario. 5a. Si el sistema falla al ingresar el nuevo usuario 1. Muestra el mensaje de error y los cambios no se realizan.	
Observaciones	
La longitud del nombre de usuario y la contraseña no debe ser menor de 6 caracteres.	

ID	**CASO DE USO 4**

Caso de uso	Modificar usuario de la agenda
Actor principal	Administrador de actividad
Actores relacionados	Usuario-agenda: quiere que sus datos personales queden registrados correctamente.
Precondiciones	El administrador de actividad ha ingresado al sistema Se necesita actualizar los datos del usuario
Poscondiciones	Se actualiza los datos del usuario
Flujo Básico	
1. El administrador de actividad comienza la modificación del nuevo usuario. 2. El sistema muestra un listado de todos los usuarios que se encuentran registrados. 3. El administrador de actividad escoge el usuario cuyos datos se van a modificar. 4. El administrador de actividad ingresa los nuevos datos y solicita actualizar los datos del usuario. 5. El sistema registra los datos actualizados del usuario.	
Flujos Alternativos	
5a. Si el sistema falla al actualizar los datos del usuario 1. Muestra el mensaje de error y los cambios no se realizan.	
Observaciones	

ID	**CASO DE USO 5**
Caso de uso	Cambiar el estado de un usuario de la agenda
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema El usuario se haya desvinculado del Departamento
Poscondiciones	Se cambia el estado del usuario de la agenda
Flujo Básico	
1. El administrador de actividad comienza el cambio del estado del usuario. 2. El sistema muestra un listado de todos los usuarios que se encuentran registrados, ya sean activos o inactivos 3. El administrador de actividad escoge el usuario y cambia su estado de activo a inactivo o viceversa. 4. El sistema verifica que el usuario no es responsable del desarrollo de alguna tarea. 5. El sistema registra el estado del usuario.	
Flujos Alternativos	
4a. El sistema indica que el usuario tiene tareas asignadas. 1. El sistema señala el error y rechaza la petición. 2. El administrador de actividad inicia **modificar tarea** para reasignar las tareas pendientes. 5a. Si el sistema falla al actualizar los datos del usuario 1. Muestra el mensaje de error y los cambios no se realizan.	
Observaciones	

ID	CASO DE USO 6
Caso de uso	Buscar usuario de la agenda
Actor principal	Administrador
Precondiciones	El administrador de actividad ha ingresado al sistema El administrador de actividad necesita conocer más información sobre uno o más usuarios El administrador de actividad debe conocer algún dato del usuario o usuarios que quiere buscar
Poscondiciones	Se encuentra la información del usuario o usuarios que coincidan con los datos suministrados
Flujo Básico	
1. El administrador de actividad inicia la búsqueda del usuario. 2. El administrador de actividad escoge el criterio de búsqueda, escribe las palabras claves para buscar en ese criterio e inicia la búsqueda. 3. El sistema busca la información de todos los usuarios que coincidan con los criterios seleccionados. 4. El sistema muestra la información de los usuarios seleccionados.	
Flujos Alternativos	
3a. Si el sistema falla al buscar la información de los usuarios 1. Muestra el mensaje de error y los cambios no se realizan. 4a. Si no encuentra ninguna información que coincida con los criterios 1. El sistema informa que no existe ningún usuario con esos criterios	
Observaciones	
Los criterios pueden ser identificación, nombre, dirección de trabajo, dirección de domicilio, teléfono fijo, teléfono móvil y correo electrónico.	

ID	CASO DE USO 7
Caso de uso	Registrar tarea
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema Existe una tarea que se debe registrar
Poscondiciones	Se registra la tarea Puede asignarse uno o más responsables de la tarea
Flujo Básico	
1. El administrador de actividad comienza el registro de una nueva tarea. 2. El administrador de actividad ingresa la información de la tarea como es: tema, fecha de inicio, hora de inicio, fecha de finalización, hora de finalización, prioridad, categoría, descripción y recordatorio. 3. El sistema registra la tarea. 4. El sistema asigna a la tarea el estado de la tarea: *registrada*	
Flujos Alternativos	

2a. El administrador de actividad puede asignar uno o más responsables de la tarea registrada.
1. Se inicia **asignar responsables de tarea**.

Observaciones

Recordatorio. Fecha y hora en la cual se muestra un mensaje al usuario recordando su ejecución, la fecha desde la cual comienza el conteo regresivo para cumplir la tarea
Se maneja un color para la proximidad de ejecución de la tarea, para advertir al usuario a través de la agenda:
Negro – Normal Antes de recordatorio
Azul – Esta en el recordatorio antes de la fecha de finalización
Rojo – Se venció la fecha de finalización y aun esta en desarrollo

ID	**CASO DE USO 8**
Caso de uso	Asignar Responsables de Tarea
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema Existe una tarea que se debe asignar responsables
Poscondiciones	Se asigna uno o varios responsables a una tarea

Flujo Básico

1. El administrador de actividad comienza la asignación de responsables de una tarea.
2. El sistema despliega una lista de tareas en estado: *pendiente*, *reasignando* y *cancelado*.
3. El administrador de actividad seleccionan la tarea de la lista.
4. El administrador de actividad inicia la búsqueda de usuarios para asignar la tarea.
5. El sistema muestra los usuarios y la carga actual (*número total de tareas asignadas*).
6. El administrador de actividad selecciona los usuarios responsables y asigna la tarea.
7. El sistema cambia el estado de asignación de la tarea de *pendiente* a *en desarrollo*.
8. El sistema cambia el estado de desarrollo a 0%.

Flujos Alternativos

6a. Si el usuario tiene asignado más de un número determinado de tareas
1. El sistema advierte el número total de tareas asignadas
2. El sistema solicita confirmación para la asignación de la nueva tarea.

ID	**CASO DE USO 9**
Caso de uso	Registrar estado de la tarea
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema El estado de la tarea ha cambiado

Poscondiciones	Actualiza el estado de la tarea
Flujo Básico	
1. El administrador de actividad selecciona la tarea que se va modificar. 2. El administrador de actividad inicia la actualización del estado de la tarea. 3. El administrador de actividad determina el nuevo estado de la tarea de acuerdo a las novedades, junto con una descripción. 4. El sistema registra el estado de desarrollo de la tarea.	
Flujos Alternativos	
3a. El administrador de actividad cambia el estado de la tarea de *en desarrollo* a *suspendido* 1. El sistema registra la fecha de suspensión. 3b. El administrador de actividad cambia el estado de la tarea de *suspendido* a *en desarrollo* 1. El sistema recalcula la fecha de terminación de la tarea de acuerdo a la fecha de suspensión	
Observaciones	
Los estados posibles para asignar a una tarea son: *en desarrollo* a *cancelada*, *suspendida*, *reasignada* y *finalizada*. El estado de una tarea de *suspendido* debe pasar necesariamente a *en desarrollo*.	

ID	**CASO DE USO 10**
Caso de uso	Modificar tarea
Actor principal	Administrador de actividad
Precondiciones	El administrador Actividad ha ingresado al sistema Se necesita actualizar información de la tarea
Poscondiciones	Algunos datos de la tarea se modifican
Flujo Básico	
1. El administrador de actividad comienza la actualización de la tarea 2. El administrador de actividad escoger tarea a modificar. 3. El administrador de actividad modifica o actualiza la información de la tarea. 4. El sistema pide confirmación. 5. El administrador de actividad confirma la acción. 6. El sistema registra la modificación en la información de la tarea.	
Flujos Alternativos	
3a. El administrador de actividad necesita cambiar el estado de la tarea 1. El administrador de actividad inicia **registrar estado de la tarea.** 3b. El administrador de actividad necesita cambiar los responsables de la tarea 1. El administrador de actividad debe cambiar el estado de la tarea a *reasignado* y *cancelado* iniciando **registrar estado de la tarea**. 2. El administrador de actividad inicia **asignar responsable de tarea.**	
Observaciones	

ID	**CASO DE USO 11**

Caso de uso	Consultar tarea
Actor principal	Administrador actividad
Precondiciones	El administrador de actividad han ingresado al sistema Se necesita obtener información de varias tareas en un momento dado
Poscondiciones	Se obtienen listados de tareas Se obtiene información especifica de un tarea Se obtiene reportes impresos de listados de tareas

Flujo Básico

1. El administrador de actividad comienza la consulta de la tarea
2. El administrador de actividad establece los criterios para buscar tareas, por ejemplo: responsable(s), estado actual de la tarea, fecha de inicio, fecha de finalización, por rango de fecha, por prioridad, por categoría, por porcentaje de desarrollo, por palabras clave que se aplica sobre el tema (titulo) o la descripción de la tarea.
3. El sistema despliega todas las tareas que cumplan los criterios. La información que se muestra por cada tarea incluye: el tema, responsables, prioridad, fecha de inicio y de finalización, estado actual de la tarea, el porcentaje de desarrollo (cuando el estado seleccionado sea *en desarrollo*).
4. El administrador de la actividad puede organizar el listado de las tareas, ascendente o descendente por cualquier columna.

Flujos Alternativos

3a. Si el administrador de la actividad quiere consultar más información de una tarea.

1. El administrador de la actividad ubica la tarea en el listado y le indica al sistema que quiere acceder a información detallada de la misma.
2. El sistema muestra información detallada de la tarea que incluye: el tema, responsables, prioridad, categoría, fecha de inicio, fecha de finalización, la historia de los estados de la tarea junto con los comentarios, los porcentajes de desarrollo por cada estado entre otros.
3. El administrador de la actividad inicia **imprimir archivo** o **guardar archivo** con la información detallada de la tarea.

3b. Si el administrador de la actividad quiere modificar la información de una tarea especifica.

1. El administrador de la actividad ubica la tarea en el listado e inicia la **modificación de la tarea.**

4a. El administrador de actividad quiere imprimir el listado de las tareas

1. El administrador de la actividad inicia **imprimir archivo.**

4b. El administrador quiere guardar en un archivo el listado de las tareas.

1. El administrador de la actividad inicia **guardar archivo.**

Observaciones

El rango de fechas, involucra a todas las actividades cuya fecha de inicio y de finalización este dentro de los limites del rango.

ID	CASO DE USO 12
Caso de uso	Registrar cita
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema Existe una cita que se debe registrar
Poscondiciones	Se registra la cita con los citados Puede registrarse el periodo de tiempo de repetición de la cita
Flujo Básico	
1. El administrador de actividad comienza el registro de una nueva cita. 2. El administrador de actividad ingresa la información de la cita como es: asunto (texto explicativo sobre el motivo de la reunión), fecha y hora de publicación, fecha y hora de inicio, duración (expresada en horas; determina cuando se elimina de la agenda), prioridad (alta, media, baja), quien convoca (entidad o persona que convoca la reunión) y observaciones. 3. Se inicia **Asignar citado a una cita.** 4. El sistema registra la cita.	
Flujos Alternativos	
2a. El administrador de actividad puede establecer un periodo de repetición de la cita en el tiempo. 1. Se inicia **Especificar periodo de repetición de una cita.**	

ID	CASO DE USO 13
Caso de uso	Asignar citado a una cita
Actor principal	Administrador de actividad
Precondiciones	El administrador Actividad ha ingresado al sistema Existe una cita a la cual se le deben asignar citados
Poscondiciones	Se asigna uno o varios citados a una cita
Flujo Básico	
1. El administrador de actividad inicia la búsqueda de usuarios para especificar los citados. 2. El sistema muestra los usuarios registrados en el sistema. 3. El administrador de actividad selecciona los usuarios a ser citados. 4. El administrador de actividad confirma la selección. 5. El sistema registra los usuarios citados a la cita.	
Flujos Alternativos	
5a. Si el usuario tiene otra cita pendiente en la misma fecha y hora. 1. El sistema advierte que el usuario no puede ser citado.	
Observaciones	

ID	CASO DE USO 14
Caso de uso	Especificar periodo de repetición de una cita
Actor principal	Administrador de actividad
Precondiciones	El administrador Actividad ha ingresado al sistema Existe una cita a la cual se le quiere colocar el periodo de repetición.
Poscondiciones	Se especifica el periodo de repetición
Flujo Básico	
1. El administrador de actividad inicia la especificación del periodo de repetición de una cita. 2. El administrador especifica el periodo de repetición que puede ser: diaria (repetición cada determinado número de días), semanal (repetición cada determinado número de semanas, algunos del los siete días de la semana) y mensual (cada determinado número de meses, un día de una de las cuatro semanas del mes). 3. El administrador de actividad especifica la fecha de finalización del ciclo. 4. El sistema registra el periodo de repetición de la cita.	
Flujos Alternativos	
Observaciones	

ID	CASO DE USO 15
Caso de uso	Modificar cita
Actor principal	Administrador de actividad
Precondiciones	El administrador Actividad ha ingresado al sistema Se necesita actualizar información de la cita
Poscondiciones	Algunos datos de la cita se modifican
Flujo Básico	
1. El administrador de actividad comienza la actualización de la cita 2. El administrador de actividad escoge la cita a modificar. 3. El administrador de actividad actualiza la información de la cita. 4. El sistema pide confirmación. 5. El administrador de actividad confirma la acción. 6. El sistema registra la modificación en la información de la cita.	
Flujos Alternativos	
3a. Si el administrador de actividad necesita cambiar los citados a una cita 1. El administrador de actividad inicia **Asignar citados a una cita.** 3b. Si el administrador de actividad necesita cambiar el periodo de repetición de una cita 1. El administrador de actividad inicia **Especificar periodo de repetición de una cita**	
Observaciones	

ID	**CASO DE USO 16**
Caso de uso	Consultar cita
Actor principal	Administrador actividad
Precondiciones	El administrador de actividad han ingresado al sistema Se necesita obtener información de varias citas en un momento dado
Poscondiciones	Se obtienen listados de citas Se obtiene información especifica de un cita Se obtiene reportes impresos de listados de citas

Flujo Básico

1. El administrador de actividad comienza la consulta de la cita.
2. El administrador de actividad establece los criterios para buscar citas, por ejemplo: por rango de fecha, por prioridad.
3. El sistema despliega todas las citas que cumplan los criterios. La información que se muestra por cada cita incluye: fecha de la cita, asunto, convocante y tipo.
4. El administrador de la actividad puede organizar el listado de las citas, ascendente o descendente por cualquier columna.

Flujos Alternativos

3a. Si el administrador de la actividad quiere consultar más información de una cita.

1. El administrador de la actividad ubica la cita en el listado y le indica al sistema que quiere acceder a información detallada de la misma.
2. El sistema muestra información detallada de la tarea que incluye: fecha de la cita, fecha de publicación, duración, asunto, observación, convocante y tipo.
3. El administrador de la actividad inicia **imprimir archivo** o **guardar archivo** con la información detallada de la cita.

3b. Si el administrador de la actividad quiere modificar la información de una cita especifica.

2. El administrador de la actividad ubica la tarea en el listado e inicia la **modificación de la cita.**

4a. El administrador de actividad quiere imprimir el listado de las citas

2. El administrador de la actividad inicia **imprimir archivo.**

4b. El administrador quiere guardar en un archivo el listado de las citas.

2. El administrador de la actividad inicia **guardar archivo.**

Observaciones

El rango de fechas, involucra a todas las actividades cuya fecha de inicio y de finalización este dentro de los limites del rango.

ID	CASO DE USO 17
Caso de uso	Registrar nota
Actor principal	Administrador de actividad
Precondiciones	El administrador de actividad ha ingresado al sistema Existe una nota que se debe registrar
Poscondiciones	Se registra la nota
Flujo Básico	
1. El administrador de actividad comienza el registro de una nueva nota. 2. El administrador de actividad ingresa la información de la cita como es: titulo, descripción, hora y fecha. 3. El sistema registra la nota.	
Flujos Alternativos	

ID	CASO DE USO 18
Caso de uso	Consultar nota
Actor principal	Administrador actividad
Precondiciones	El administrador de actividad han ingresado al sistema Se necesita obtener información de varias notas en un momento dado
Poscondiciones	Se obtienen listados de notas Se obtiene información especifica de un nota Se obtiene reportes impresos de listados de notas
Flujo Básico	
El administrador de actividad comienza la consulta de la nota. El administrador de actividad establece los criterios para buscar notas, por ejemplo: por rango de fecha. El sistema despliega todas las notas que cumplan los criterios. La información que se muestra por cada cita incluye: fecha y hora de la nota, titulo y descripción. El administrador de la actividad puede organizar el listado de las notas, ascendente o descendente por cualquier columna.	
Flujos Alternativos	
4a. El administrador de actividad quiere imprimir el listado de las notas El administrador de la actividad inicia **imprimir archivo.** 4b. El administrador quiere guardar en un archivo el listado de las notas 3. El administrador de la actividad inicia **guardar archivo.**	
Observaciones	
El rango de fechas, involucra a todas las actividades cuya fecha de inicio y de finalización este dentro de los limites del rango.	

ID	CASO DE USO 19
Caso de uso	Generar archivo
Actor principal	Administrador de actividades

Actores relacionados	
Precondiciones	El administrador de actividades ha iniciado el sistema. Se necesita una copia digital del documento.
Poscondiciones	
Flujo Básico	
1. El administrador de actividades solicita al sistema una copia del archivo, especifica el nombre del archivo, el formato del archivo y la unidad de almacenamiento 2. El sistema guarda el archivo de acuerdo con las especificaciones dadas.	
Flujos Alternativos	
2a. El sistema encuentra algún fallo al comunicarse la unidad de almacenamiento. 1. El sistema muestra el mensaje de error.	
Observaciones	

ID	**CASO DE USO 20**
Caso de uso	Imprimir archivo
Actor principal	Administrador de actividades
Actores relacionados	
Precondiciones	El administrador de actividades ha iniciado el sistema. Se necesita imprimir el contenido de un documento.
Poscondiciones	
Flujo Básico	
1. El administrador de actividades solicita al sistema una copia impresa de un archivo 2. El sistema genera la impresión del archivo en papel	
Flujos Alternativos	
2a. El sistema encuentra algún fallo al comunicarse con la impresora. 1. El sistema muestra el mensaje de error.	
Observaciones	

MODULO DE CONSULTA DE LA AGENDA

ID	**CASO DE USO 21**
Caso de uso	Iniciar sesión agenda
Actor principal	Usuario-agenda
Precondiciones	El usuario-agenda debe estar registrado en el sistema
Poscondiciones	El usuario-agenda inicia sesión en la agenda
Flujo Básico	
5. El usuario-agenda introduce el nombre y la clave.	

6. El sistema verifica que el usuario-agenda existe. 7. El sistema inicia una sesión en la agenda. 8. El sistema carga la agenda actualizada del usuario.
Flujos Alternativos
2a. Si el sistema no encuentra un usuario que corresponda al nombre y a la clave introducida 2. El sistema indica que no existe un usuario con ese nombre y clave 2b. Si el sistema encuentra un usuario que corresponde al nombre introducido pero no coincide la clave 1. El sistema indica que la clave es incorrecta.

ID	**CASO DE USO 22**
Caso de uso	Cerrar sesión agenda
Actor principal	Usuario-agenda
Precondiciones	El usuario-agenda debe estar registrado en el sistema
Poscondiciones	El usuario-agenda cierra la sesión en la agenda
Flujo Básico	
4. El usuario-agenda inicia el cierre de la agenda 5. El sistema cierra la sesión del usuario. 6. El sistema muestra al usuario un mensaje confirmando que la agenda se cerró correctamente.	
Flujos Alternativos	

ID	**CASO DE USO 23**
Caso de uso	Consultar agenda
Actor principal	Usuario-agenda
Precondiciones	El usuario-agenda debe estar registrado en el sistema
Poscondiciones	El usuario-agenda consulta las tareas, citas, notas
Flujo Básico	
1. El sistema presenta la agenda (resumen de tareas, citas y notas) del usuario-agenda. 2. El usuario-agenda escoge la opción para consultar tareas, citas, notas. 3. El sistema muestra el listado correspondiente de acuerdo con la opción y a los criterios establecidos por el usuario-agenda.	
Flujos Alternativos	
2a. Si el usuario-agenda escoge consulta de Tareas 1. El usuario-agenda puede establecer criterios para ordenar el listado de tareas. Los filtros son: Categoría (las existentes en el sistema), Prioridad (alta, media o baja) y Estado (en desarrollo y suspendido). 2. El usuario-agenda escoge una tarea del listado para consultar el detalle de la misma 2b. Si el usuario-agenda consulta las Citas	

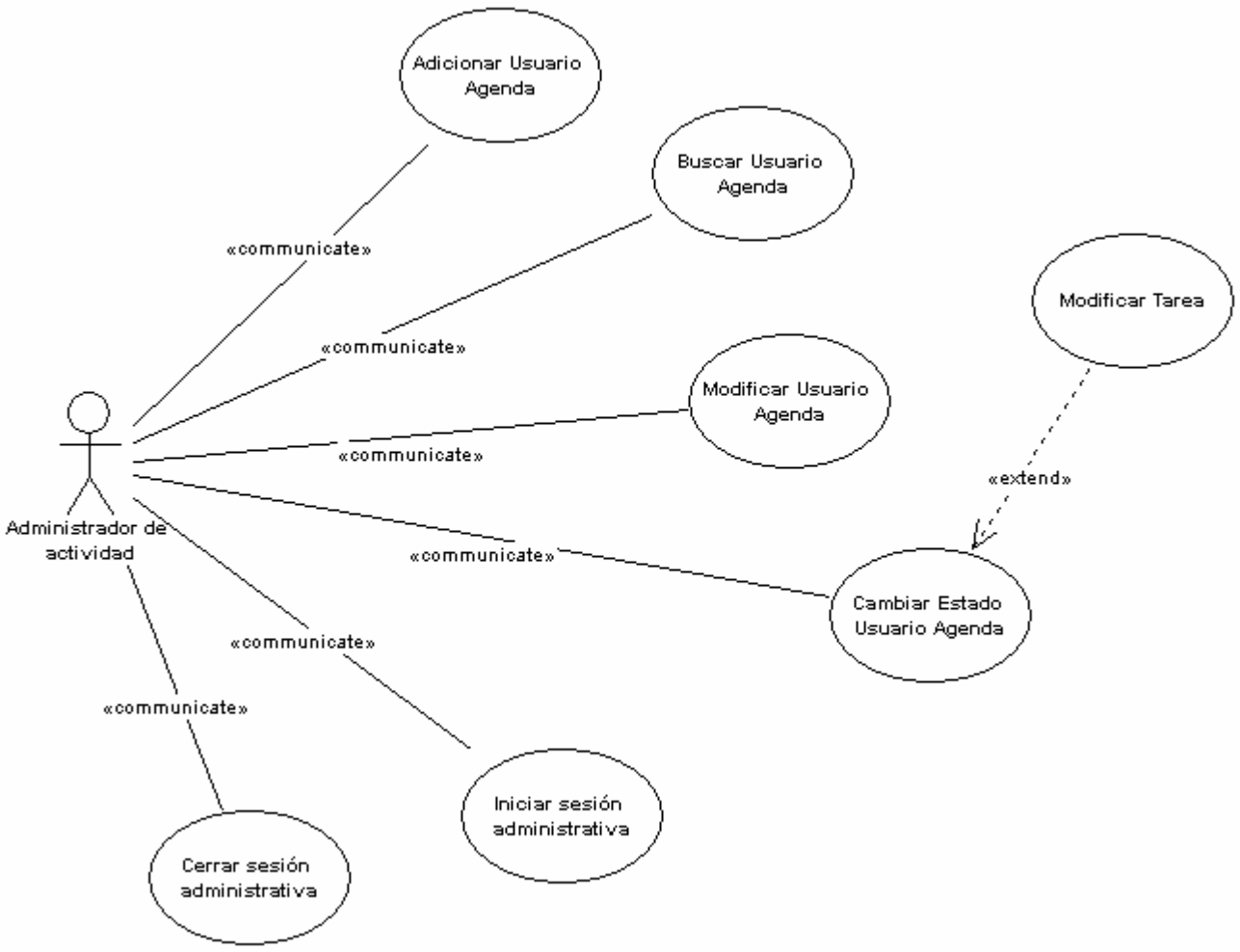
4. El usuario-agenda puede escoger criterios para ordenar el listado de citas, el criterio principal se basa en el tiempo, es decir, filtrar por día, mes, semana o semestre. 5. El usuario-agenda escoge una cita del listado para consultar el detalle de la misma. 3a. Si el usuario-agenda quiere imprimir el listado 1. El usuario-agenda inicia **imprimir archivo** 3b. Si el usuario-agenda quiere almacenar el listado 1. El usuario-agenda inicia **guardar archivo**.
Observaciones
La interfaz principal muestra las tareas, citas y notas próximas a realizarse.

ID	**CASO DE USO 24**
Caso de uso	Registrar avance de tarea
Actor principal	Usuario
Precondiciones	El usuario ha ingresado a la agenda El usuario debe tener tareas asignadas
Poscondiciones	Se actualiza el estado de desarrollo de la tarea
Flujo Básico	
1. El usuario selecciona la tarea que se va modificar. 2. El usuario inicia la actualización del avance del desarrollo de la tarea. 3. El usuario selecciona el porcentaje actual de la tarea acompañado de una descripción y fecha dependiendo de los resultados obtenidos hasta el momento. 4. El sistema registra el avance de desarrollo de la tarea.	
Flujos Alternativos	
1a. Si el usuario selecciona un a tarea cuya fecha de terminación ha expirado 1. El sistema no permite modificar el estado de desarrollo de la tarea. 3a. Si el usuario selecciona un porcentaje de desarrollo inferior al actual 1. El sistema rechaza la acción.	
Observaciones	
El desarrollo de una Tarea puede pasar de 0% a 100% directamente, como también puede desarrollarse de forma gradual. (0% - 25% - 50% - 75% y 100%)	

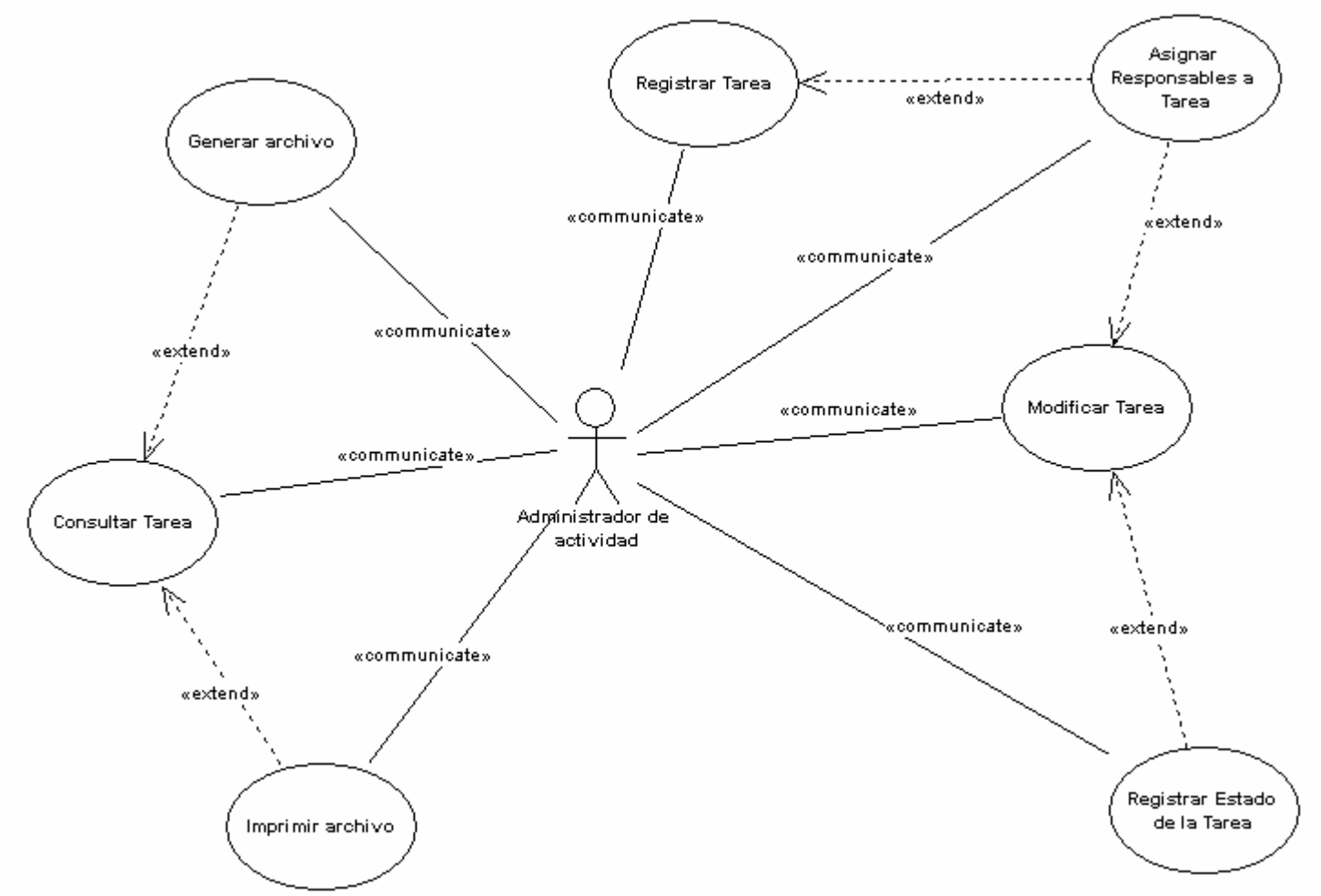
6.1.5 Diagrama de casos de uso.

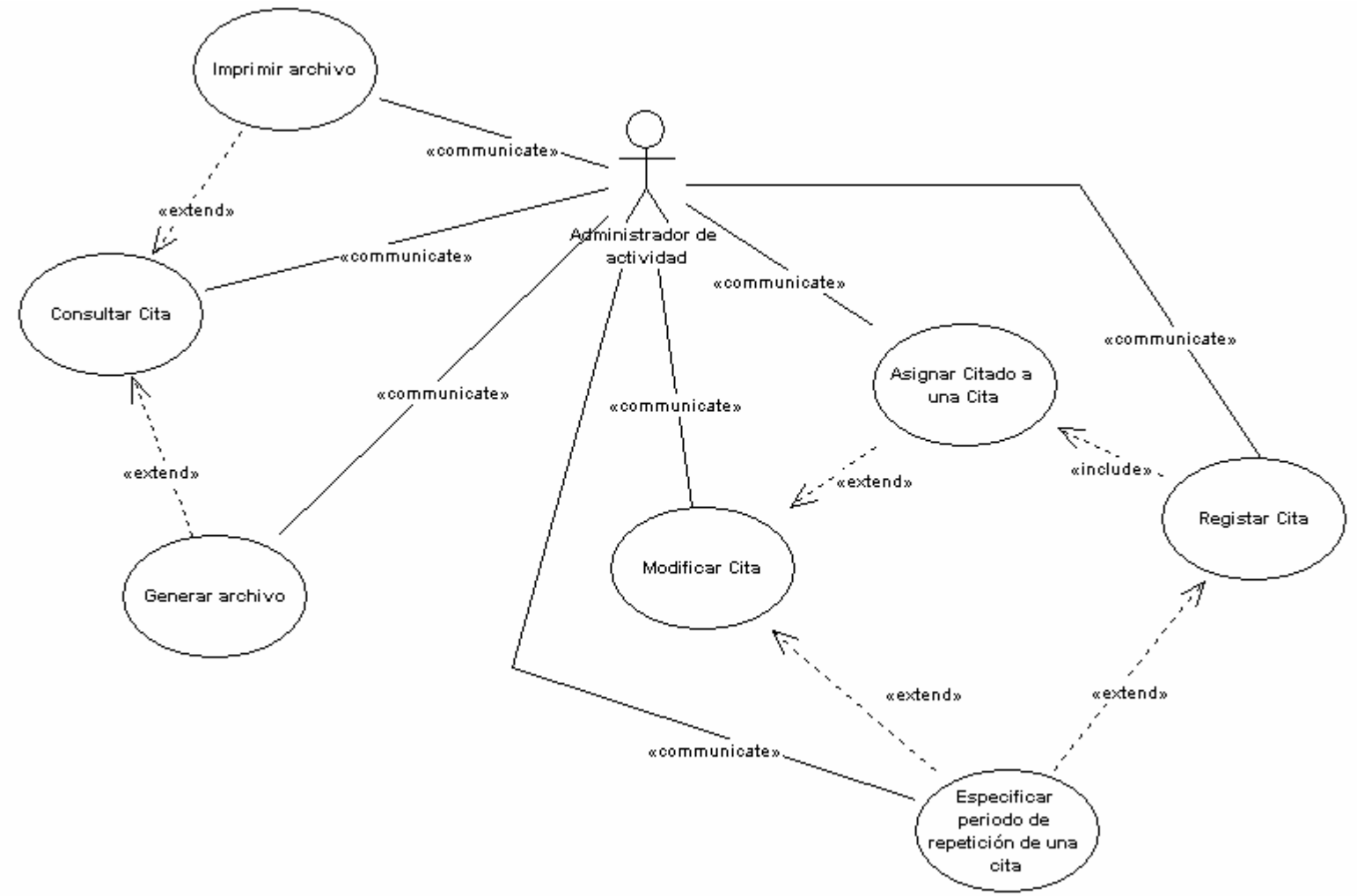
Figura 7. Diagramas de caso de uso

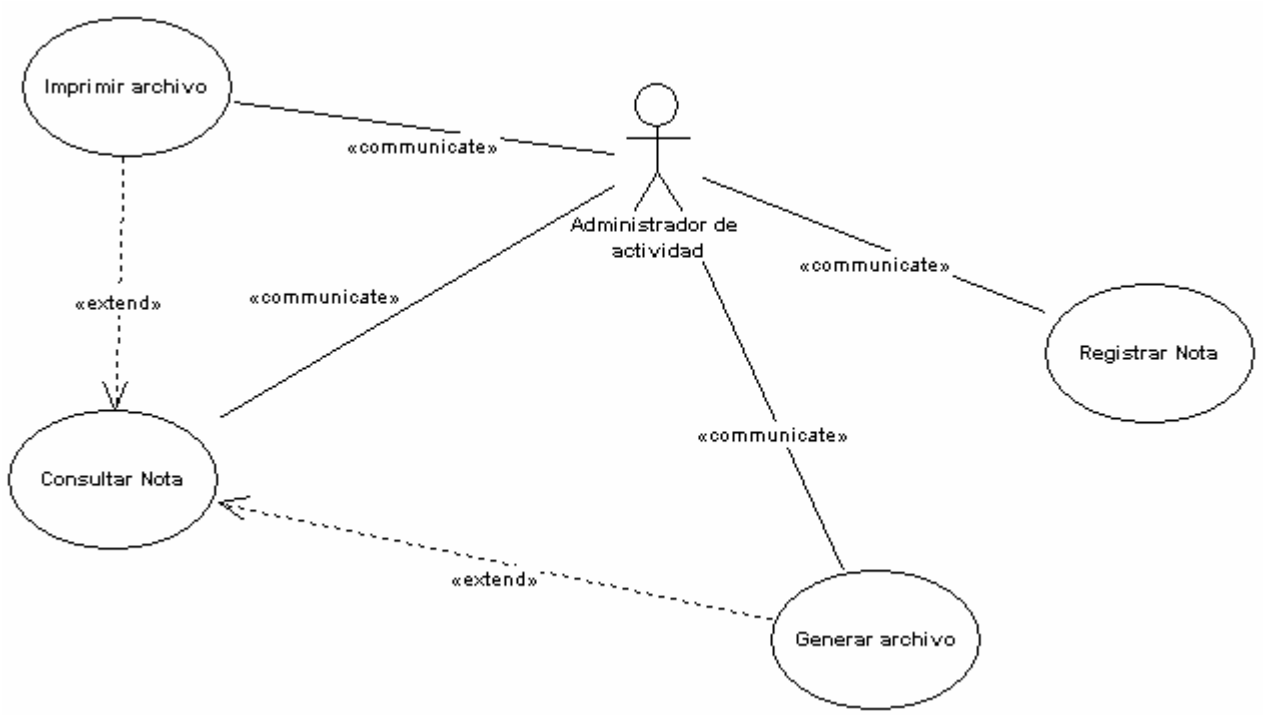
Documento	Título
DCU1	Diagrama de casos de uso - Módulo Administración de la agenda - Paquete Manejo de Usuarios



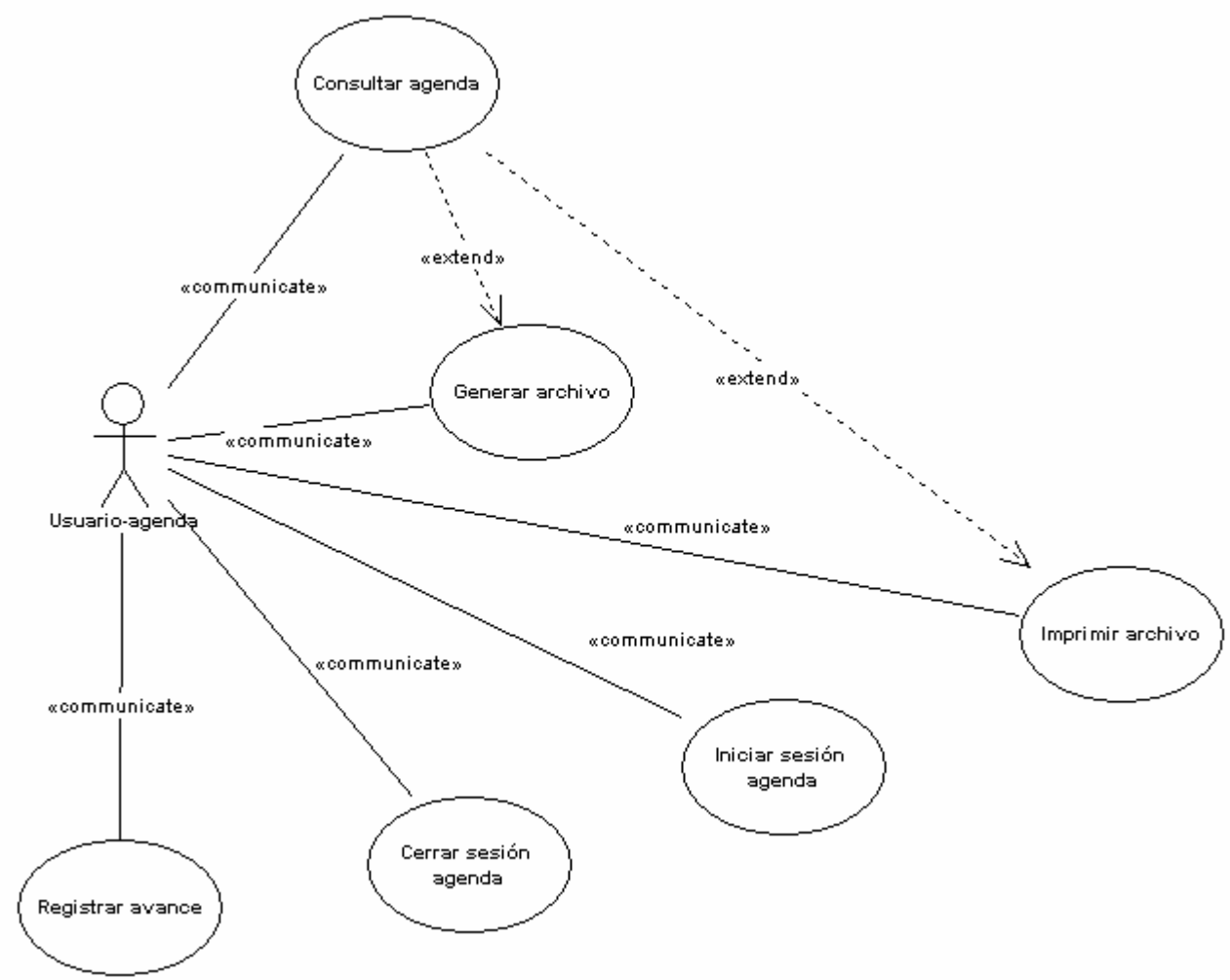
Documento	Título
DCU2	Diagrama de casos de uso - Módulo Administración de la agenda - Paquete Manejo de Actividad




Imprimir archivo
«communicate»
«extend»
Administrador de actividad
«communicate»
Consultar Cita
«communicate»
«communicate»
Asignar Citado a una Cita
«communicate»
«communicate»
«extend»
«include»
Registar Cita
«extend»
Modificar Cita
Generar archivo
«extend»
«extend»
«communicate»
Especificar periodo de repetición de una cita


Imprimir archivo
«communicate»
Administrador de actividad
«communicate»
Registrar Nota
«extend»
«communicate»
«communicate»
Consultar Nota
«extend»
Generar archivo

Documento	Título
DCU3	Diagrama de casos de uso - Módulo Administración de la agenda - Paquete Manejo de la Agenda

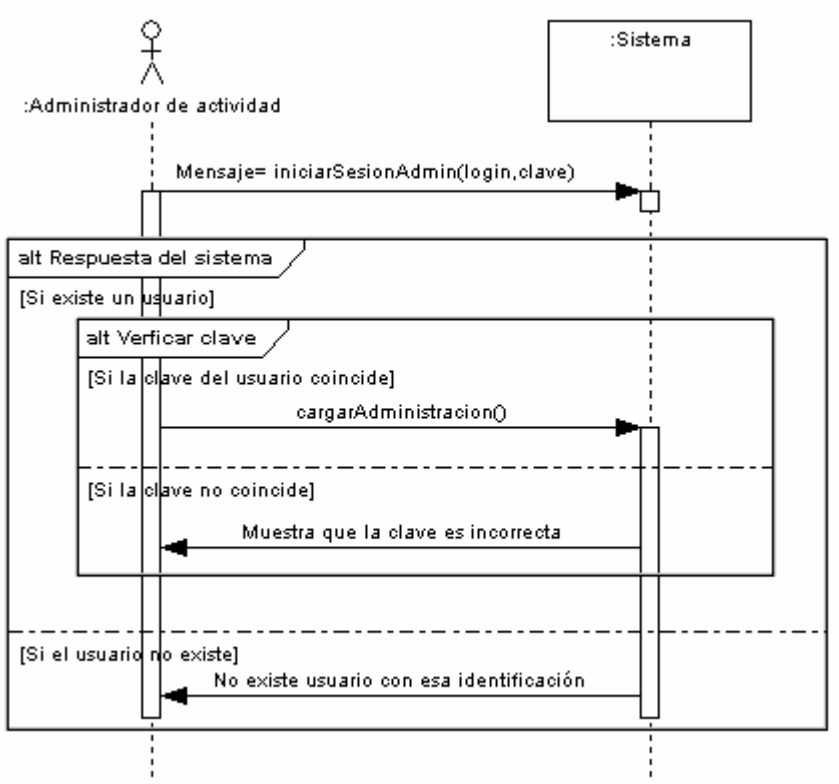


6.1.6 Diagramas de secuencia del sistema

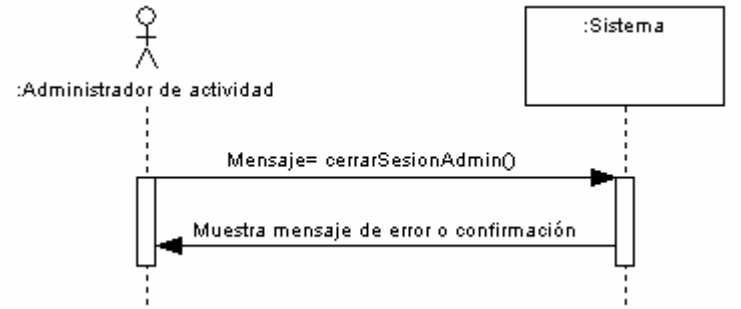
Figura 8. Diagramas de secuencia del sistema

Documento	Título
DSS1	Diagrama de secuencia del sistema - Módulo Administración de la agenda - Paquete Manejo de Usuarios

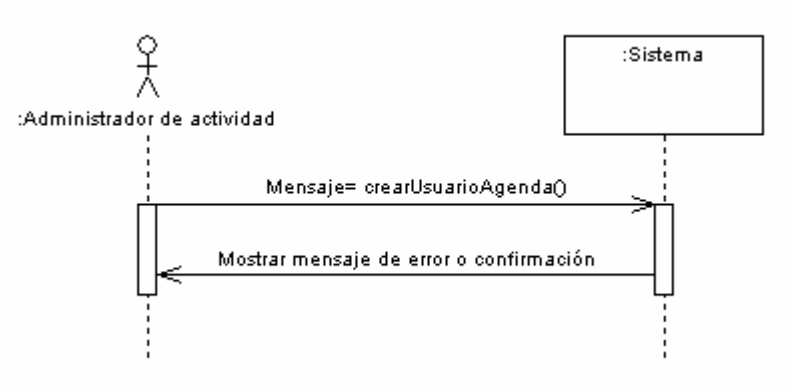
CU1	Iniciar sesión administrativa



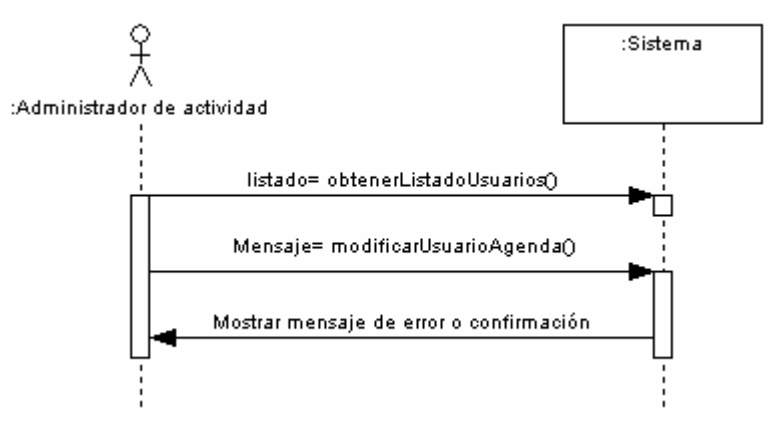
CU2	Cerrar sesión administrativa



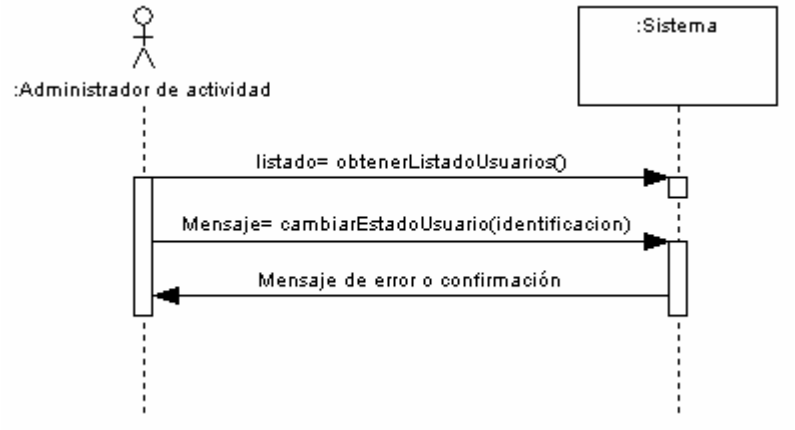
CU3	Adicionar Usuario de la Agenda



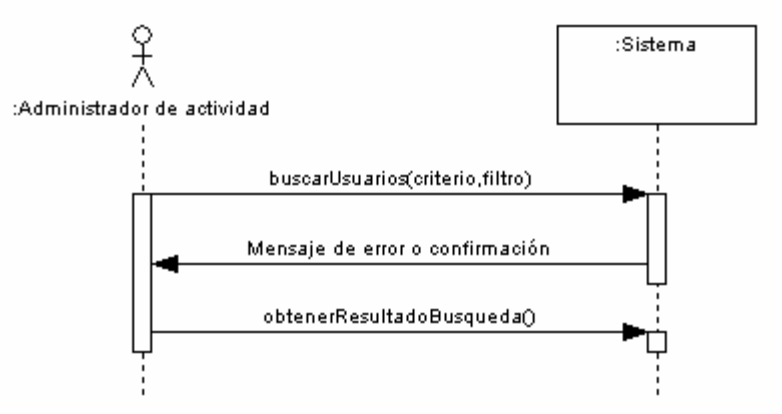
CU4	Modificar Usuario de la Agenda



CU5	Cambiar Estado Usuario de la Agenda

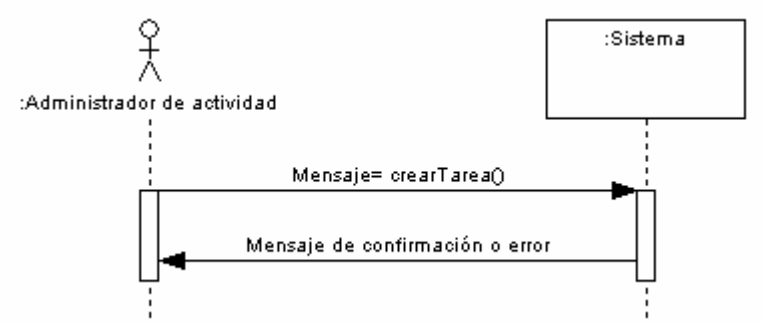


CU6	Buscar Usuario de la Agenda

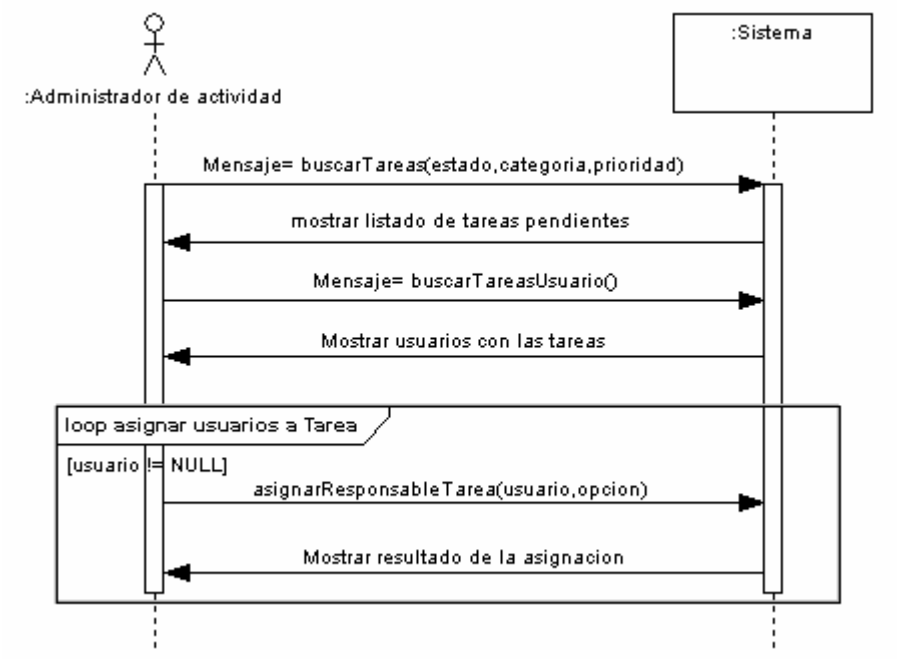


Documento	Título
DSS2	Diagrama de secuencia del sistema - Módulo Administración de la agenda - Paquete Manejo de Actividades

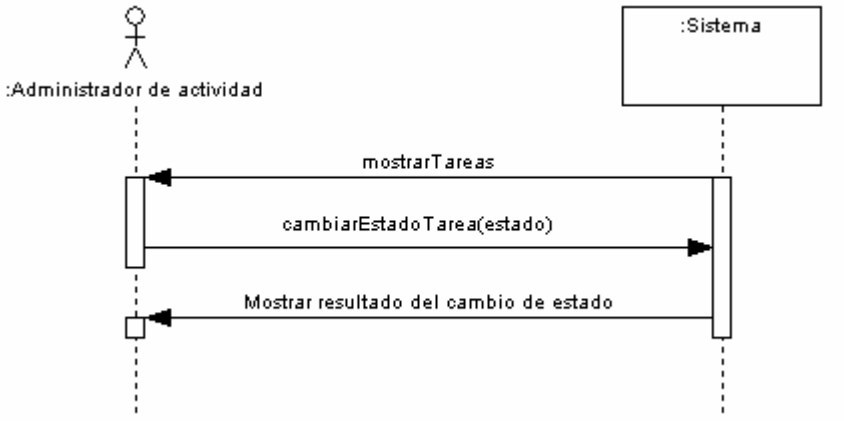
CU7	Registrar Tarea



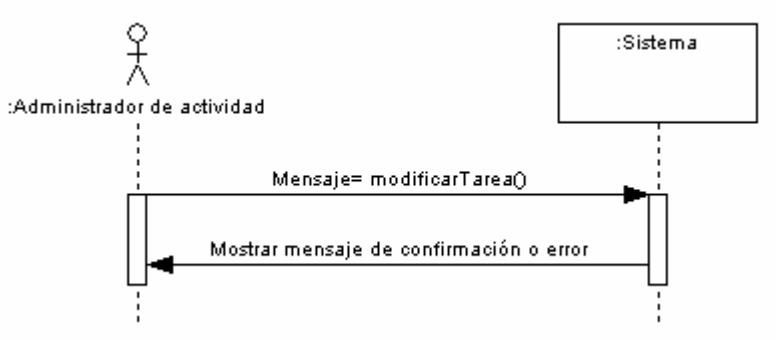
CU8	Asignar Responsables de Tarea



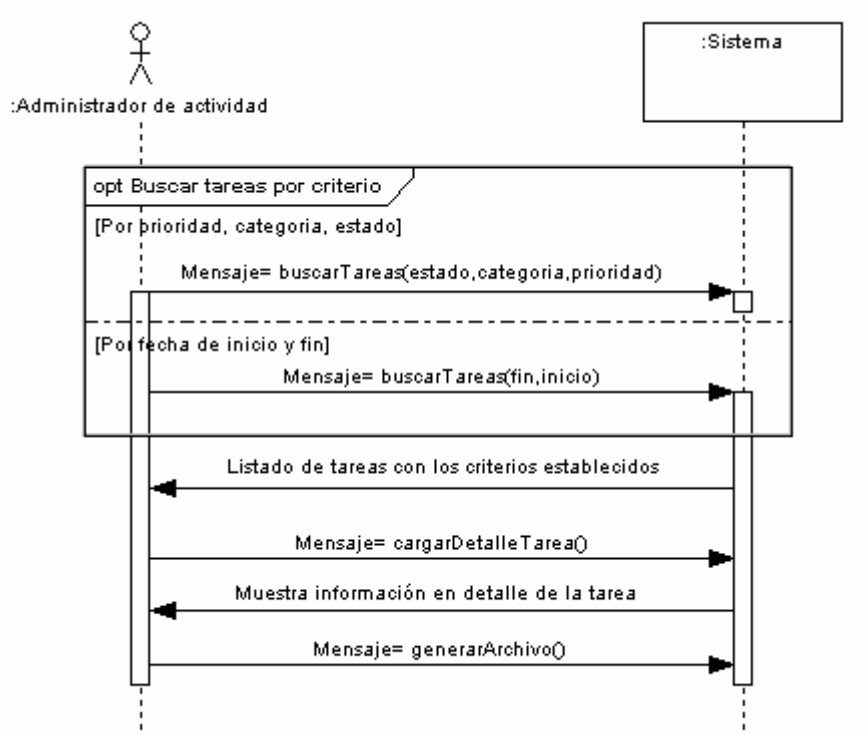
CU9	Registrar estado de la tarea



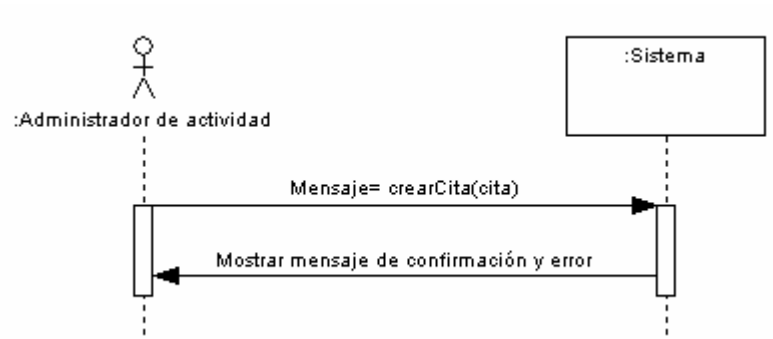
CU10	Modificar tarea



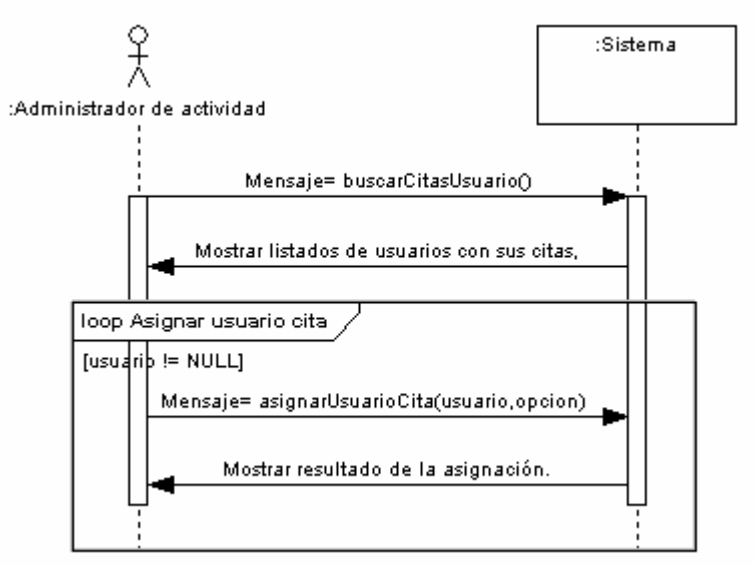
CU11	Consultar tarea



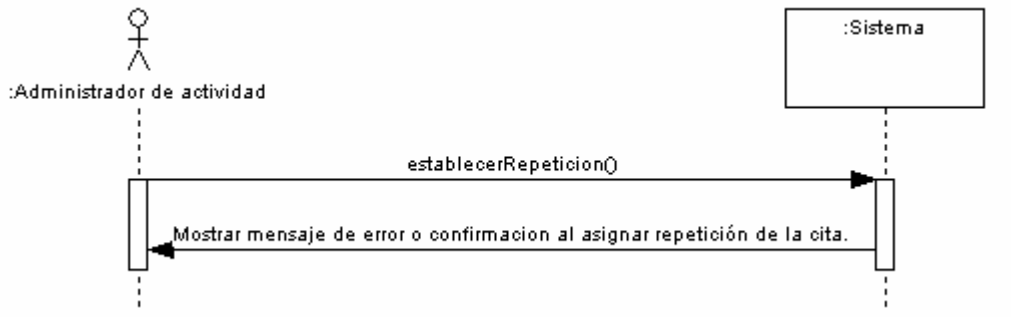
CU12	Registrar cita



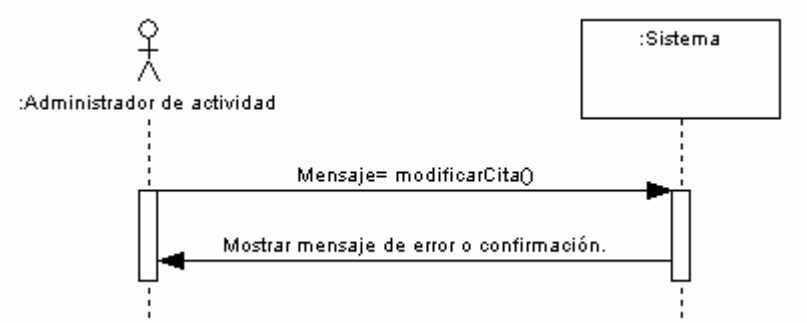
CU13	Asignar citado a una cita



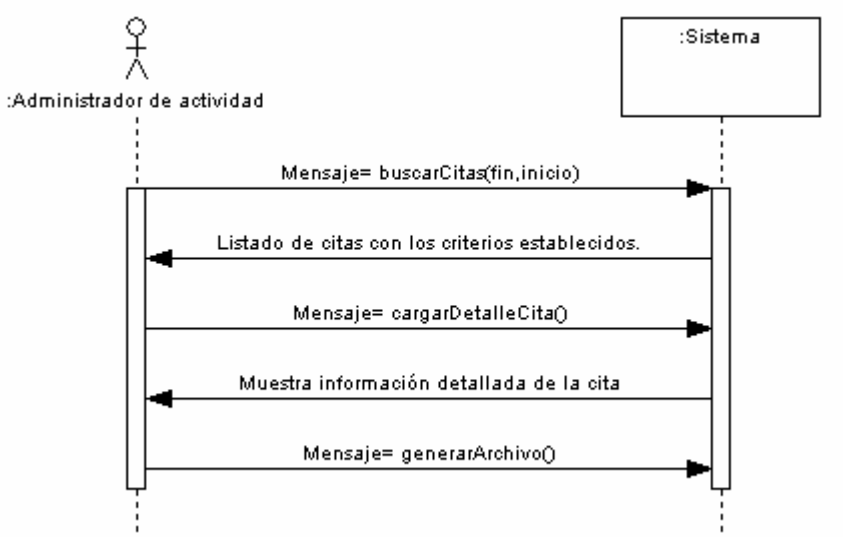
CU14	Especificar periodo de repetición de una cita



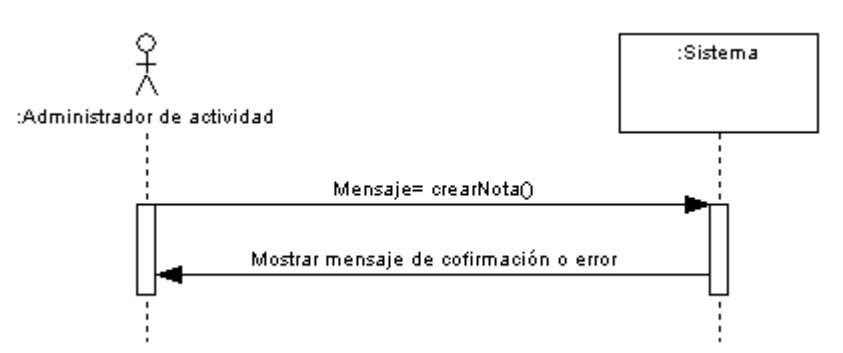
CU15	Modificar cita



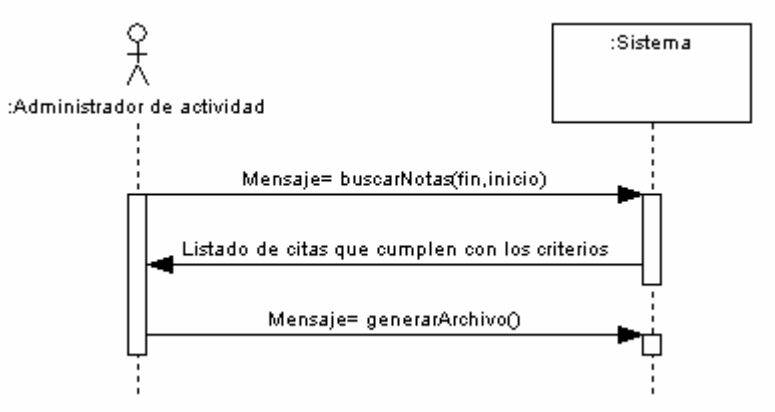
CU16	Consultar cita



CU17	Registrar nota

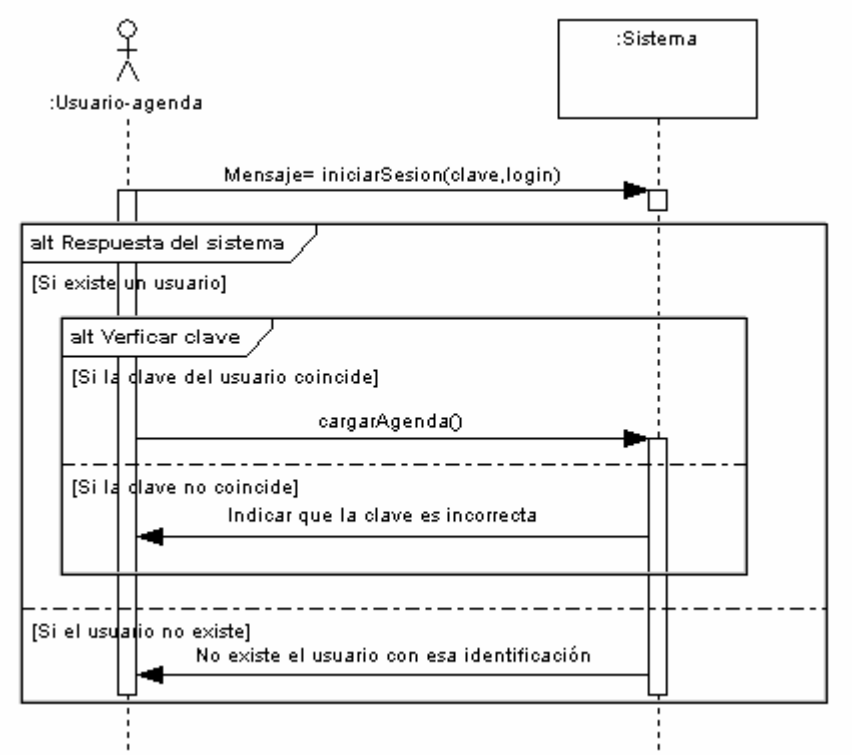


CU18	Consultar Nota

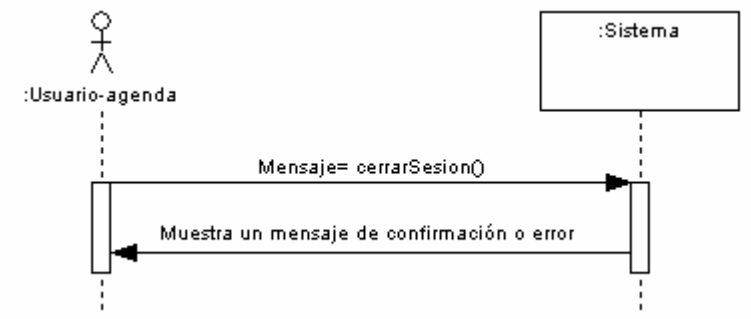


Documento	Título
DSS3	Diagrama de secuencia del sistema - Módulo consulta de la agenda

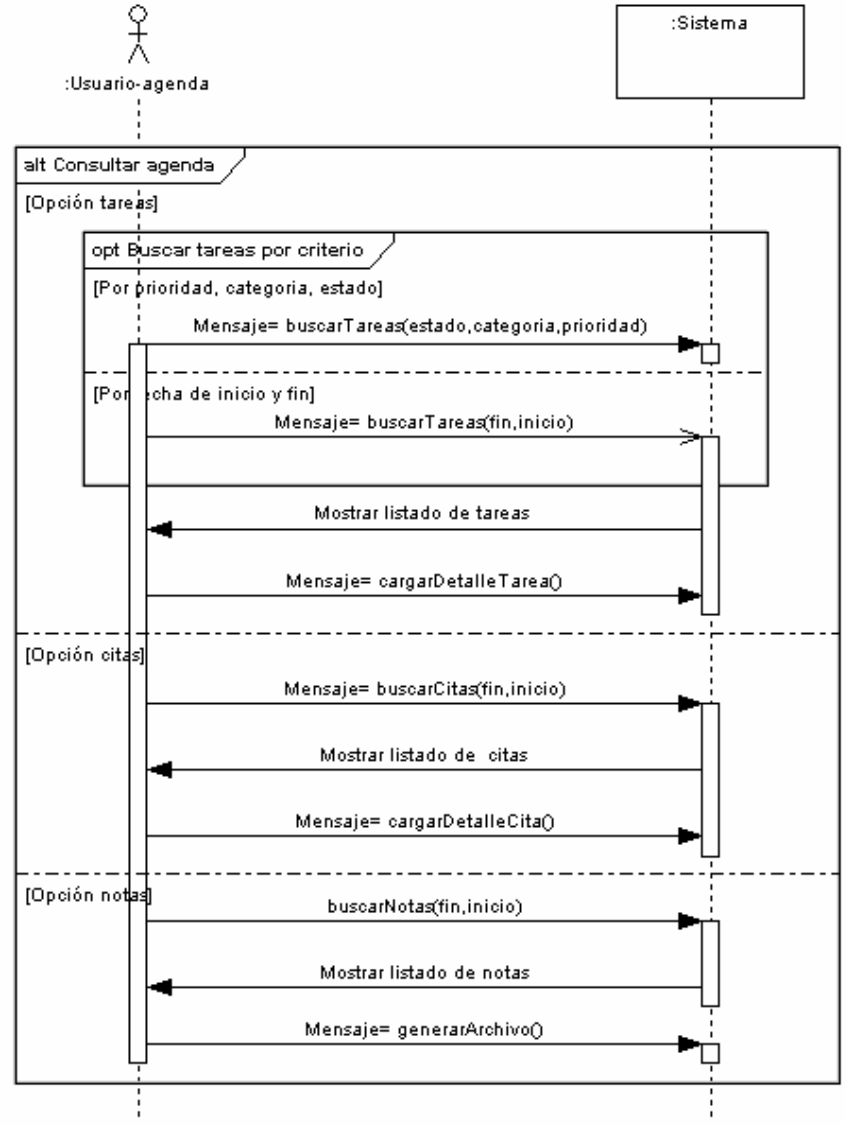
CU21	Iniciar sesión agenda



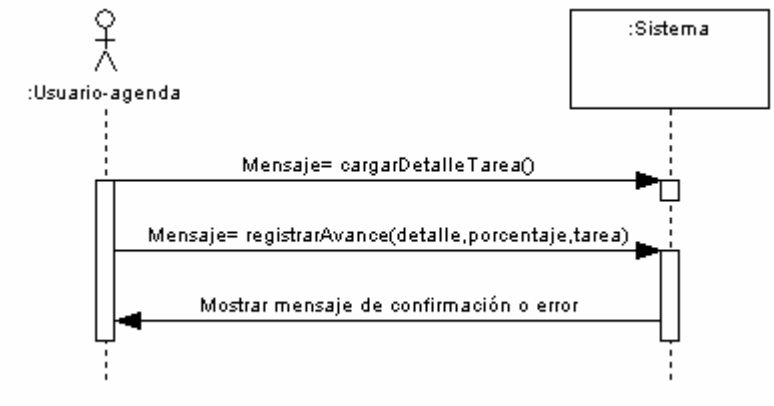
CU22	Cerrar sesión agenda



CU23	Consultar agenda



CU24	Registrar avance de tarea



6.1.7 Contratos de operaciones.

Tabla 6. Tabla de listado de contratos de operaciones

Documento	Título
DCO1	Documento de Contratos de operaciones - Módulo Administración de la agenda - Paquete Manejo de Usuarios

CO1	Mensaje= iniciarSesionAdmin(clave, login)
Referencias cruzadas:	Iniciar sesión administrativa
Precondiciones:	Ninguna
Postcondiciones:	➢ Se crea una instancia de UsuarioAdministrador, llamada usuario ➢ Se especifican todos los atributos de usuario. ➢ Se crea una instancia de SesionAdmin, denominada sesion. ➢ Se establece la fecha de inicio al atributo fechaInicio de sesion. ➢ Se crea una instancia de GestorActividad gActividad ➢ Se crea instancias de GestorTarea (gTarea), GestorCita (gCita) y GestorNota (gNota) ➢ Se asocia gTarea, gCita, y gNota a gActividad ➢ Se asocia usuario, sesion al gActividad ➢ Se crea una instancia de GestorUsuarioAgenda gUsuario

CO2	cargarAdministracion()
Referencias	Iniciar sesión administrativa

cruzadas:	
Precondiciones:	Ninguna
Postcondiciones:	➢ Se crean todas las instancias de usuarios registrados en el sistema ➢ Se asocian todas las instancias de usuarios al listadoUsuario ➢ Se crean todas las instancias de prioridades, categorías, estados y convocantes ➢ Se asocian todas las instancias de prioridades, categorías, estados y convocantes al listadoElemento respectivo ➢ Se crean todas las instancias de tareas, citas y notas ➢ Se asocian todas las instancias de tareas, citas y notas al listadoTareas, listadoCitas y listadoNotas respectivamente.

CO3	Mensaje= cerrarSesionAdmin()
Referencias cruzadas:	Cerrar sesión administrativa
Precondiciones:	Debe existir una instancia de GestorActividad Debe existir una instancia de SesionAdmin, sesion
Postcondiciones:	➢ Se establece la fecha de fin al atributo fechaFin de sesion.

CO4	Mensaje = crearUsuarioAgenda()
Referencias cruzadas:	Adicionar usuario de la agenda
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda
Postcondiciones:	➢ Se creo una instancia de UsuarioAgenda llamada usuario. ➢ Se estableció el contenido de los atributos identificación, nombre, dirección de domicilio, dirección de trabajo, teléfono fijo, teléfono móvil, correo electrónico, nombre de usuario y contraseña a la instancia usuario. ➢ Se asocia la instancia usuario al listadoUsuario.

CO5	Vector= obtenerListadoUsuarios()
Referencias cruzadas:	Modificar usuario de la agenda Cambiar el estado de un usuario de la agenda
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda
Postcondiciones:	➢ Se devuelve la instancia listadoUsuario, con todas las instancias de UsuarioAgenda de los usuarios registrados en el sistema.

CO6	Mensaje= modificarUsuarioAgenda()
Referencias cruzadas:	Modificar usuario de la agenda
Precondiciones:	Debe existir una instancia de UsuarioAgenda que se va a modificar Debe existir una instancia de GestorUsuarioAgenda
Postcondiciones:	➢ Se estableció el contenido de los atributos identificación, nombre, dirección de domicilio, dirección de trabajo, teléfono fijo, teléfono móvil, correo electrónico, nombre de usuario y contraseña a la instancia usuario.

CO7	Mensaje= cambiarEstadoUsuario(identificación)
Referencias cruzadas:	Cambiar el estado de un usuario de la agenda Debe existir una instancia de GestorUsuarioAgenda
Precondiciones:	Debe existir una instancia de UsuarioAgenda cuyo estado se va a cambiar
Postcondiciones:	➢ Se estableció el contenido del atributo estado a la instancia usuario

CO8	Mensaje= busquedaUsuarios(criterio, filtro)
Referencias cruzadas:	Buscar Usuario de la Agenda
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda
Postcondiciones:	➢ Se crean todas las instancias de usuarios registrados en el sistema y que cumplan con el criterio especificado ➢ Se asocian todas las instancias de usuarios al listadoUsuarioActual

CO9	Vector= obtenerResultadoBusqueda()
Referencias cruzadas:	Buscar Usuario de la Agenda
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda
Postcondiciones:	➢ Se devuelve la instancia listadoUsuarioActual, con todas las instancias de UsuarioAgenda de los usuarios registrados en el sistema de acuerdo con los criterios.

Documento	**Título**
DCO2	Documento de Contratos de operaciones - Módulo Administración de la agenda - Paquete Manejo de Actividades

CO10	Mensaje = crearTarea()
Referencias cruzadas:	Registrar tarea

Precondiciones:	Debe existir una instancia de GestorTarea asociada a GestorActividad
Postcondiciones:	➢ Se creo una instancia de Tarea llamada tarea. ➢ Se estableció el contenido de los atributos categoría, estado, prioridad, descripción, fecha de inicio, fecha de fin, y tema ➢ Se asocia la instancia tarea al listadoTarea.

CO11	Mensaje= buscarTareas(prioridad, categoría, estado)
Referencias cruzadas:	Asignar Responsables de Tarea Consultar agenda
Precondiciones:	Debe existir una instancia de GestorActividad o una instancia de GestorAgenda
Postcondiciones:	➢ Se crean todas las instancias de tareas registradas en el sistema y que cumplan con los criterios establecidos. ➢ Se asocian todas las instancias de tareas al listadoTareasActual.

CO12	buscarTareasUsuario()
Referencias cruzadas:	Asignar Responsables de Tarea
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda Deben existir todas las instancias de UsuarioAgenda registrados en el sistema.
Postcondiciones:	➢ Se crean todas las instancias de categorías, estados y prioridades. ➢ Se asocian todas las instancias de categorías, estados y prioridades al listadoCategoría, listadoEstado y listadoPrioridad respectivamente. ➢ Se crean todas las instancias de tareas registradas en el sistema y que pertenezcan al usuario que se esta trabajando. ➢ Se asocian todas las instancias de tareas al listadoTarea ➢ Se asocia el arreglo de instancias tareas al usuario actual.

CO13	Mensaje = asignarResponsableTarea(usuario, opcion)
Referencias cruzadas:	Asignar Responsables de Tarea
Precondiciones:	Debe existir una instancia de GestorTarea asociada a GestorActividad
Postcondiciones:	➢ Se asocia la instancia usuario a la instancia tareaActual.

CO14	cambiarEstadoTarea(estado)
Referencias	Registrar estado de la tarea

cruzadas:	
Precondiciones:	Debe existir una instancia de GestorTarea asociada a GestorActividad Debe existir una instancia de Tarea cuyo estado va ha cambiar.
Postcondiciones:	➢ Se estableció el contenido del atributo estado a la instancia usuario.

CO15	Mensaje= modificarTarea()
Referencias cruzadas:	Modificar tarea
Precondiciones:	Debe existir una instancia de GestorTarea asociada a GestorActividad Debe existir una instancia de Tarea, tareaActual, que se va ha modificar.
Postcondiciones:	➢ Se estableció el contenido de los atributos categoría, estado, prioridad, descripción, fecha de inicio, fecha de fin, y tema a la instancia tarea.

CO16	Mensaje= buscarTareas(fin, inicio)
Referencias cruzadas:	Consultar tarea Consultar agenda
Precondiciones:	Debe existir una instancia de GestorActividad o una instancia de GestorAgenda
Postcondiciones:	➢ Se crean todas las instancias de tareas registradas en el sistema y que cumplan con los criterios establecidos. ➢ Se asocian todas las instancias de tareas al listadoTareasActual.

CO17	Mensaje= cargarDetalleTarea()
Referencias cruzadas:	Consultar tarea Consultar agenda Registrar avance de tarea
Precondiciones:	Debe existir una instancia de GestorTarea asociada a GestorActividad o a GestorAgenda Debe existir una instancia de Tarea, tareaActual, cuyos detalles se van a cargar.
Postcondiciones:	➢ Se crea cada instancia de usuario que figuran como responsables de la tarea ➢ Se asocia cada instancia de usuario al listadoUsuarios ➢ Se asocia el listadoUsuarios a la tareaActual

CO18	Mensaje= generarArchivo()
Referencias cruzadas:	Consultar tarea Consultar agenda
Precondiciones:	Debe existir una instancia de GestorRTF y GestorPDF asociada a GestorTarea y esta a su vez asociada a GestorActividad o GestorAgenda
Postcondiciones:	➢ Se crea una instancia Tabla para construir la tabla del reporte ➢ Se crean instancias de Celda con la información de cada tarea del reporte ➢ Se asocia a la instancia de Tabla todas las Celdas para construir la tabla. ➢ Se asocia la instancia Tabla a la instancia de GestorPDF y Gestor RTF

CO19	Mensaje= crearCita()
Referencias cruzadas:	Registrar cita
Precondiciones:	Debe existir una instancia de GestorCita asociada a GestorActividad
Postcondiciones:	➢ Se creo una instancia de Cita o CitaDia, o CitaSemana o CitaMes llamada cita, dependiendo de lo que se necesita. ➢ Se estableció el contenido de los atributos fecha de la cita, fecha de publicación, duración, prioridad, asunto, observación, convocante y tipo. ➢ Se asocia la instancia cita al listadoCita.

CO20	buscarCitasUsuario()
Referencias cruzadas:	Asignar citados a una cita
Precondiciones:	Debe existir una instancia de GestorUsuarioAgenda Deben existir todas las instancias de UsuarioAgenda registrados en el sistema.
Postcondiciones:	➢ Se crean todas las instancias de prioridades. ➢ Se asocian todas las instancias de prioridades al listadoPrioridad respectivamente. ➢ Se crean todas las instancias de citas registradas en el sistema y que pertenezcan al usuario que se esta trabajando. ➢ Se asocian todas las instancias de citas al listadoCita. ➢ Se asocia el arreglo de instancias cita al usuario actual.

CO21	Mensaje = asignarUsuarioCita(usuario, opcion)
Referencias cruzadas:	Asignar citados a una cita
Precondiciones:	Debe existir una instancia de GestorCita asociada a GestorActividad Debe existir una instancia de Cita, llamada citaActual
Postcondiciones:	➢ Se asocia la instancia usuario a la instancia citaActual.

CO22	Mensaje = establecerRepeticion()
Referencias cruzadas:	Especificar periodo de repetición de una cita
Precondiciones:	Debe existir una instancia de GestorCita asociada a GestorActividad Debe existir una instancia de Cita, llamada citaActual
Postcondiciones:	➢ Se estableció los atributos propios de CitaDia, CitaSemana o CitaMes, correspondientes dependiendo del tipo de cita.

CO23	Mensaje= modificarCita()
Referencias cruzadas:	Modificar cita
Precondiciones:	Debe existir una instancia de GestorCita asociada a GestorActividad Debe existir una instancia de Cita, citaActual que se va ha modificar.
Postcondiciones:	➢ Se estableció el contenido de los atributos fecha de la cita, fecha de publicación, duración, prioridad, asunto, observación, convocante y tipo. ➢ Se estableció los atributos propios de CitaDia, CitaSemana o CitaMes, correspondientes dependiendo del caso.

CO24	Mensaje= buscarCitas(fin, inicio)
Referencias cruzadas:	Consultar cita Consultar agenda
Precondiciones:	Debe existir una instancia de GestorActividad o una instancia de GestorAgenda.
Postcondiciones:	➢ Se crean todas las instancias de citas registradas en el sistema y que cumplan con los criterios establecidos. ➢ Se asocian todas las instancias de citas al listadoCitasActual.

CO25	Mensaje= cargarDetalleCita()

Referencias cruzadas:	Consultar cita Consultar agenda
Precondiciones:	Debe existir una instancia de GestorCita asociada a GestorActividad o a GestorAgenda Debe existir una instancia de cita, citaActual, cuyos detalles se van a cargar.
Postcondiciones:	➢ Se crea cada instancia de usuario que están citados a la cita. ➢ Se asocia cada instancia de usuario al listadoUsuarios ➢ Se asocia el listadoUsuarios a la citaActual.

CO26	Mensaje= crearNota()
Referencias cruzadas:	Registrar nota
Precondiciones:	Debe existir una instancia de GestorNota asociada a GestorActividad
Postcondiciones:	➢ Se creo una instancia de Nota nota, dependiendo de lo que se necesita. ➢ Se estableció el contenido de los atributos asunto, fecha de la nota, duración y observación. ➢ Se asocia la instancia nota al listadoNota.

CO27	buscarNotas(fin, inicio)
Referencias cruzadas:	Consultar notas Consultar agenda
Precondiciones:	Debe existir una instancia de GestorActividad o una instancia de GestorAgenda
Postcondiciones:	➢ Se crean todas las instancias de notas registradas en el sistema y que cumplan con los criterios establecidos. ➢ Se asocian todas las instancias de notas al listadoNotasActual.

Documento	**Título**
DCO3	Documento de Contratos de Operaciones - Módulo consulta de la agenda

CO28	Mensaje= iniciarSesion(clave, login)
Referencias cruzadas:	Iniciar sesión agenda
Precondiciones:	Ninguna

Postcondiciones:	➢ Se crea una instancia de UsuarioAgenda, usuario ➢ Se establece el contenido de los atributos identificación, nombre, dirección de domicilio, dirección de trabajo, teléfono fijo, teléfono móvil, correo electrónico, nombre de usuario y contraseña a la instancia usuario. ➢ Se crea una instancia de Sesion, denominada sesion. ➢ Se establece la fecha de inicio al atributo fechaInicio de sesion. ➢ Se crea instancias de GestorTarea (gTarea), GestorCita (gCita) y GestorNota (gNota) ➢ Se crea una instancia de Agenda, denominada agenda ➢ Se asocia gTarea, gCita, y gNota a agenda. ➢ Se crea una instancia de GestorAgenda llamada gAgenda ➢ Se asocia usuario, sesion y agenda al gAgenda

CO29	cargarAgenda()
Referencias cruzadas:	Iniciar sesión agenda
Precondiciones:	Debe existir una instancia de GestorAgenda
Postcondiciones:	➢ Se crean todas las instancias de prioridades, categorías, estados y convocantes relacionadas con las actividades del usuario ➢ Se asocian todas las instancias de prioridades, categorías, estados y convocantes al listadoElemento respectivo ➢ Se crean todas las instancias de tareas, citas y notas relacionadas con el usuario y que estén pendientes ➢ Se asocian todas las instancias de tareas, citas y notas al listadoTareas, listadoCitas y listadoNotas respectivamente. ➢ Se asocian las instancias de listadoTareas, listadoCitas y listadoNotas, así como también listadoPrioridad, listadoCategoria y listadoEstado a cada gestor respectivo.

CO30	Mensaje= cerrarSesion()
Referencias cruzadas:	Cerrar sesión agenda
Precondiciones:	Debe existir una instancia de GestorAgenda Debe existir una instancia de Sesion, llamada sesion
Postcondiciones:	➢ Se establece la fecha de fin al atributo fechaFin de sesion.

CO31	Mensaje= registrarAvance(detalle, porcentaje, tarea)
Referencias cruzadas:	Registrar avance de tarea
Precondiciones:	Debe existir una instancia de Agenda asociada a

	GestorAgenda Debe existir una instancia de Tarea, tareaActual a la cual se va a registrar el avance
Postcondiciones:	➢ Se crea una instancia de AvanceTarea llamada avance ➢ Se llenan los atributos de avance como fecha, porcentaje y descripción. ➢ Se asocia avance a tareaActual.

6.2 MODELO DE ANÁLISIS

6.2.1 Listado de conceptos.

Tabla 7. Tabla de listado de conceptos

Concepto	Descripción
UsuarioAgenda	Para manipular toda la información relacionada con los usuarios de la agenda
GestorUsuarioAgenda	Para realizar todas las operaciones relacionadas con los usuarios de la agenda, adición, modificación, cambio de estado y consulta.
ListadoUsuarioAgenda	Permite manipular el listado de usuarios cuando sea necesario en las operaciones.
UsuarioAdmin	Para manipular toda la información relacionada con los usuarios administrativos.
GestorActividad	Permite realizar todas las operaciones relacionadas con la administración de tareas, citas y notas.
SesionAgenda	Permite manipular toda la información relacionada con las sesiones iniciadas por el usuario administrador.
Tarea	Para manipular toda la información relacionada con las tareas de los usuarios.
GestorTarea	Para realizar todas las operaciones relacionadas con las tareas como la adición, modificación, cambio de estado y consulta.
ListadoTarea	Permite manipular el listado de tareas cuando sea necesario en las operaciones.
Cita	Para manipular toda la información relacionada con las citas de los usuarios
CitaDia	Permite manipular tanto la información de la cita como datos sobre la repetición diaria.
CitaSemana	Permite manipular tanto la información de la cita como datos sobre la repetición semanal.
CitaMes	Permite manipular tanto la información de la cita como

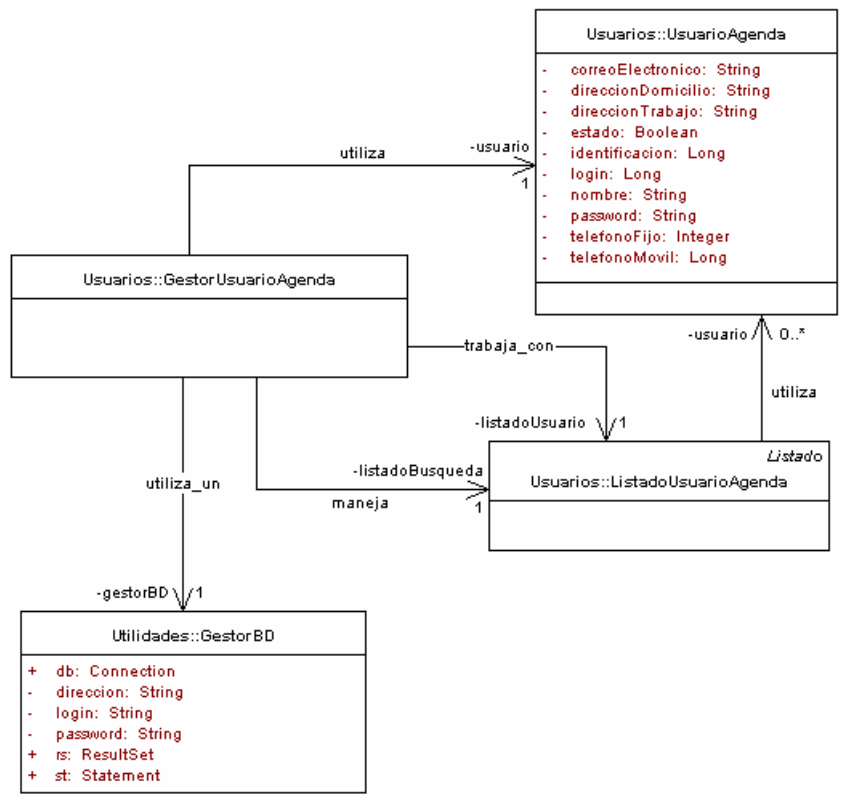
	datos sobre la repetición mensual.
GestorCita	Parar realizar todas las operaciones relacionadas con las citas como la adición, modificación y consulta de citas
ListadoCita	Permite manipular el listado de citas cuando sea necesario en las operaciones.
Conovocante	Permite manejar toda la información de las personas o entidades que pueden citar a reuniones.
ListadoConvocante	Permite manipular el listado de tareas cuando sea necesario en las operaciones.
Nota	Para manipular toda la información relacionada con las notas de los usuarios.
GestorNota	Para realizar todas las operaciones relacionadas con las notas como la adición, modificación, eliminación y consulta.
ListadoNota	Permite manipular el listado de notas cuando sea necesario en las operaciones.
Agenda	Se encarga de realizar todas las operaciones relacionadas con el manejo de la agenda.
GestorAgenda	Se encarga de realizar todas las operaciones relacionadas con el inicio de sesión en la agenda.
SesionAgenda	Permite manipular toda la información relacionada con las sesiones iniciadas por el usuario.
Elemento	Permite manipular toda la información relacionada con conceptos elementales, por lo general que tienen un código un nombre como categoría, estado y prioridad.
ListadoElemento	Permite manipular el listado de elementos cuando sea necesario para las operaciones
GestorGeneral	Permite realizar todas las operaciones relacionadas con tareas, citas y notas a un nivel común.
GestorPDF	Permite realizar el archivo con extensión PDF, cuando sea necesario para reportes.
GestorRTF	Permite realizar el archivo con extensión RTF, cuando sea necesario para reportes.
Mensaje	Permite manejar los mensajes al usuario, como advertencias, errores o información general.
GestorBD	Permite realizar todas las operaciones relacionadas con la base de datos.

6.2.2 Diagrama de clases del análisis.

Figura 9. Diagramas de clases del análisis

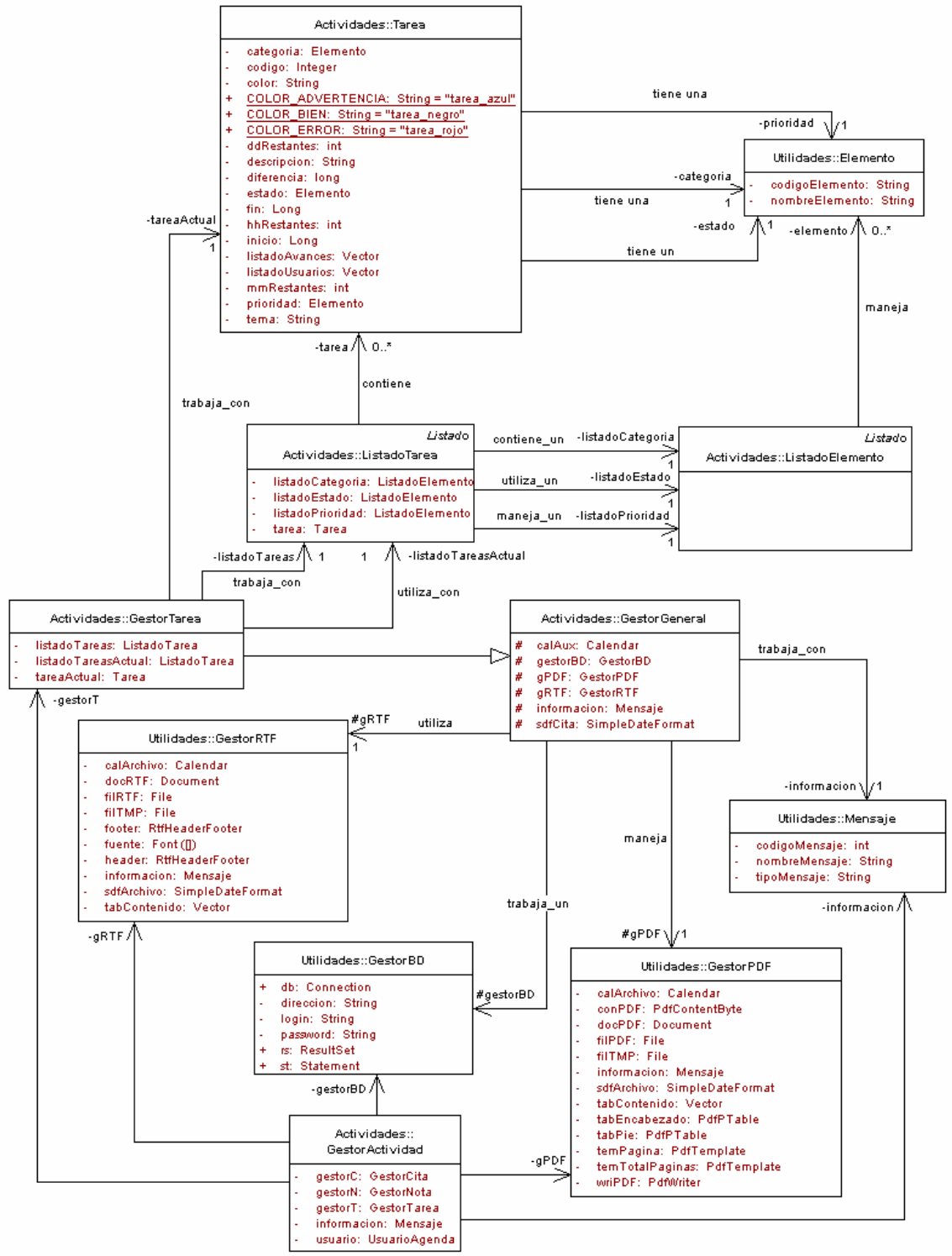
Documento	Título
DC1	Diagrama de clases - Módulo Administración de la agenda -

	Paquete Manejo de Usuarios

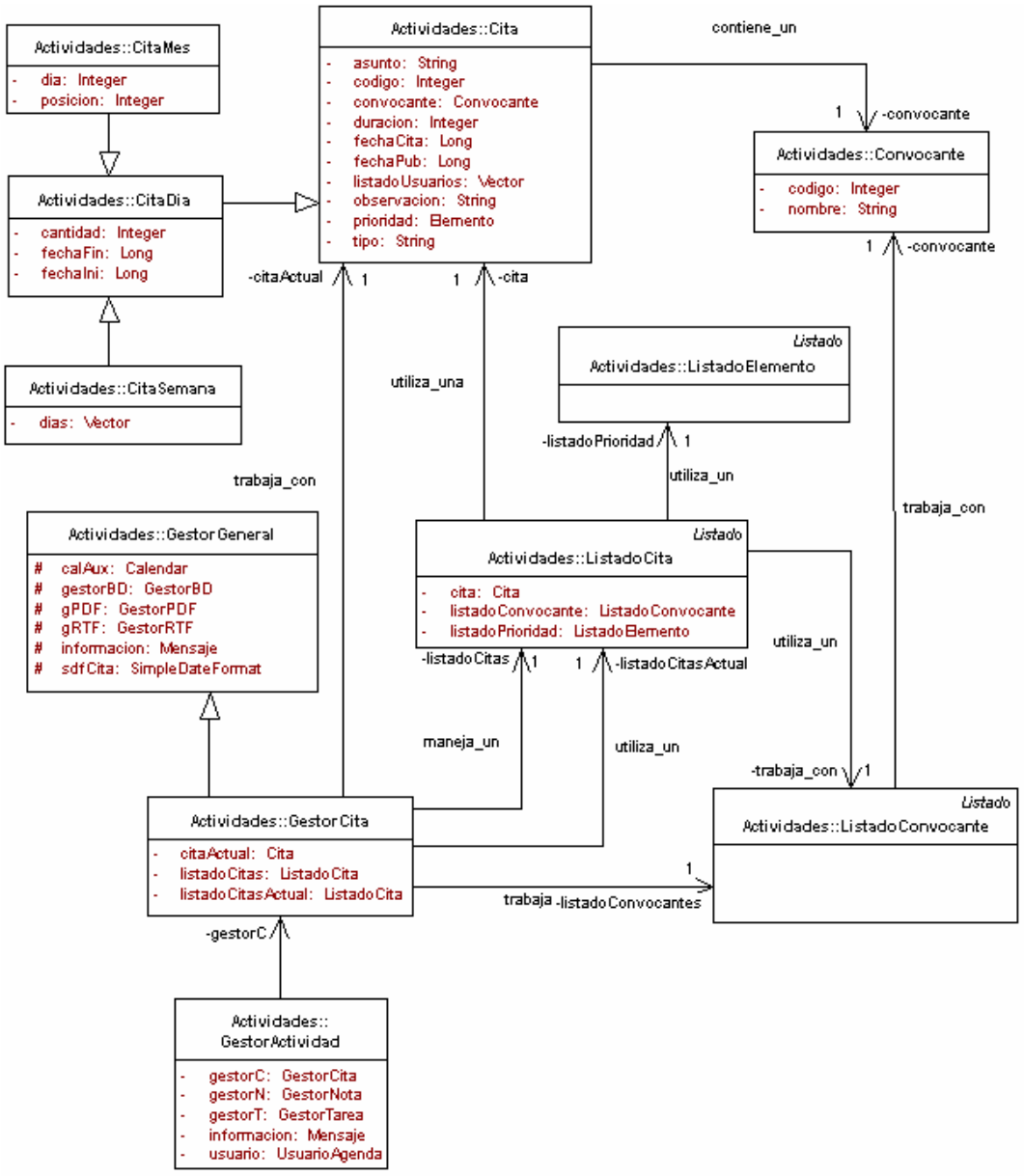


Documento	Título
DC2	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades

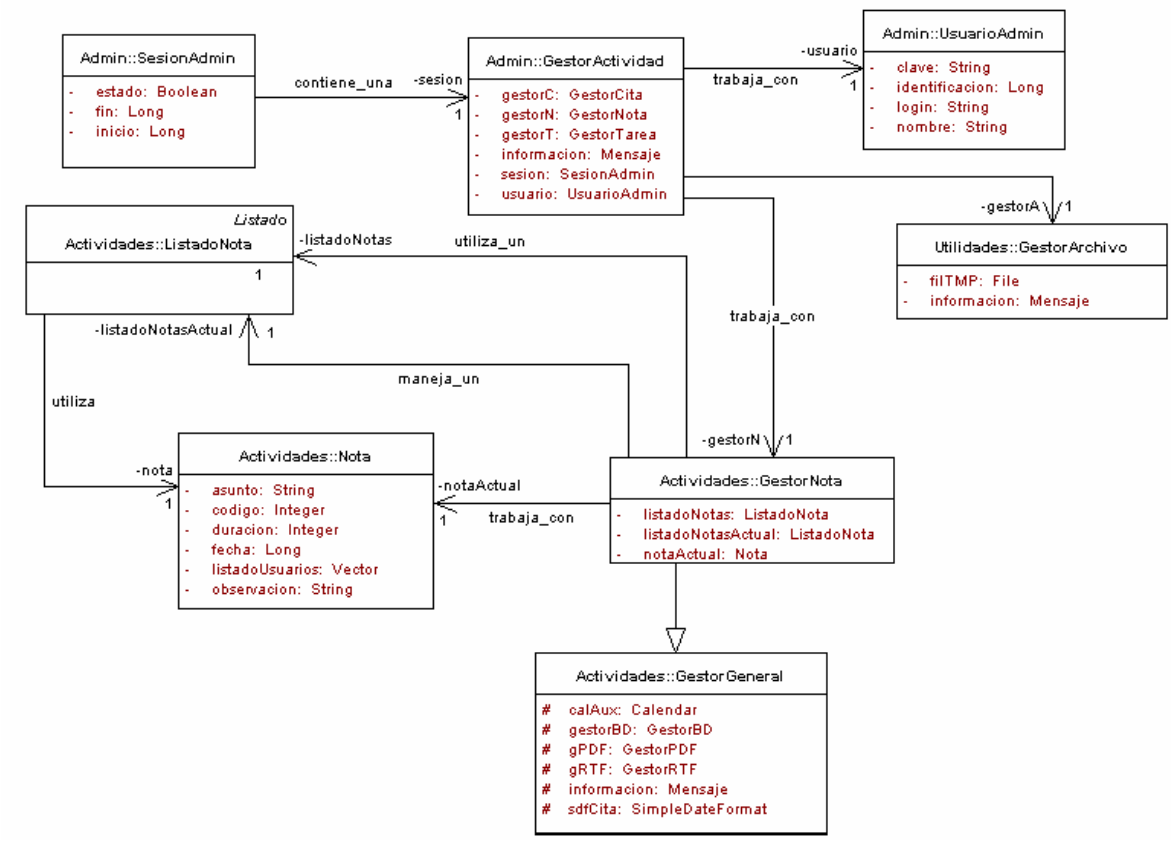
Documento	Título
DC21	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades – Gestión de tareas



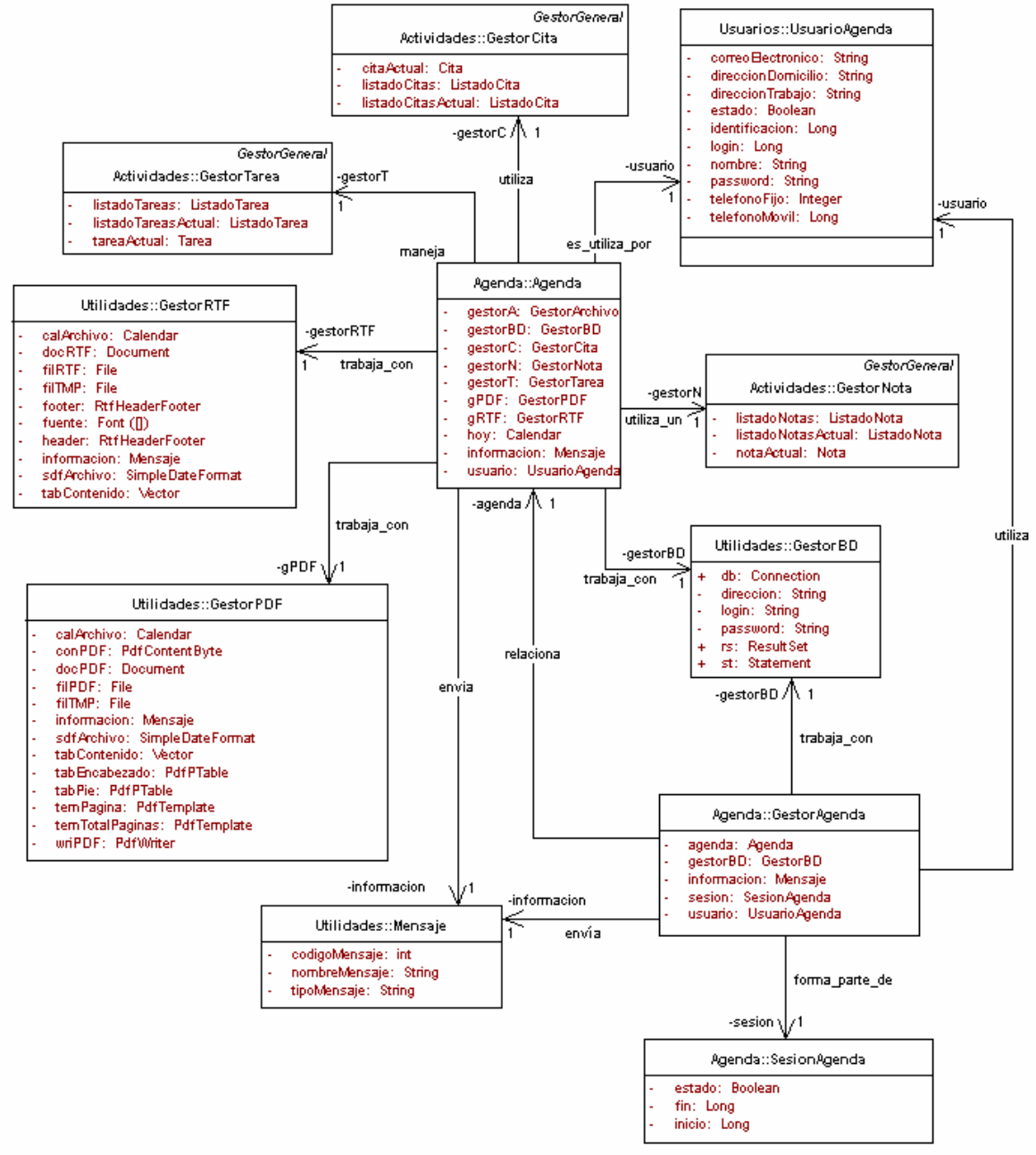
Documento	Título
DC22	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades – Gestión de citas



Documento	Título
DC23	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades – Gestión de notas



Documento	Título
DC3	Diagrama de clases - Módulo consulta de la agenda



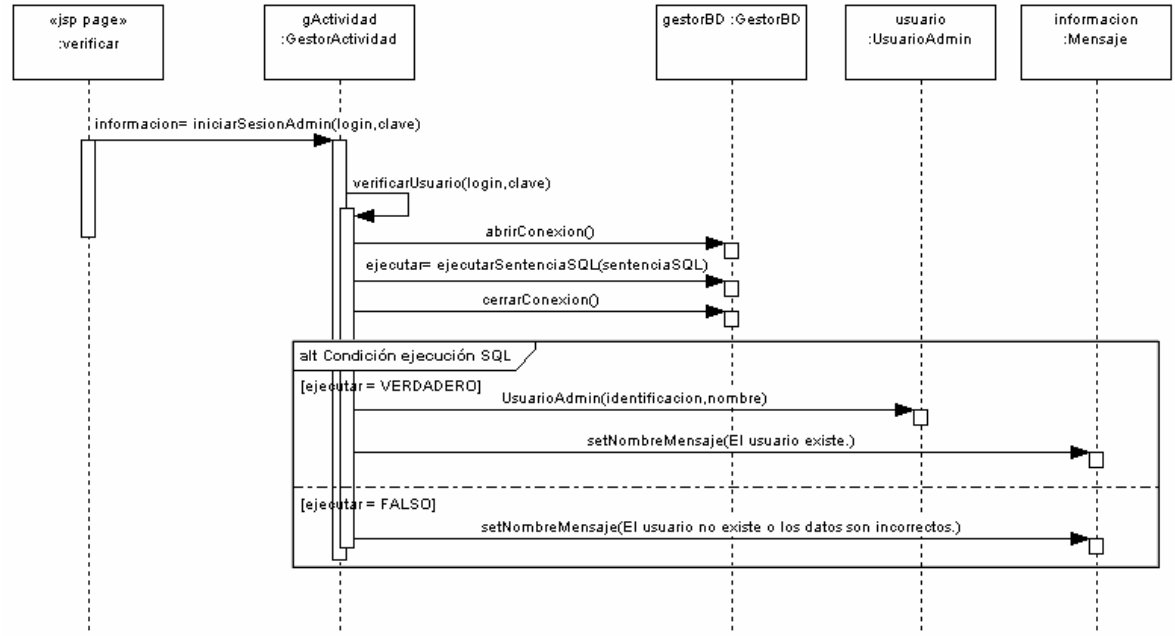
6.3 MODELO DE DISEÑO

6.3.1 Diagramas de secuencia.

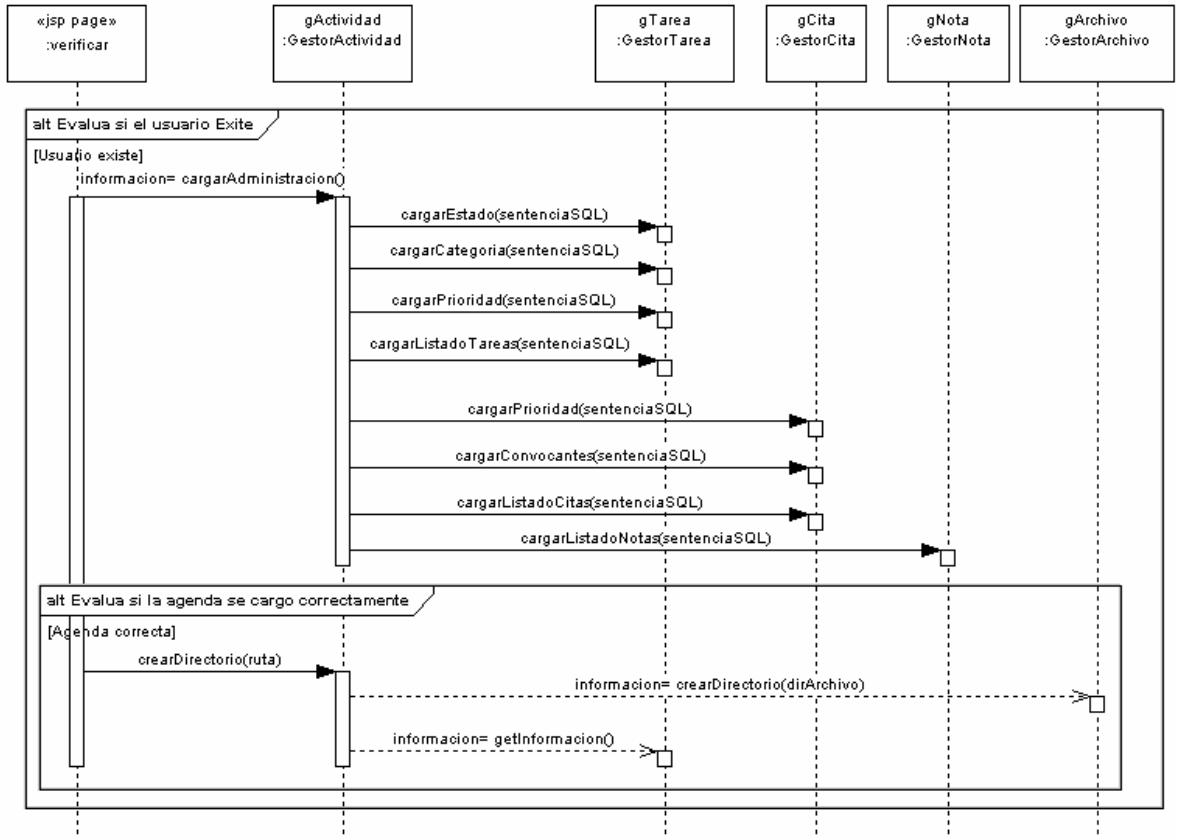
Figura 10. Diagramas de secuencia del diseño

Documento	Título
DS1	Diagrama de secuencia - Módulo Administración de la agenda - Paquete Manejo de Usuarios

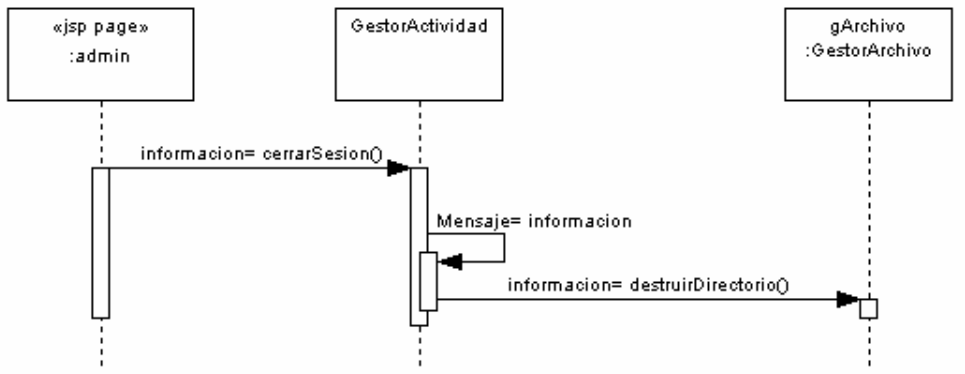
CO1	Mensaje= iniciarSesionAdmin(clave, login)



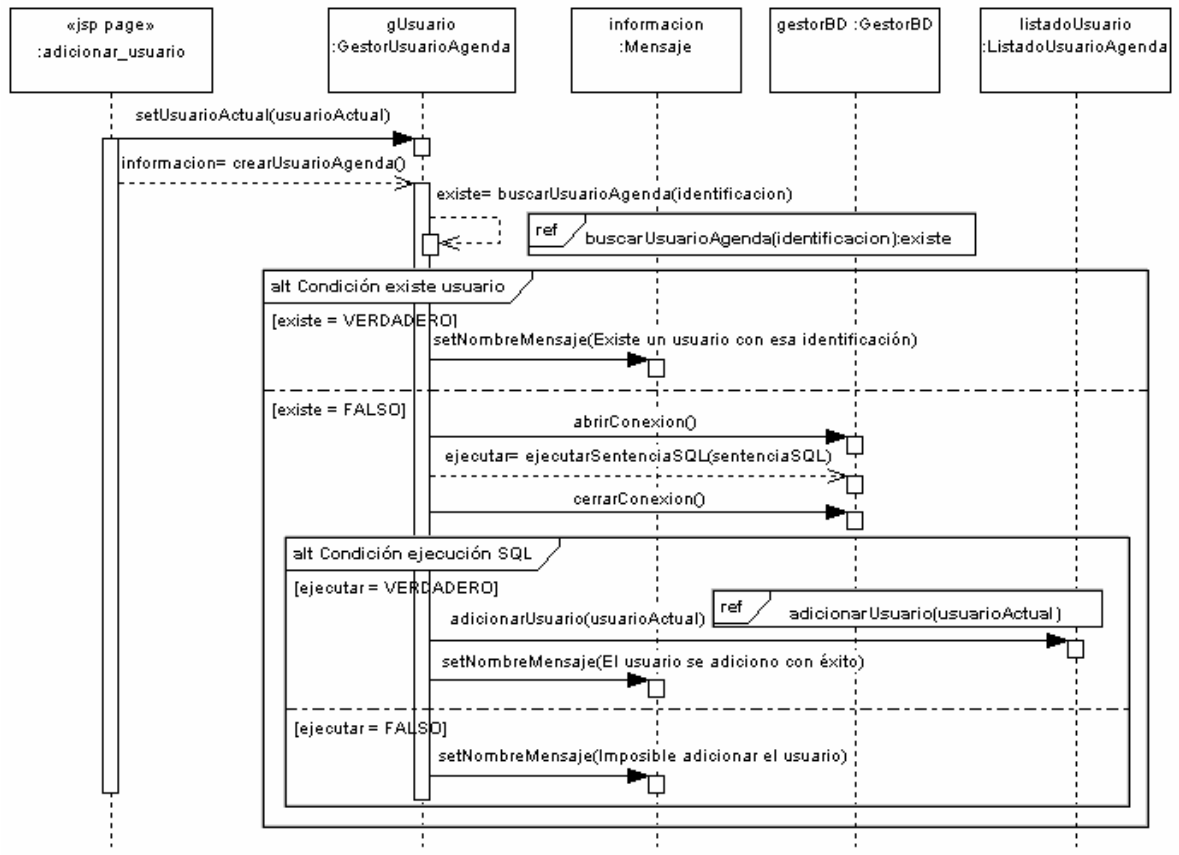
CO2	cargarAdministracion()

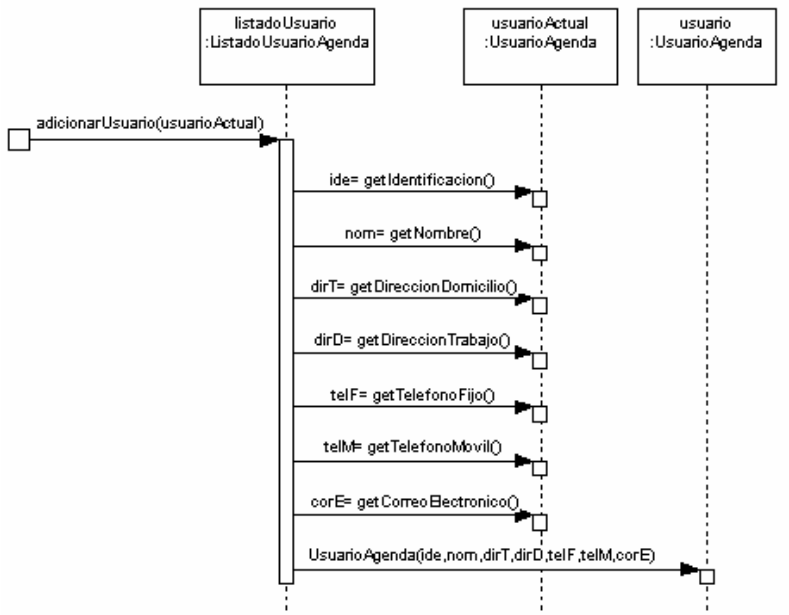


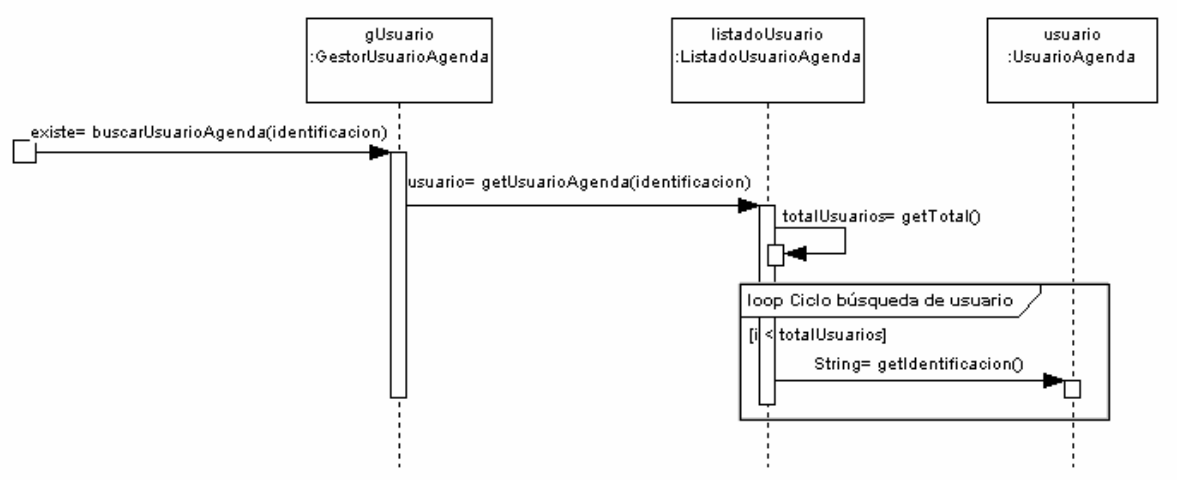
CO3	Mensaje= cerrarSesionAdmin()



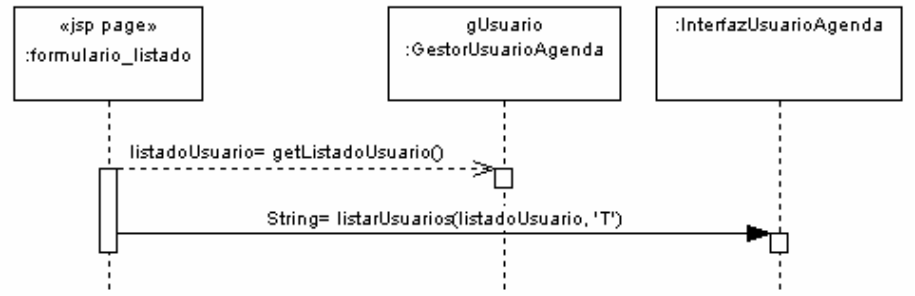
OP4	Mensaje = crearUsuarioAgenda()



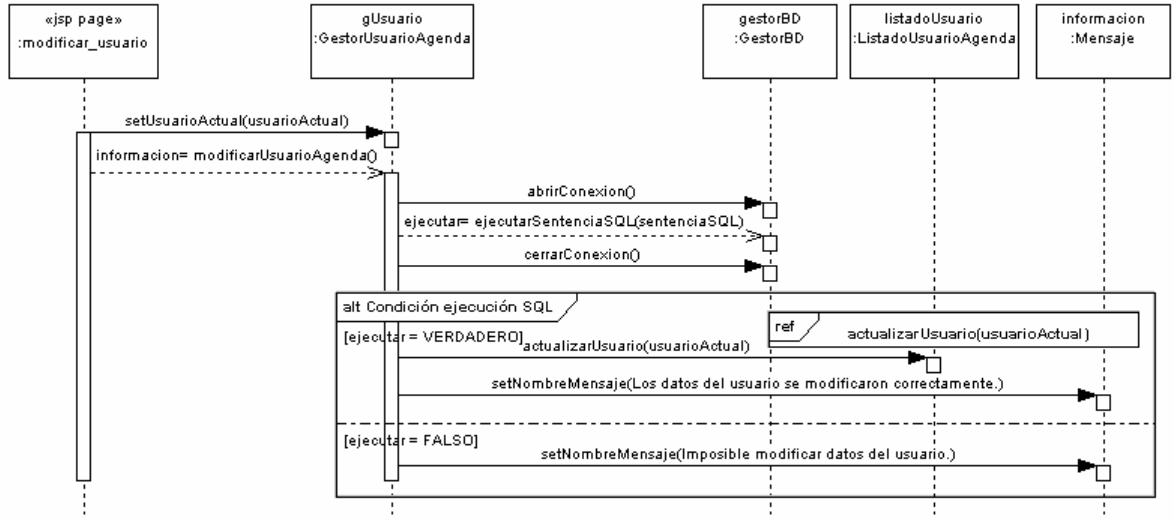


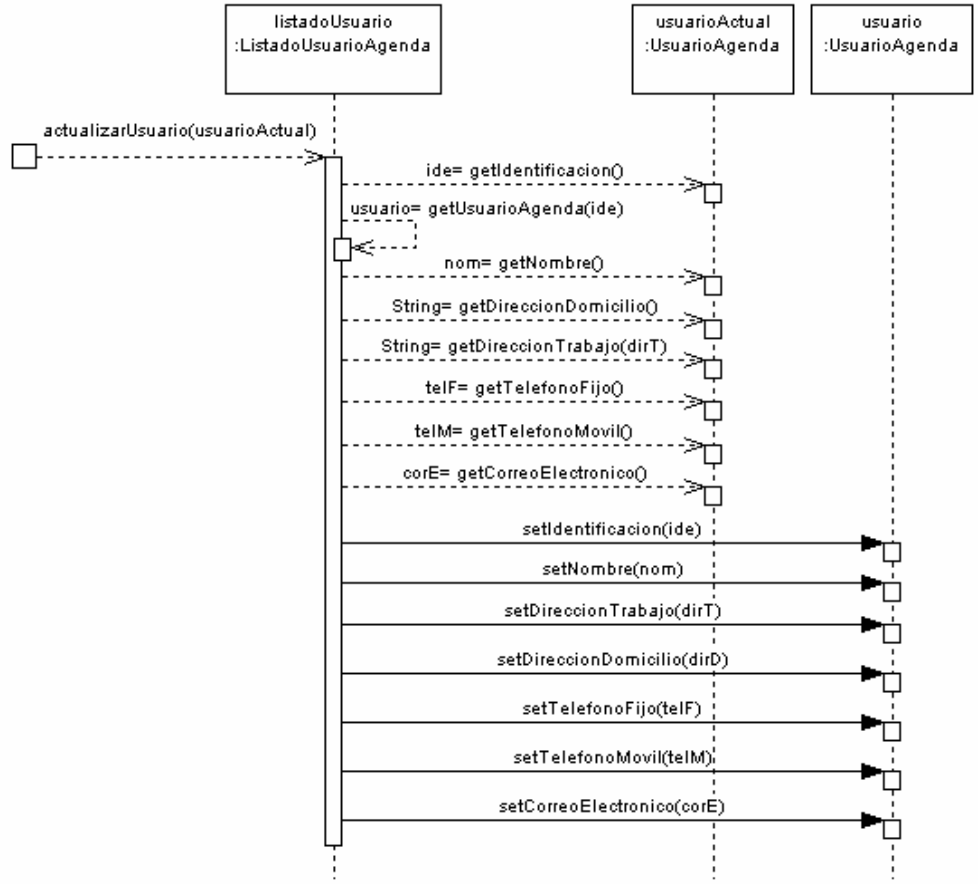


OP5	Vector= obtenerListadoUsuarios()

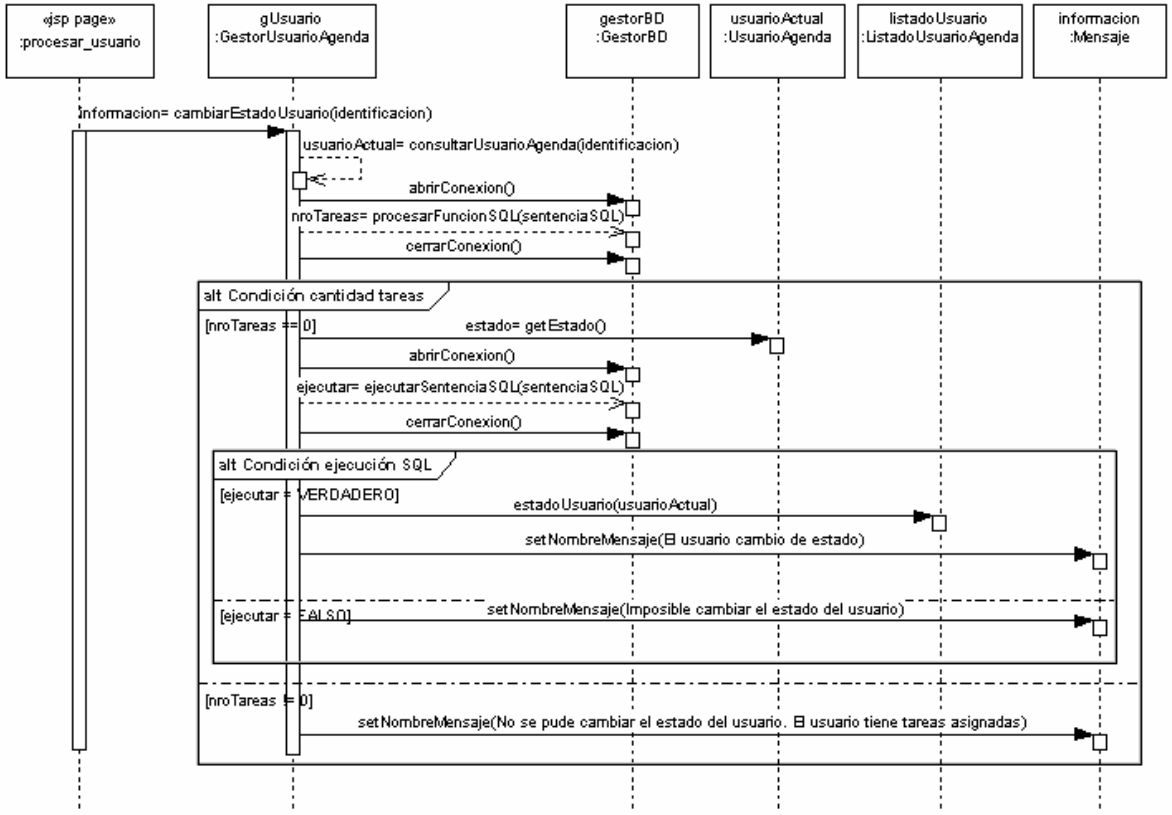


OP6	Mensaje= modificarUsuarioAgenda()

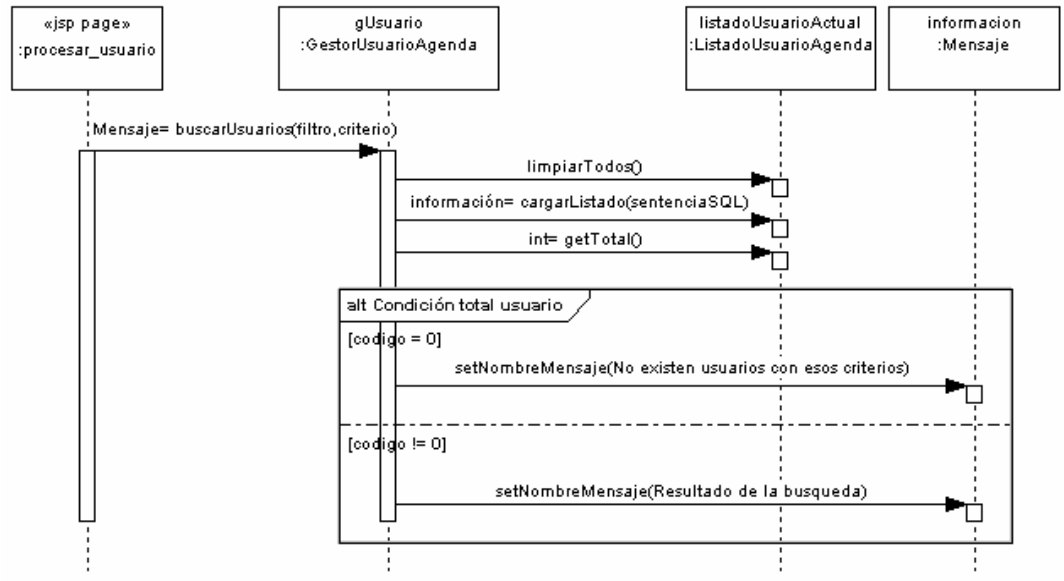




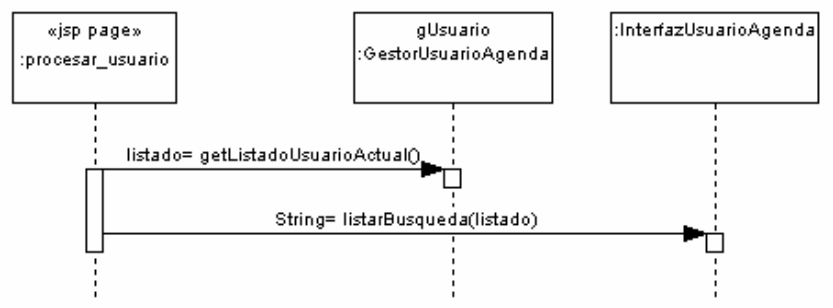
OP7	Mensaje= cambiarEstadoUsuario(identificación)



OP8	Mensaje= busquedaUsuarios(criterio, filtro)

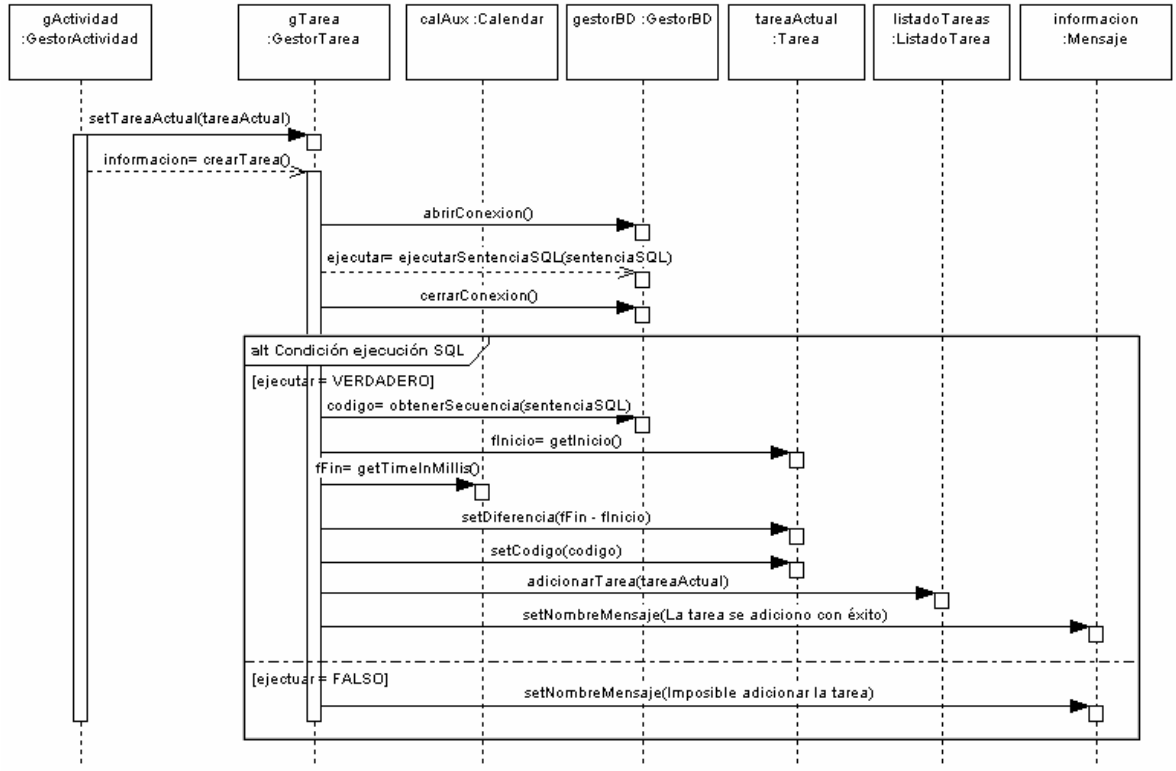


CO9	Vector= obtenerResultadoBusqueda()

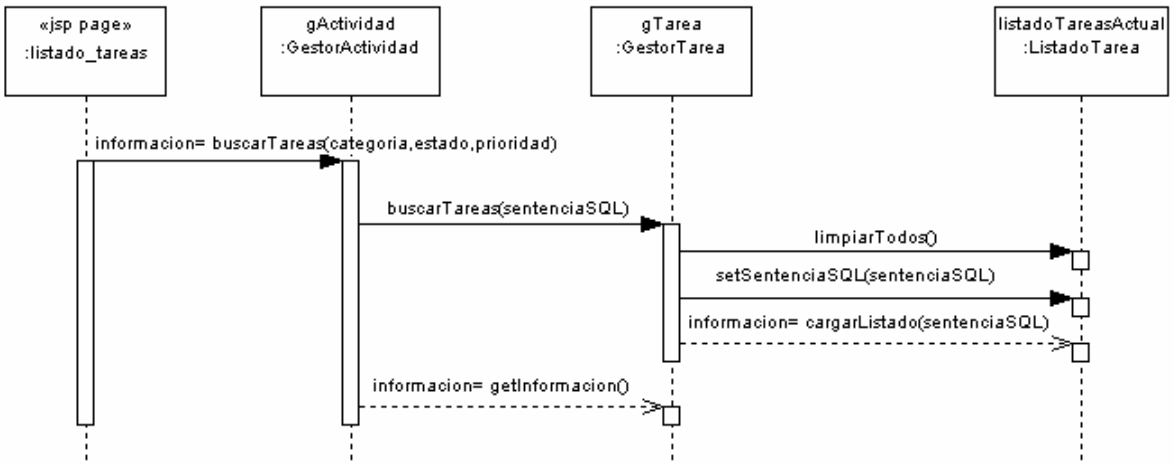


Documento	**Título**
DS2	Diagrama de secuencia - Módulo Administración de la agenda - Paquete Manejo de Actividades

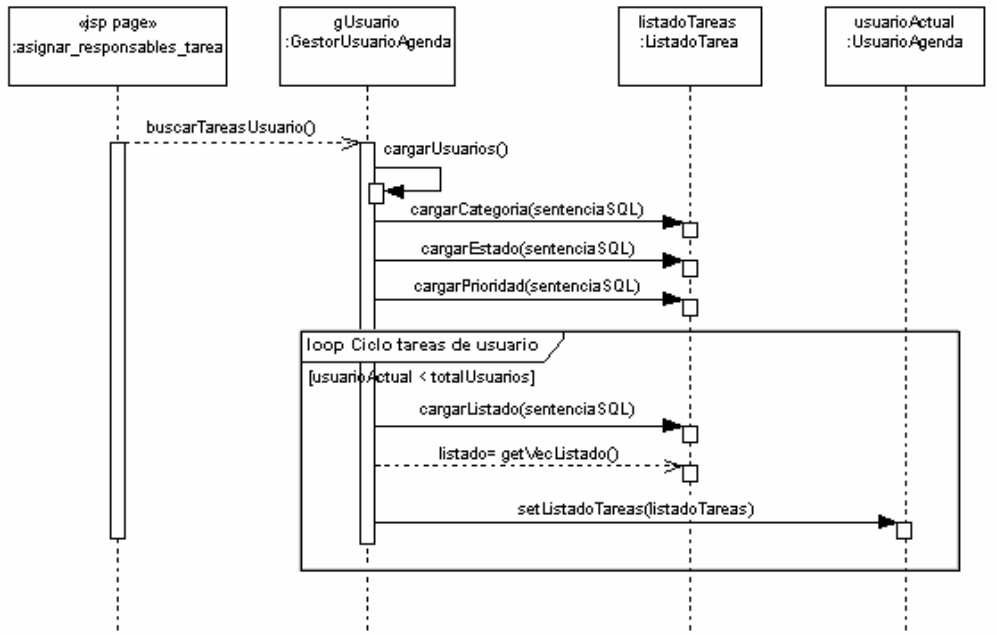
CO10	Mensaje = crearTarea()



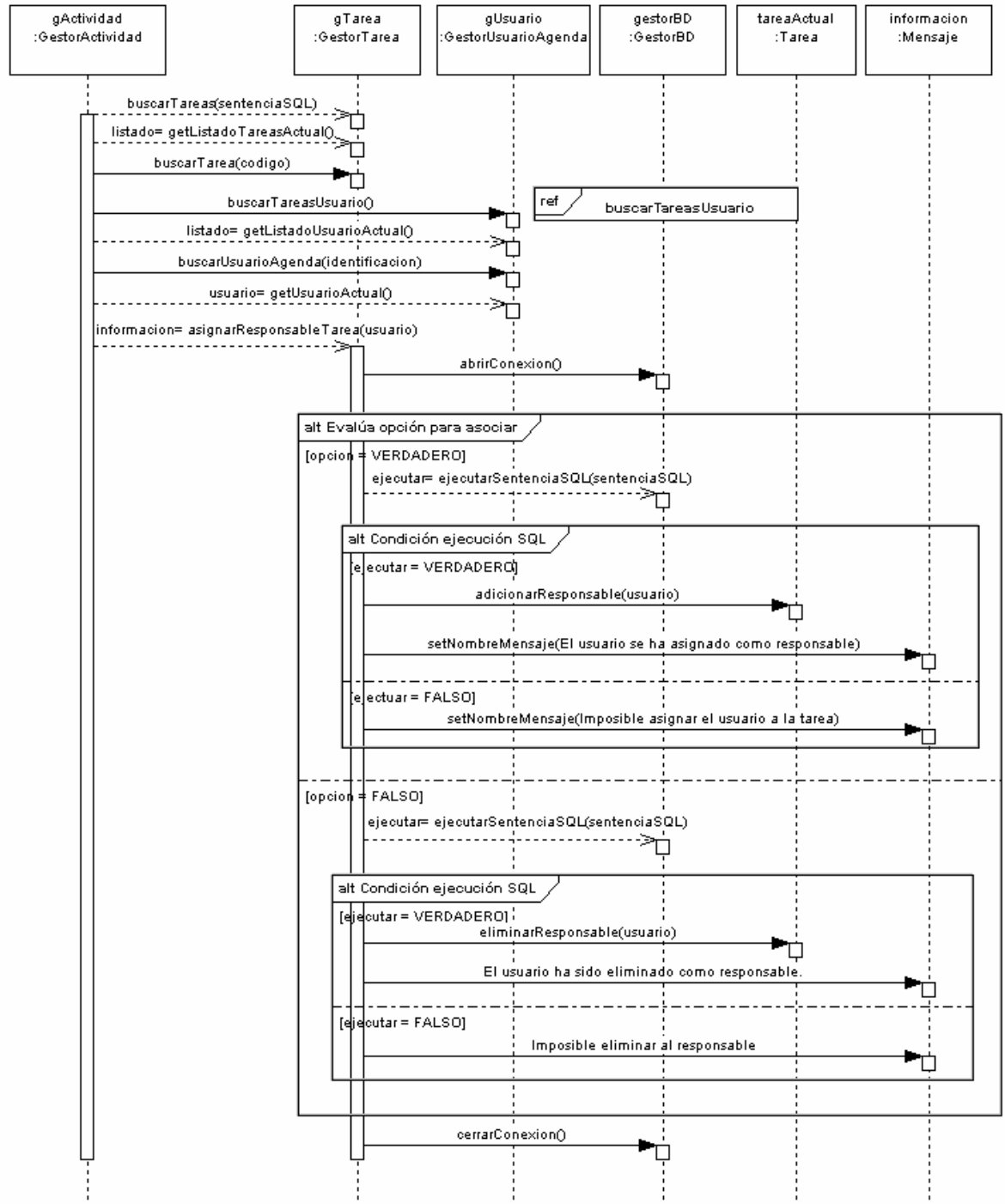
CO11	buscarTareas(categoría, estado, prioridad)



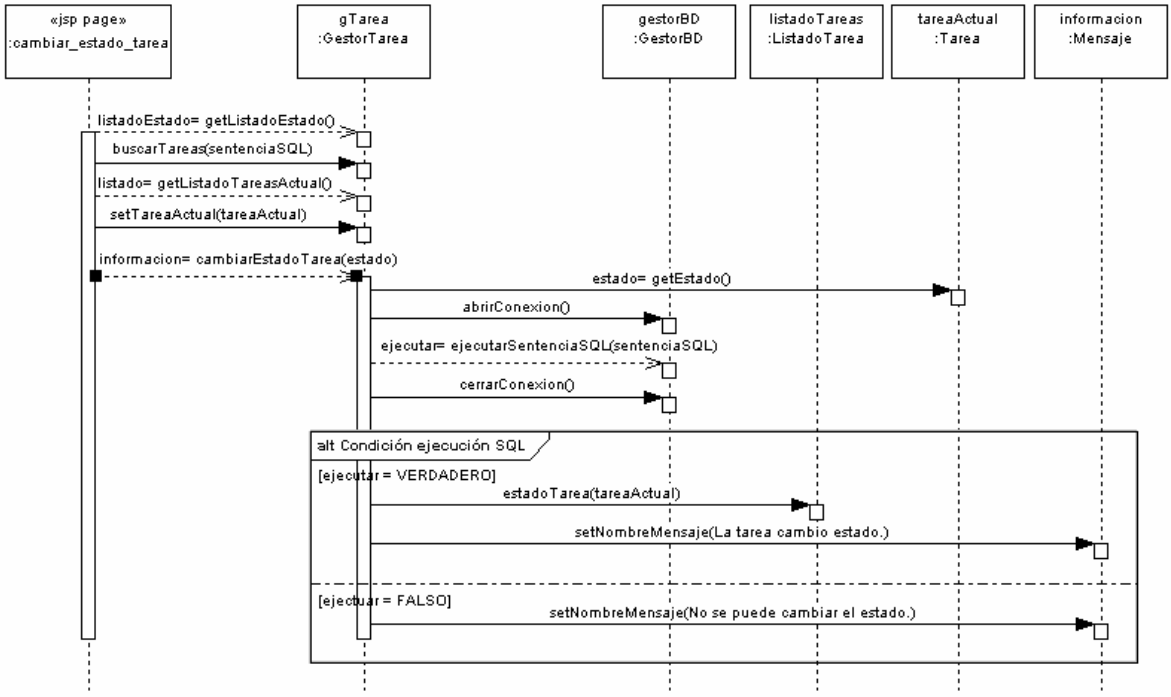
CO12	buscarTareasUsuario()



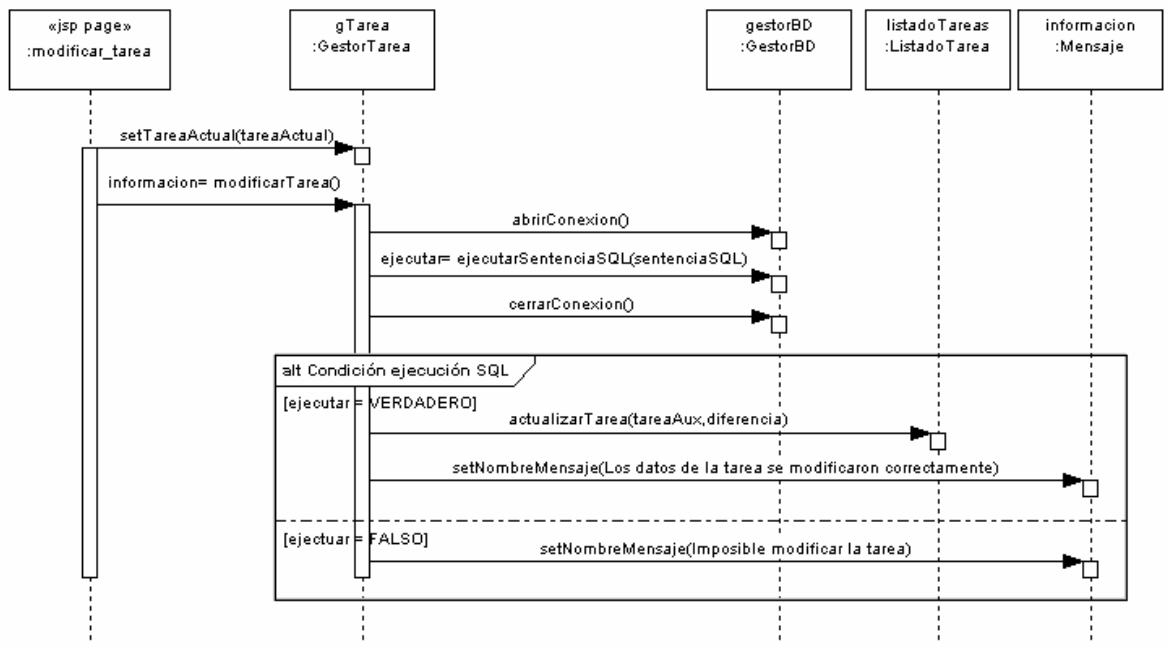
CO13	asignarResponsableTarea(usuario, opcion)



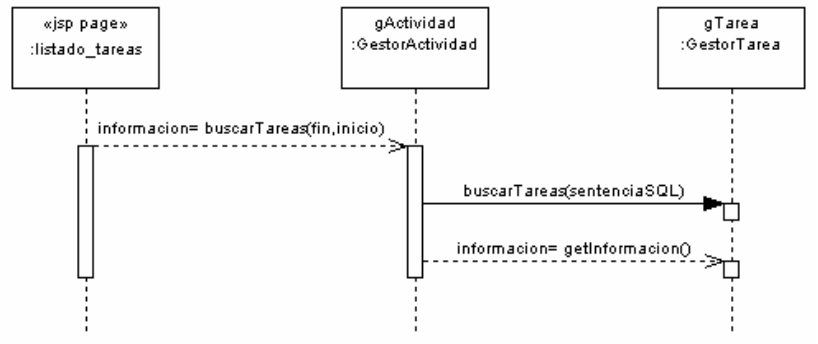
CO14	cambiarEstadoTarea(estado)



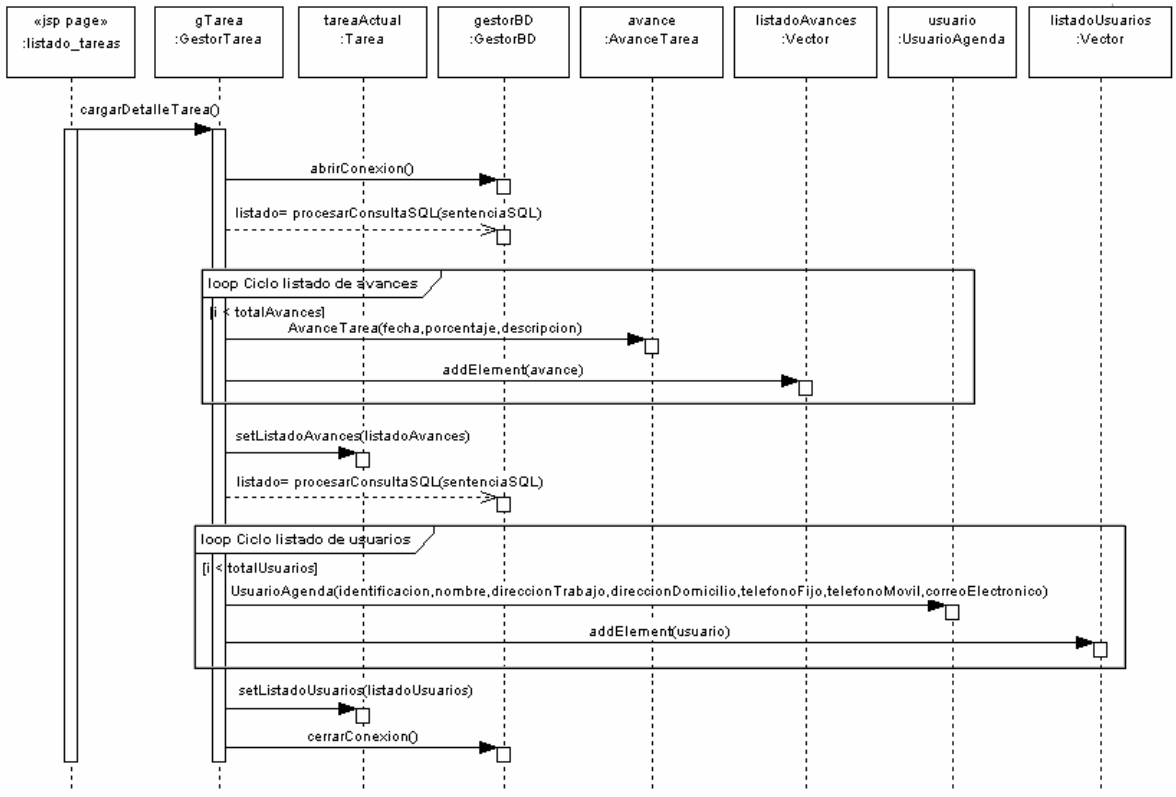
CO15	Mensaje= modificarTarea()



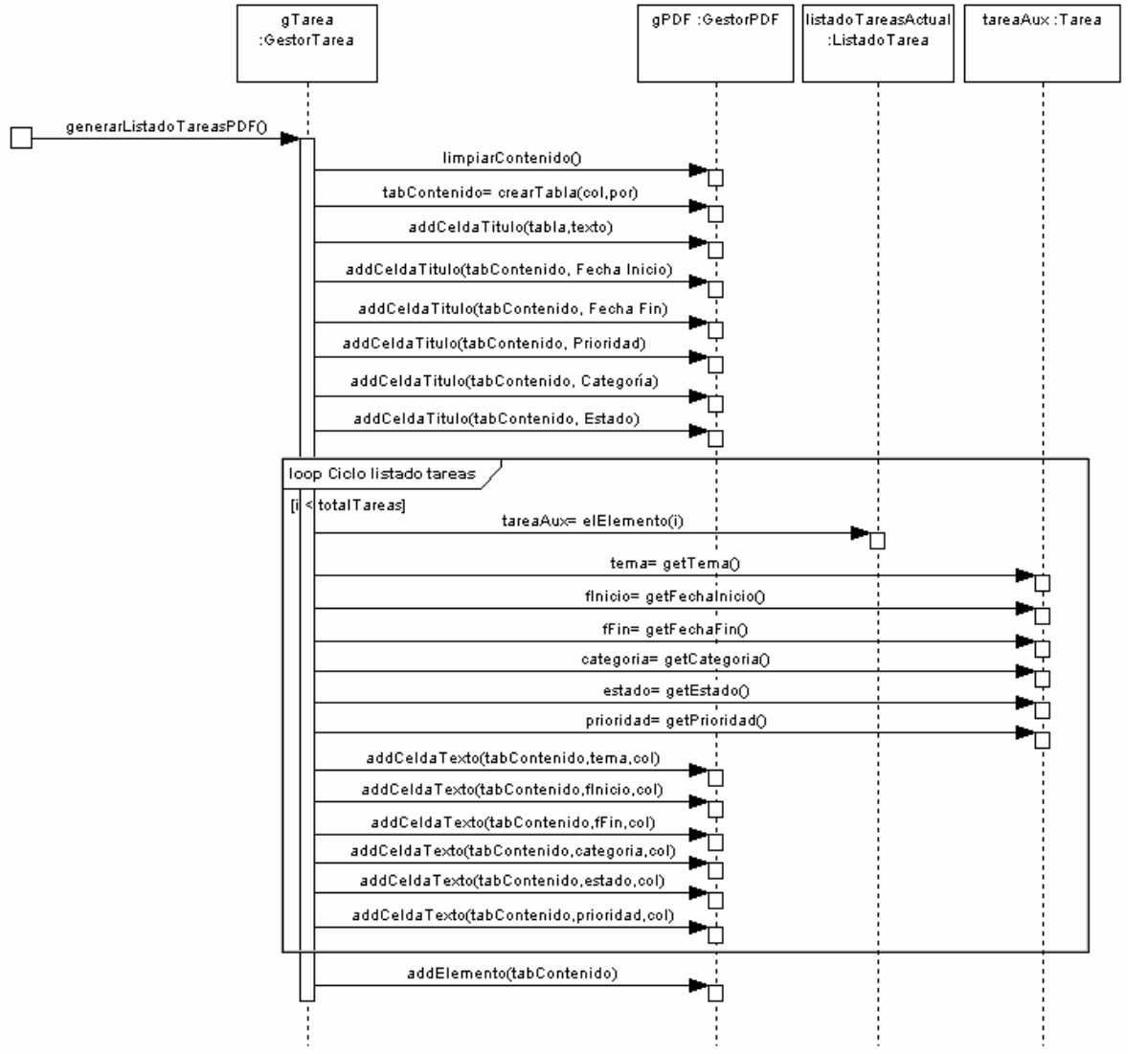
CO16	Mensaje= buscarTareas(fin, inicio)

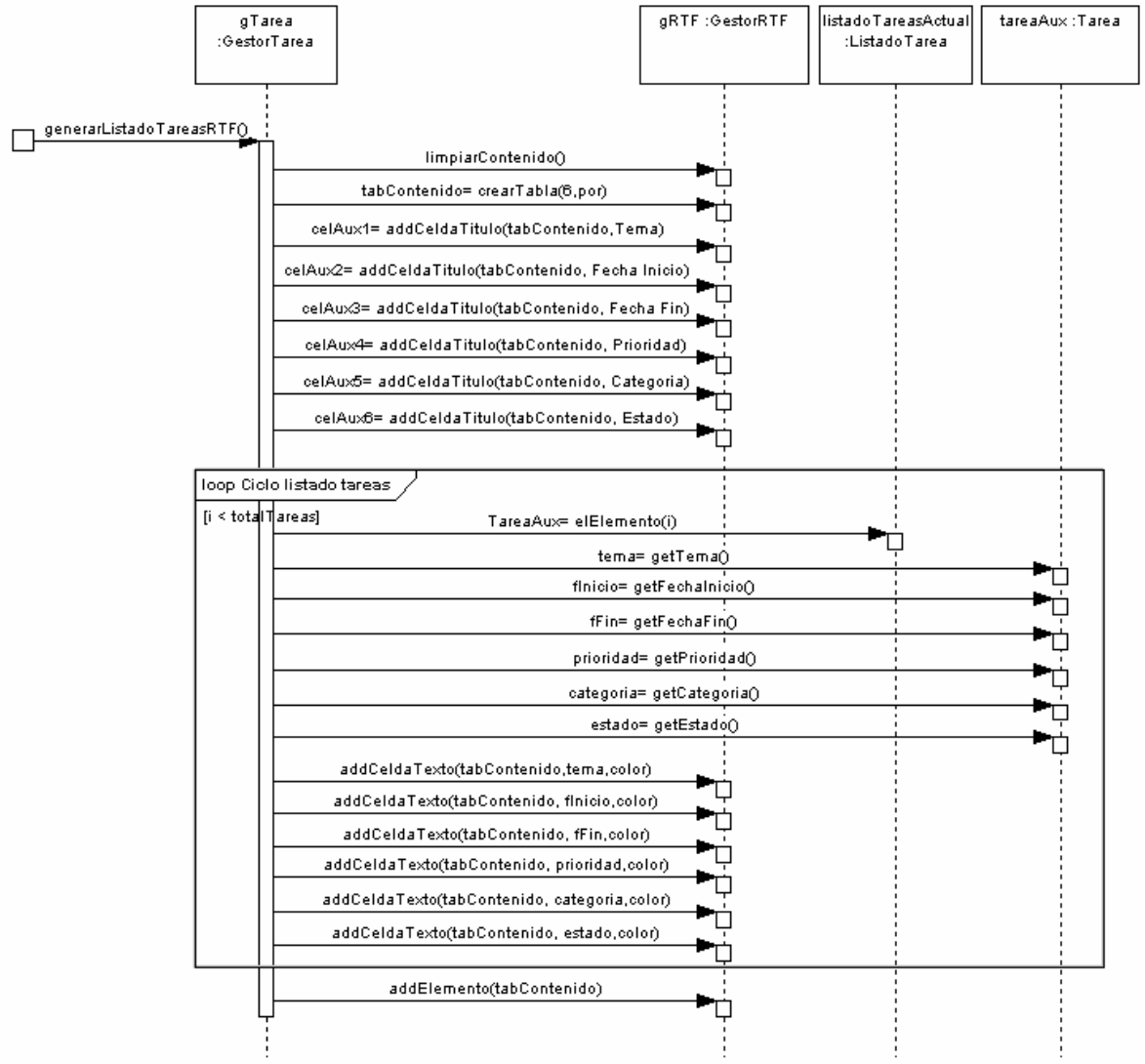


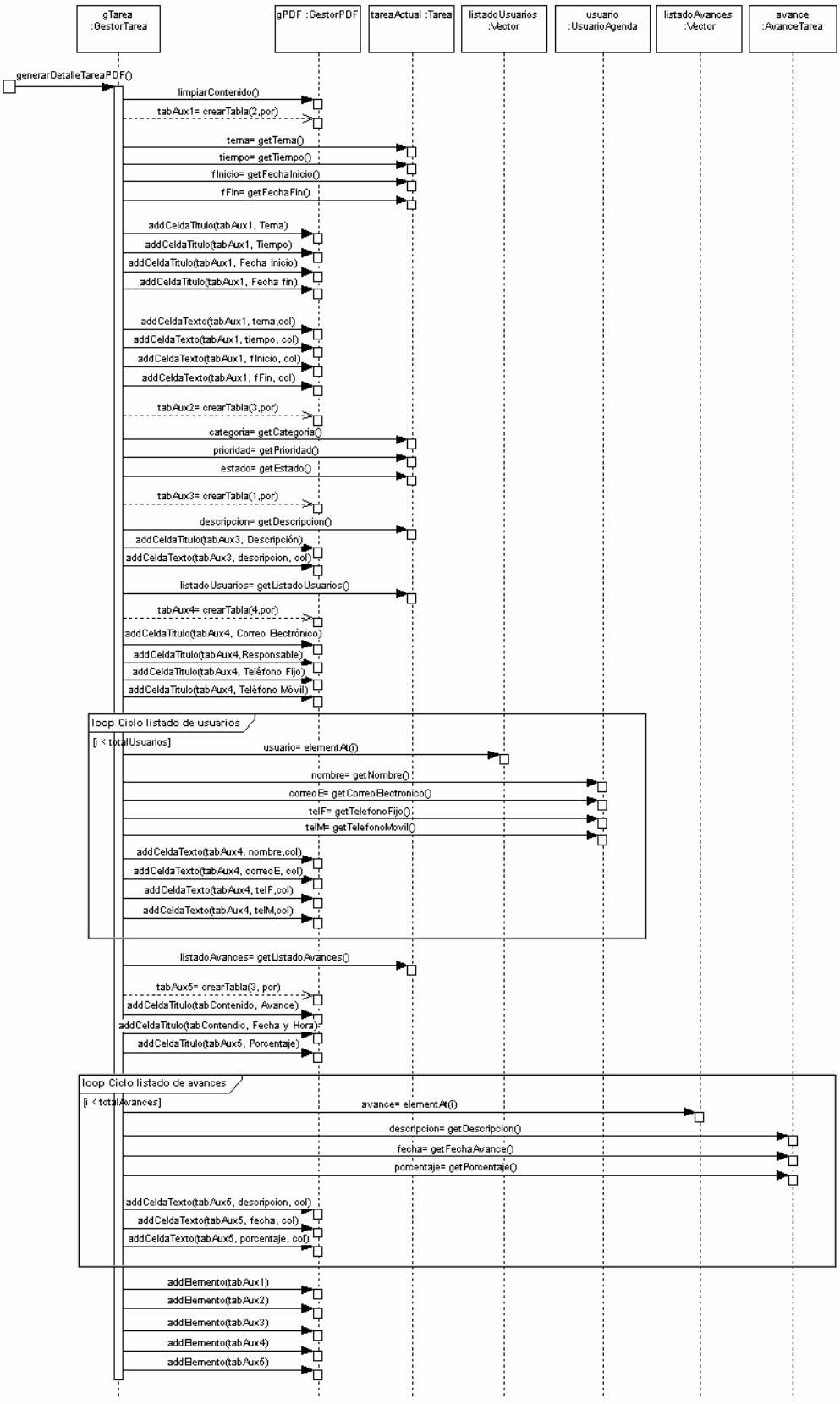
CO17	Mensaje= cargarDetalleTarea()

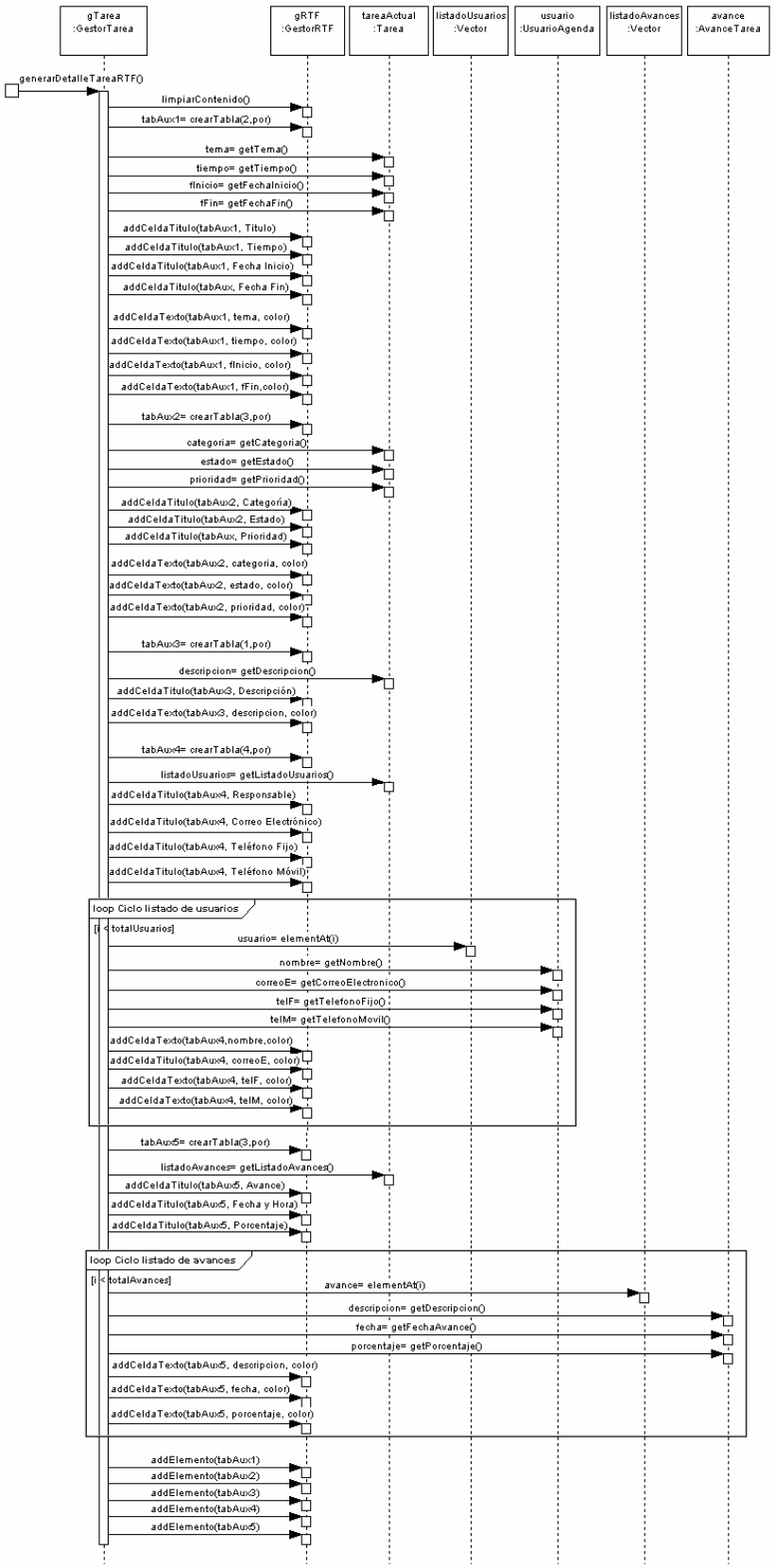


CO18	Mensaje= generarArchivo()

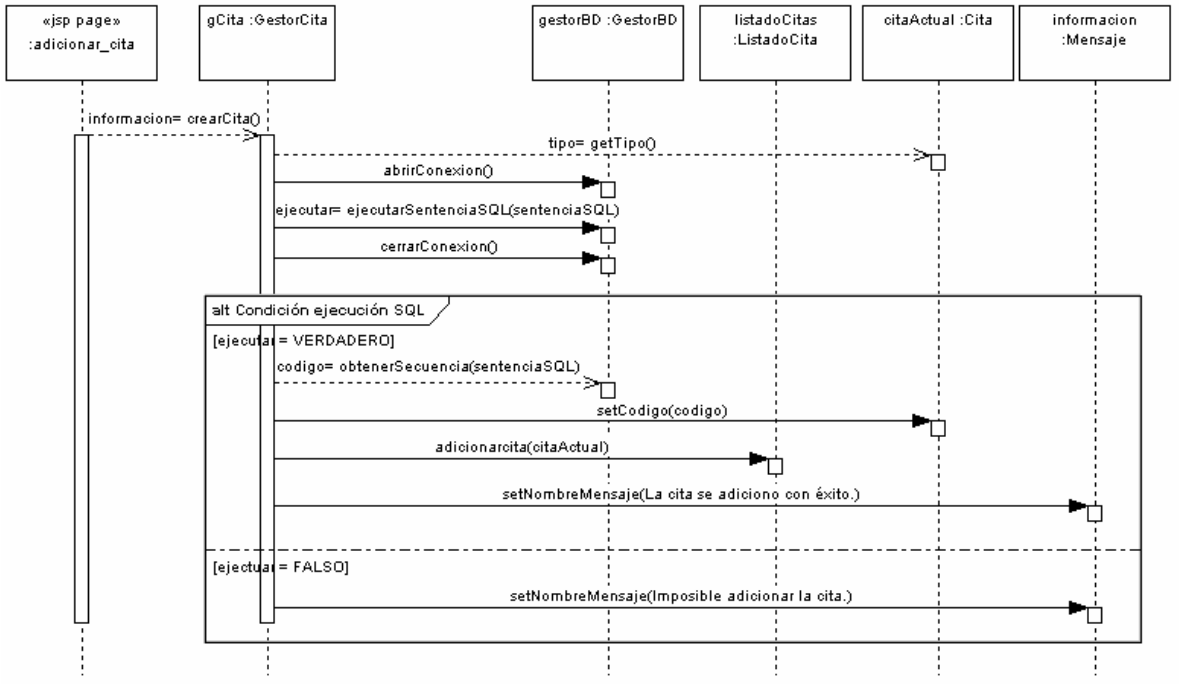



gTarea :GestorTarea
gRTF :GestorRTF
listadoTareasActual :ListadoTarea
tareaAux :Tarea
generarListadoTareasRTF()
limpiarContenido()
tabContenido= crearTabla(6,por)
celAux1= addCeldaTitulo(tabContenido,Tema)
celAux2= addCeldaTitulo(tabContenido, Fecha Inicio)
celAux3= addCeldaTitulo(tabContenido, Fecha Fin)
celAux4= addCeldaTitulo(tabContenido, Prioridad)
celAux5= addCeldaTitulo(tabContenido, Categoria)
celAux6= addCeldaTitulo(tabContenido, Estado)
loop Ciclo listado tareas
[i < totalTareas]
TareaAux= elElemento(i)
tema= getTema()
fInicio= getFechaInicio()
fFin= getFechaFin()
prioridad= getPrioridad()
categoria= getCategoria()
estado= getEstado()
addCeldaTexto(tabContenido,tema,color)
addCeldaTexto(tabContenido, fInicio,color)
addCeldaTexto(tabContenido, fFin,color)
addCeldaTexto(tabContenido, prioridad,color)
addCeldaTexto(tabContenido, categoria,color)
addCeldaTexto(tabContenido, estado,color)
addElemento(tabContenido)

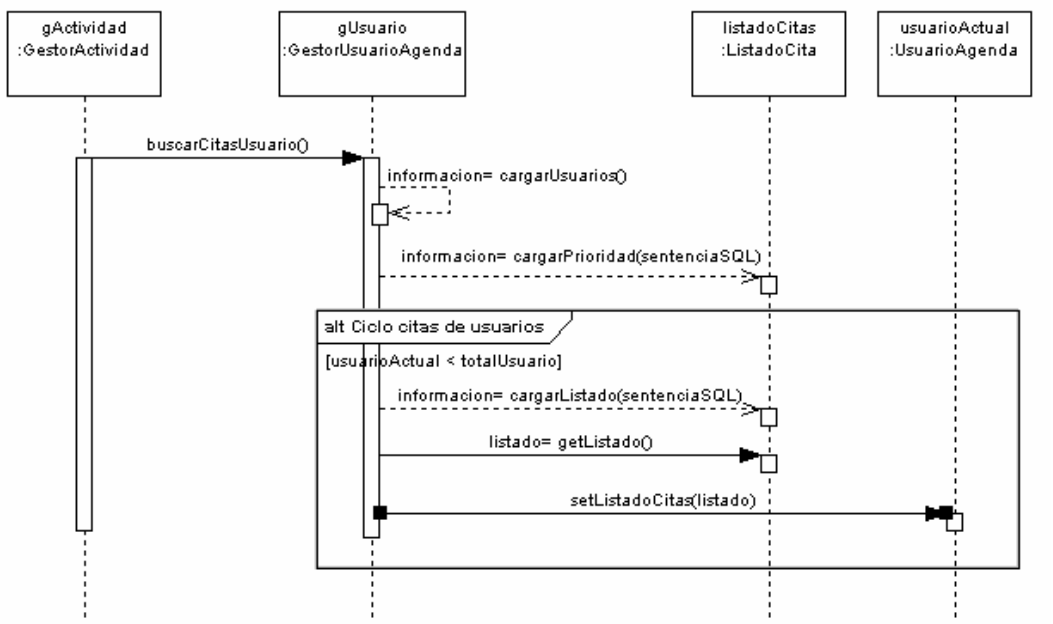

gTarea :GestorTarea
gPDF :GestorPDF
tareaActual :Tarea
listadoUsuarios :Vector
usuario :UsuarioAgenda
listadoAvances :Vector
avance :AvanceTarea
generarDetalleTareaPDF()
limpiarContenido()
tabAux1= crearTabla(2,por)
tema= getTema()
tiempo= getTiempo()
fInicio= getFechaInicio()
fFin= getFechaFin()
addCeldaTitulo(tabAux1, Tema)
addCeldaTitulo(tabAux1, Tiempo)
addCeldaTitulo(tabAux1, Fecha Inicio)
addCeldaTitulo(tabAux1, Fecha fin)
addCeldaTexto(tabAux1, tema,col)
addCeldaTexto(tabAux1, tiempo, col)
addCeldaTexto(tabAux1, fInicio, col)
addCeldaTexto(tabAux1, fFin, col)
tabAux2= crearTabla(3,por)
categoria= getCategoria()
prioridad= getPrioridad()
estado= getEstado()
tabAux3= crearTabla(1,por)
descripcion= getDescripcion()
addCeldaTitulo(tabAux3, Descripción)
addCeldaTexto(tabAux3, descripcion, col)
listadoUsuarios= getListadoUsuarios()
tabAux4= crearTabla(4,por)
addCeldaTitulo(tabAux4, Correo Electrónico)
addCeldaTitulo(tabAux4,Responsable)
addCeldaTitulo(tabAux4, Teléfono Fijo)
addCeldaTitulo(tabAux4, Teléfono Móvil)
loop Ciclo listado de usuarios
[i < totalUsuarios]
usuario= elementAt(i)
nombre= getNombre()
correoE= getCorreoElectronico()
telF= getTelefonoFijo()
telM= getTelefonoMovil()
addCeldaTexto(tabAux4, nombre,col)
addCeldaTexto(tabAux4, correoE, col)
addCeldaTexto(tabAux4, telF,col)
addCeldaTexto(tabAux4, telM,col)
listadoAvances= getListadoAvances()
tabAux5= crearTabla(3, por)
addCeldaTitulo(tabContenido, Avance)
addCeldaTitulo(tabContendio, Fecha y Hora)
addCeldaTitulo(tabAux5, Porcentaje)
loop Ciclo listado de avances
[i < totalAvances]
avance= elementAt(i)
descripcion= getDescripcion()
fecha= getFechaAvance()
porcentaje= getPorcentaje()
addCeldaTexto(tabAux5, descripcion, col)
addCeldaTexto(tabAux5, fecha, col)
addCeldaTexto(tabAux5, porcentaje, col)
addElemento(tabAux1)
addElemento(tabAux2)
addElemento(tabAux3)
addElemento(tabAux4)
addElemento(tabAux5)


gTarea :GestorTarea
gRTF :GestorRTF
tareaActual :Tarea
listadoUsuarios :Vector
usuario :UsuarioAgenda
listadoAvances :Vector
avance :AvanceTarea
generarDetalleTareaRTF()
limpiarContenido()
tabAux1= crearTabla(2,por)
tema= getTema()
tiempo= getTiempo()
fInicio= getFechaInicio()
fFin= getFechaFin()
addCeldaTitulo(tabAux1, Titulo)
addCeldaTitulo(tabAux1, Tiempo)
addCeldaTitulo(tabAux1, Fecha Inicio)
addCeldaTitulo(tabAux, Fecha Fin)
addCeldaTexto(tabAux1, tema, color)
addCeldaTexto(tabAux1, tiempo, color)
addCeldaTexto(tabAux1, fInicio, color)
addCeldaTexto(tabAux1, fFin,color)
tabAux2= crearTabla(3,por)
categoria= getCategoria()
estado= getEstado()
prioridad= getPrioridad()
addCeldaTitulo(tabAux2, Categoría)
addCeldaTitulo(tabAux2, Estado)
addCeldaTitulo(tabAux, Prioridad)
addCeldaTexto(tabAux2, categoria, color)
addCeldaTexto(tabAux2, estado, color)
addCeldaTexto(tabAux2, prioridad, color)
tabAux3= crearTabla(1,por)
descripcion= getDescripcion()
addCeldaTitulo(tabAux3, Descripción)
addCeldaTexto(tabAux3, descripcion, color)
tabAux4= crearTabla(4,por)
listadoUsuarios= getListadoUsuarios()
addCeldaTitulo(tabAux4, Responsable)
addCeldaTitulo(tabAux4, Correo Electrónico)
addCeldaTitulo(tabAux4, Teléfono Fijo)
addCeldaTitulo(tabAux4, Teléfono Móvil)
loop Ciclo listado de usuarios
[i < totalUsuarios]
usuario= elementAt(i)
nombre= getNombre()
correoE= getCorreoElectronico()
telF= getTelefonoFijo()
telM= getTelefonoMovil()
addCeldaTexto(tabAux4,nombre,color)
addCeldaTitulo(tabAux4, correoE, color)
addCeldaTexto(tabAux4, telF, color)
addCeldaTexto(tabAux4, telM, color)
tabAux5= crearTabla(3,por)
listadoAvances= getListadoAvances()
addCeldaTitulo(tabAux5, Avance)
addCeldaTitulo(tabAux5, Fecha y Hora)
addCeldaTitulo(tabAux5, Porcentaje)
loop Ciclo listado de avances
[i < totalAvances]
avance= elementAt(i)
descripcion= getDescripcion()
fecha= getFechaAvance()
porcentaje= getPorcentaje()
addCeldaTexto(tabAux5, descripcion, color)
addCeldaTexto(tabAux5, fecha, color)
addCeldaTexto(tabAux5, porcentaje, color)
addElemento(tabAux1)
addElemento(tabAux2)
addElemento(tabAux3)
addElemento(tabAux4)
addElemento(tabAux5)

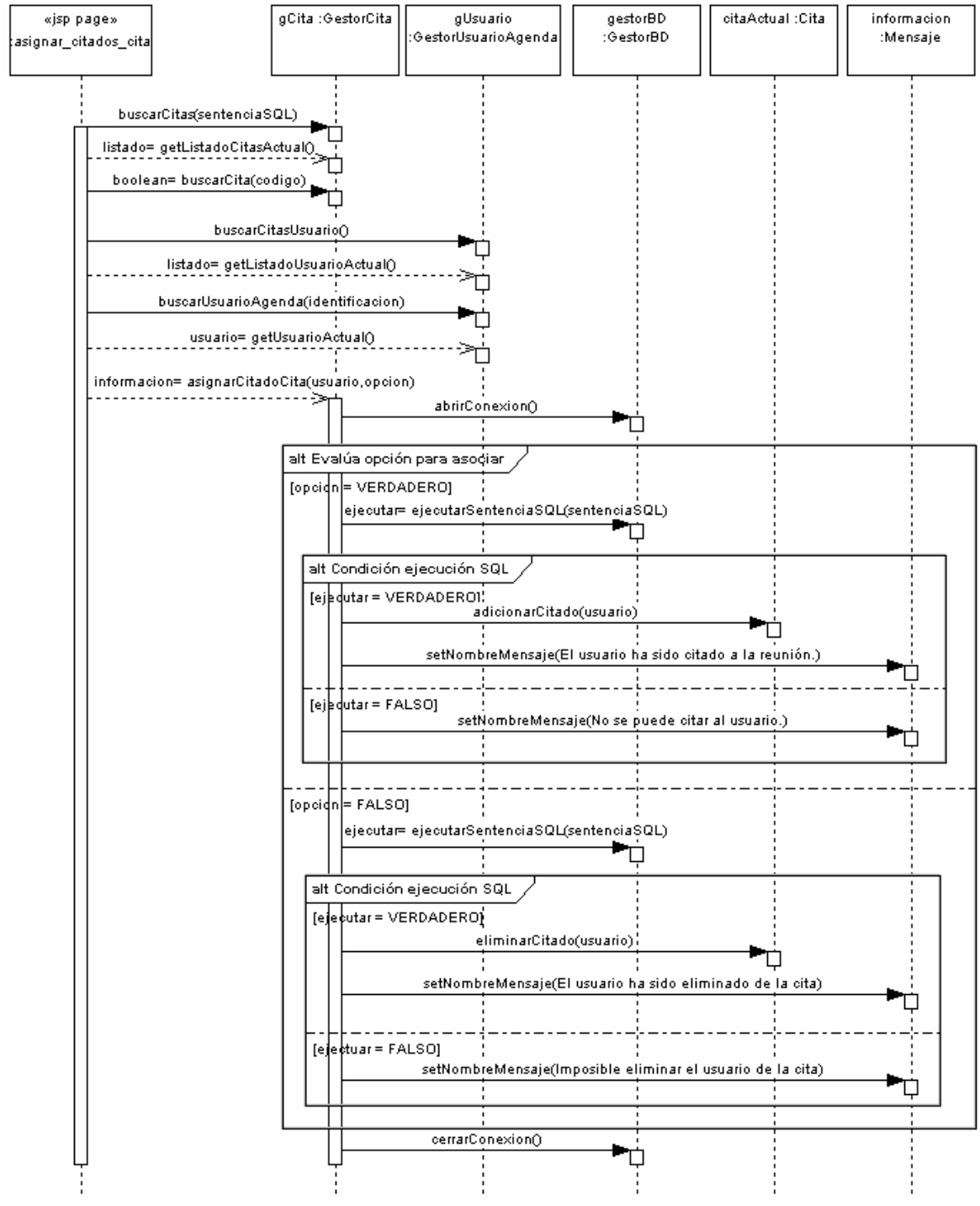
CO19	Mensaje= crearCita()



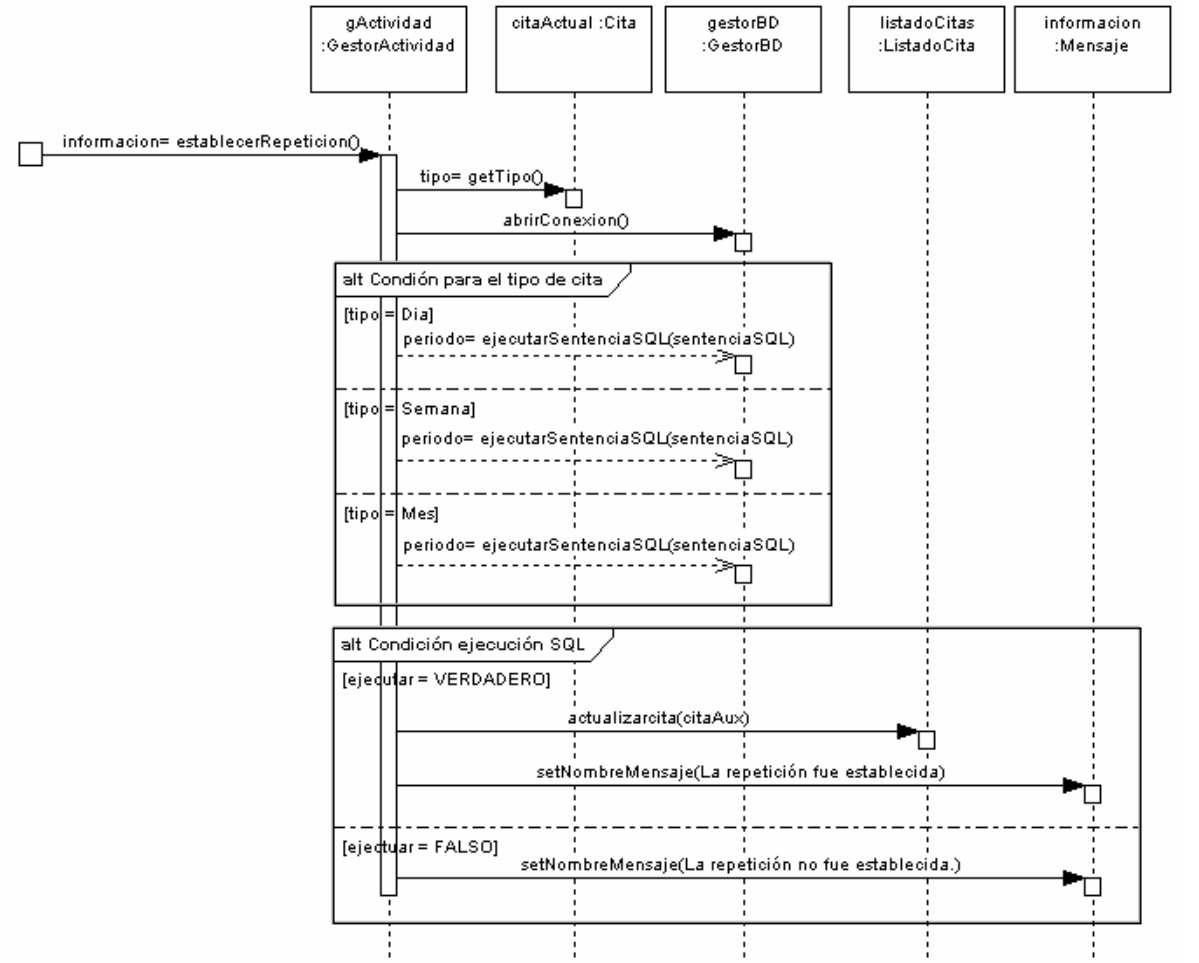
CO20	buscarCitasUsuario()



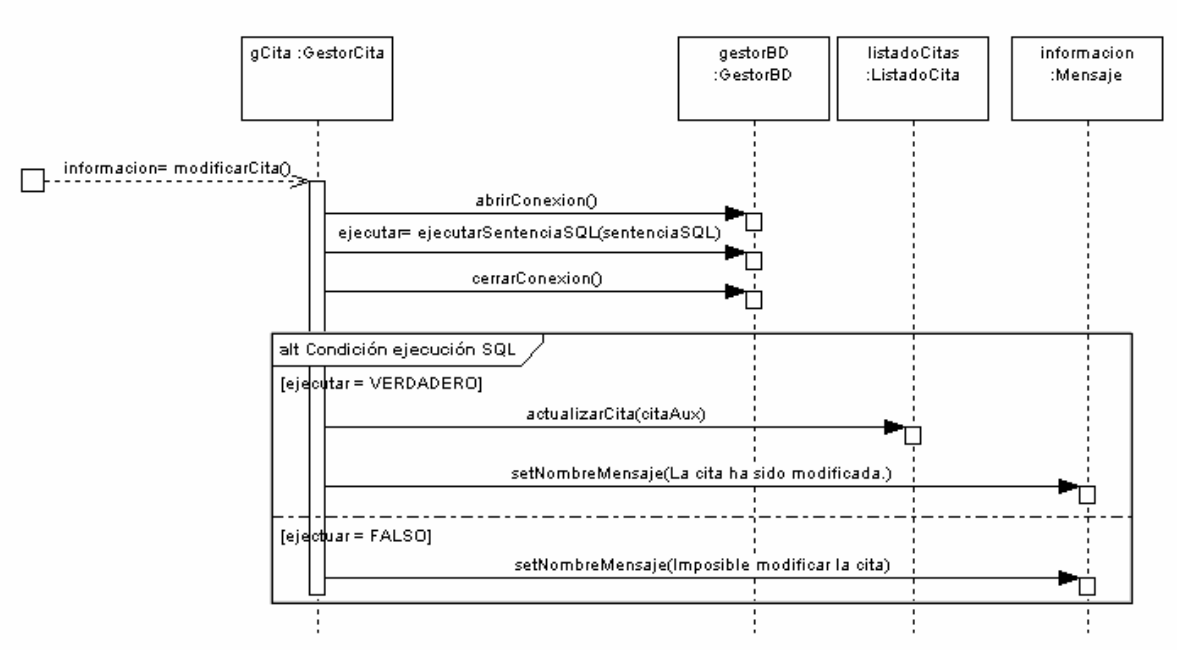
CO21	Mensaje= asignarUsuarioCita(usuario, opcion)



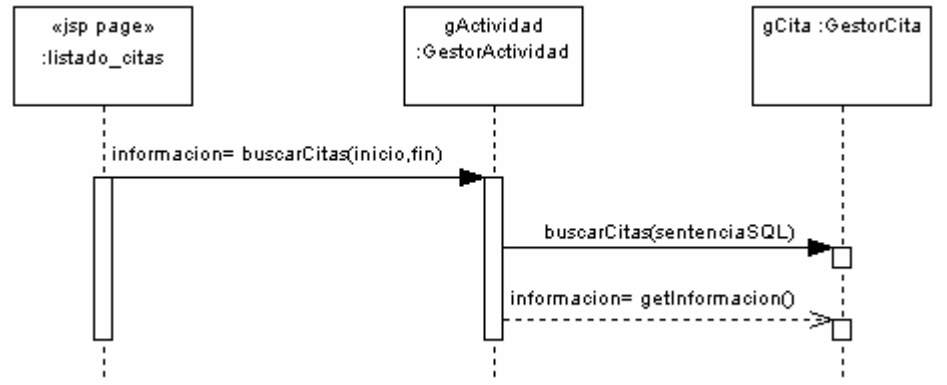
CO22	Mensaje = establecerRepeticion()



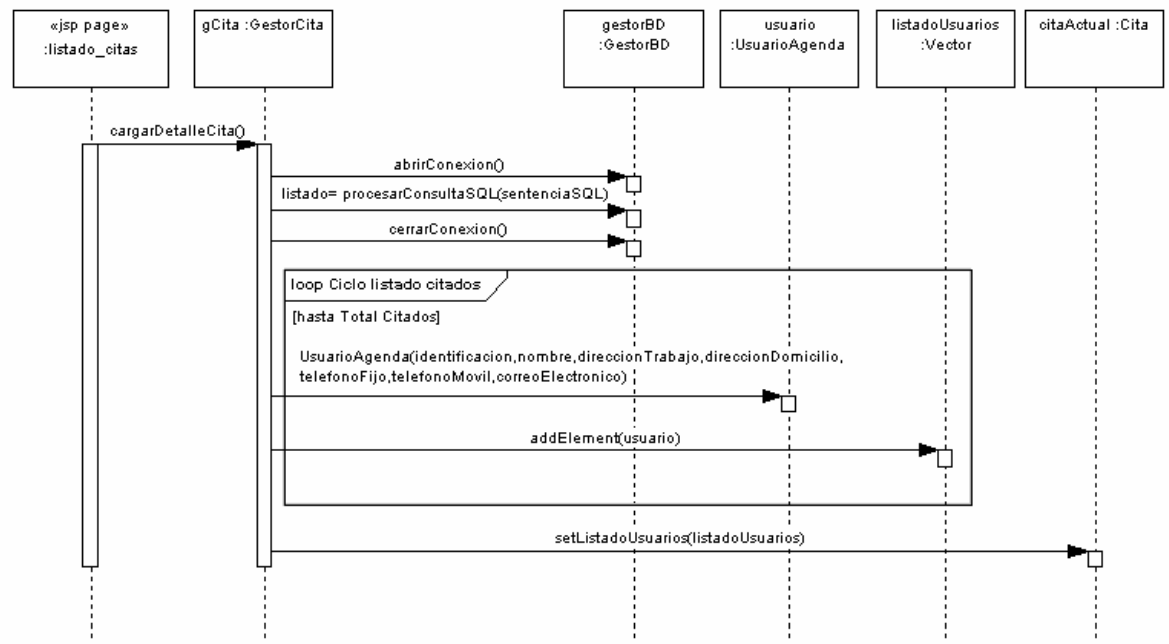
CO23	Mensaje= modificarCita()



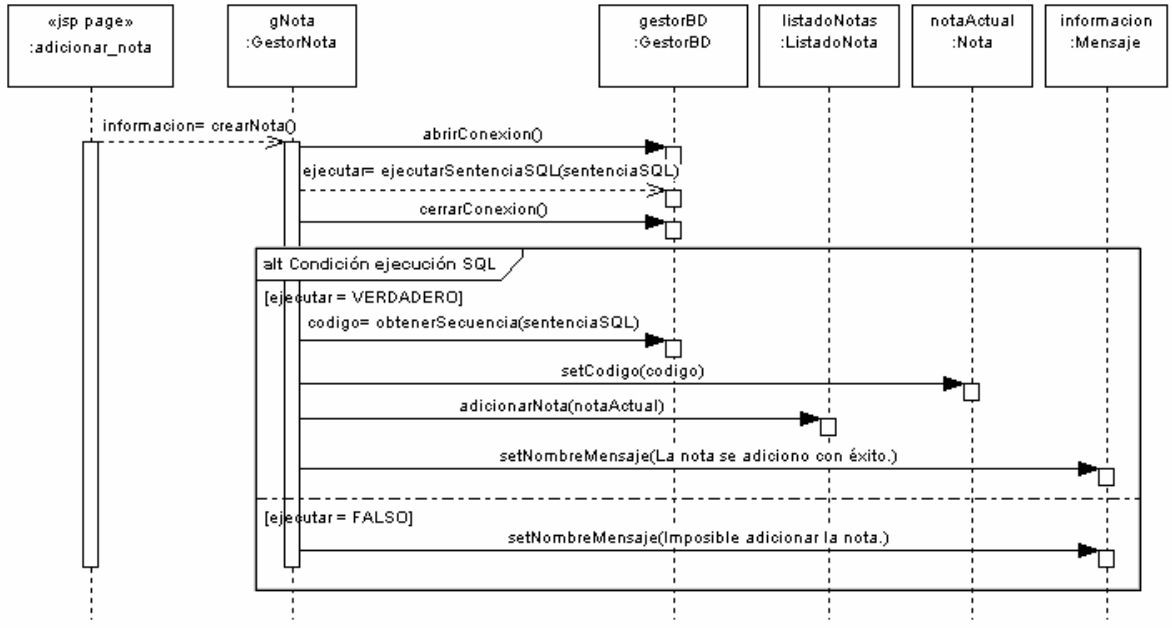
CO24	Mensaje= buscarCitas(fin, inicio)



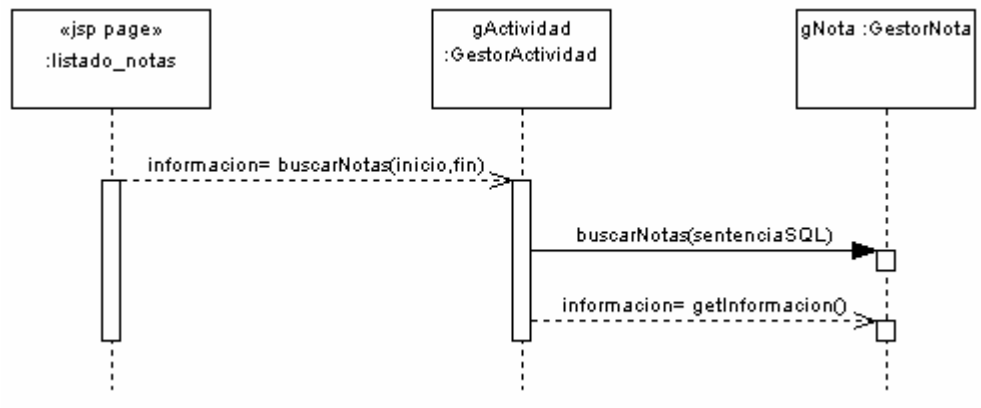
CO25	Mensaje= cargarDetalleCita()



CO26	Mensaje= crearNota()

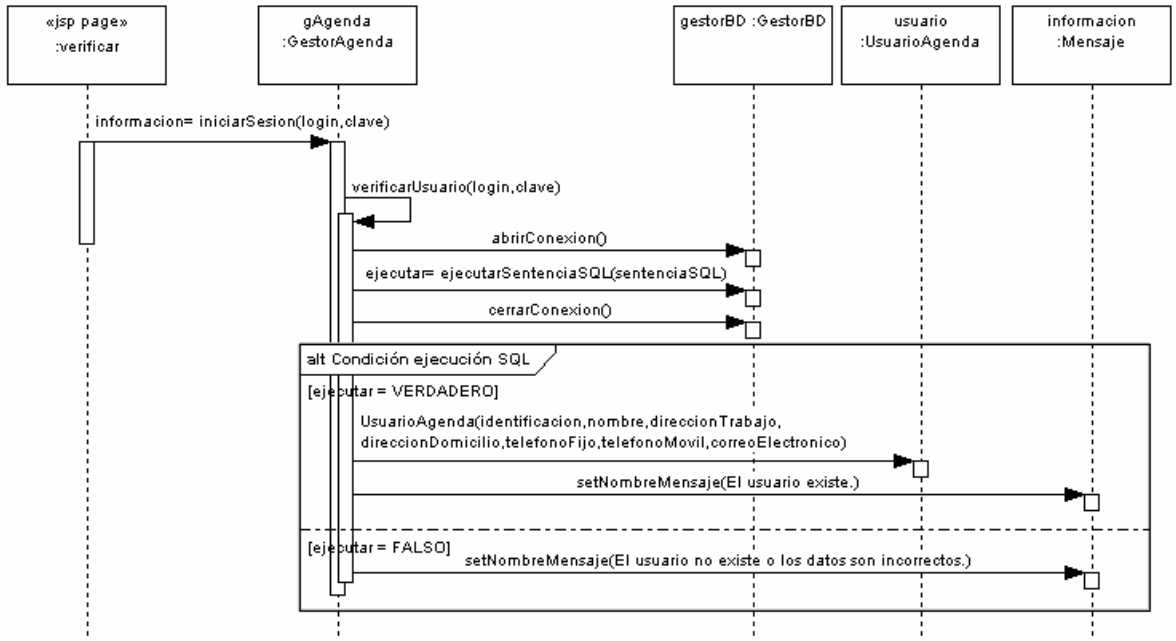


CO27	buscarNotas(fin, inicio)

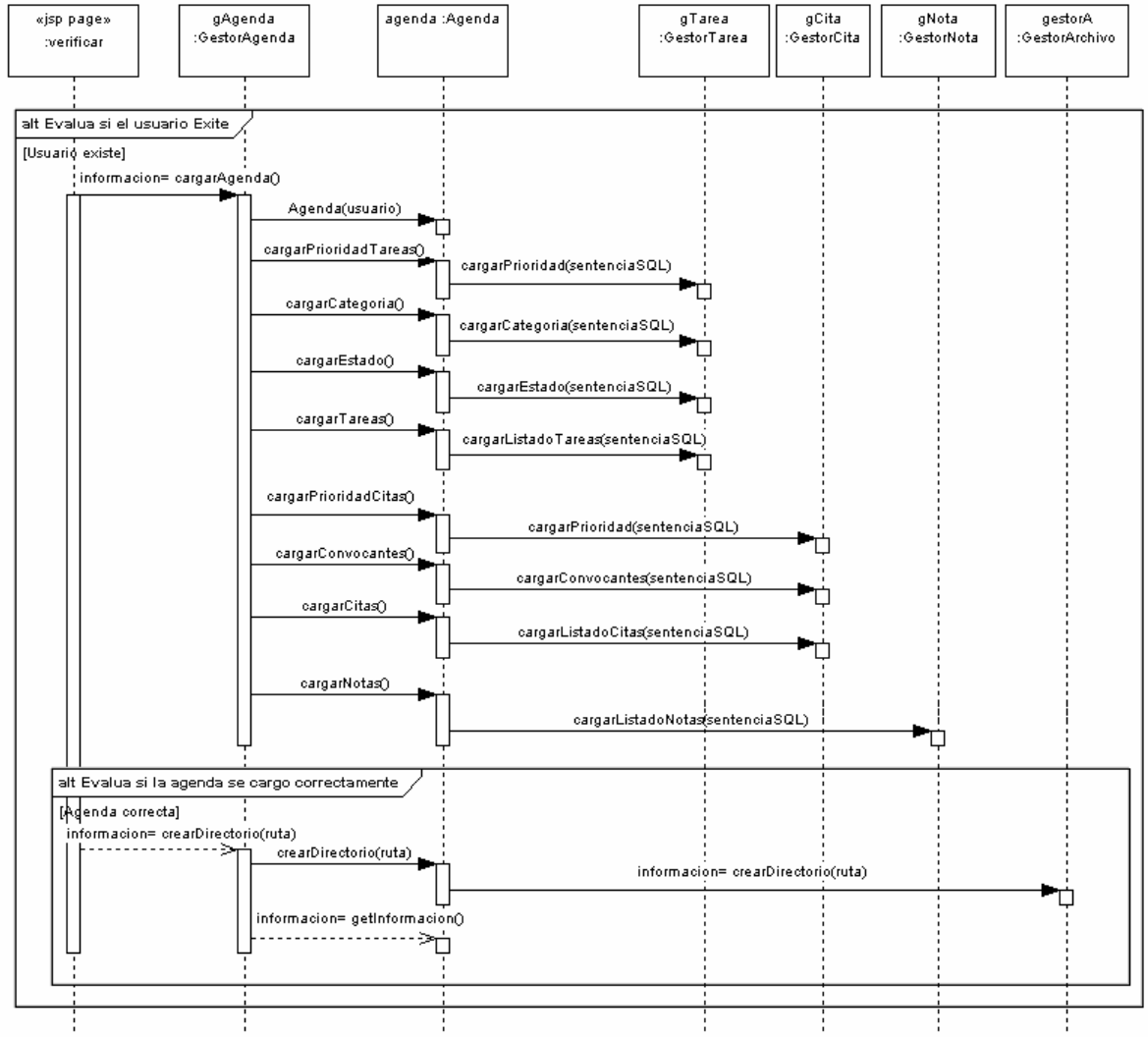


Documento	**Título**
DS3	Diagrama de secuencia - Módulo consulta de la agenda

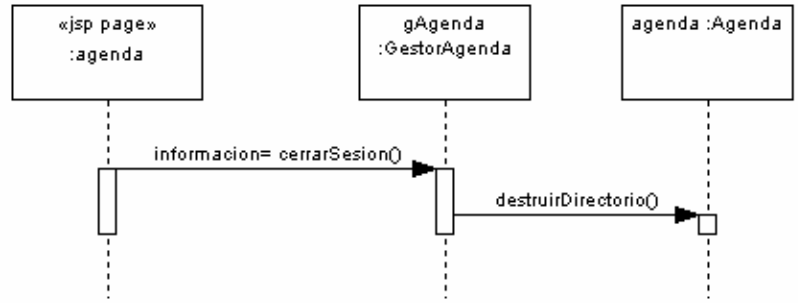
CO28	Mensaje= iniciarSesion(clave, login)



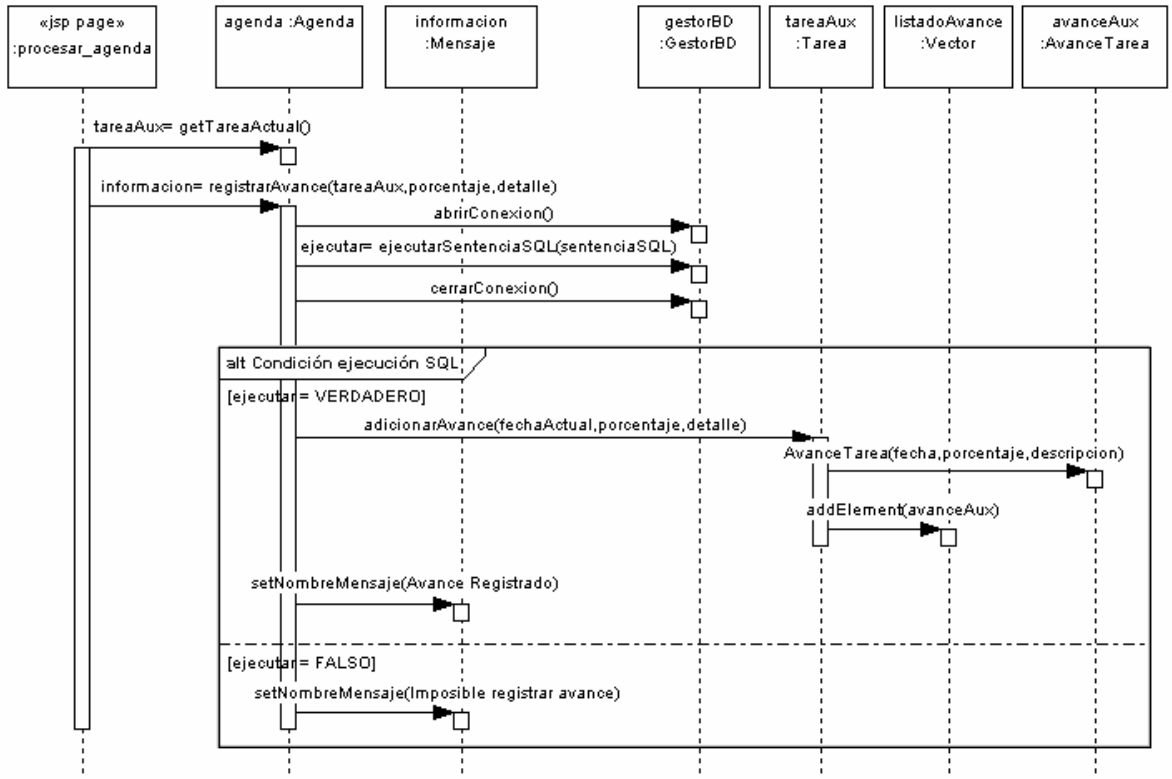
CO29	cargarAgenda()



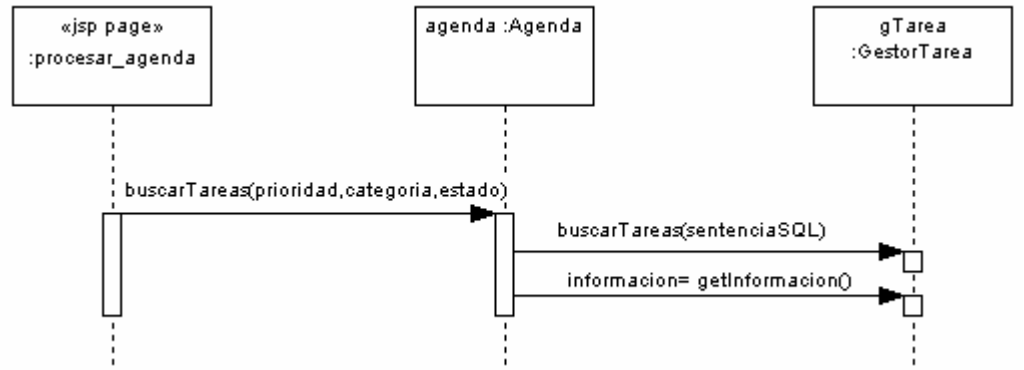
CO30	Mensaje= cerrarSesion()



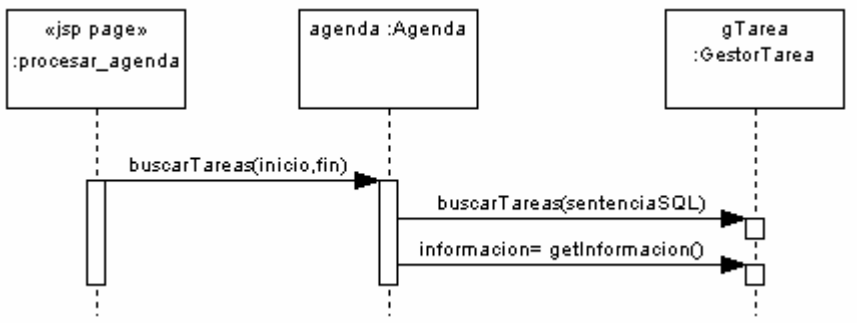
CO31	Mensaje= registrarAvance(detalle, porcentaje, tarea)



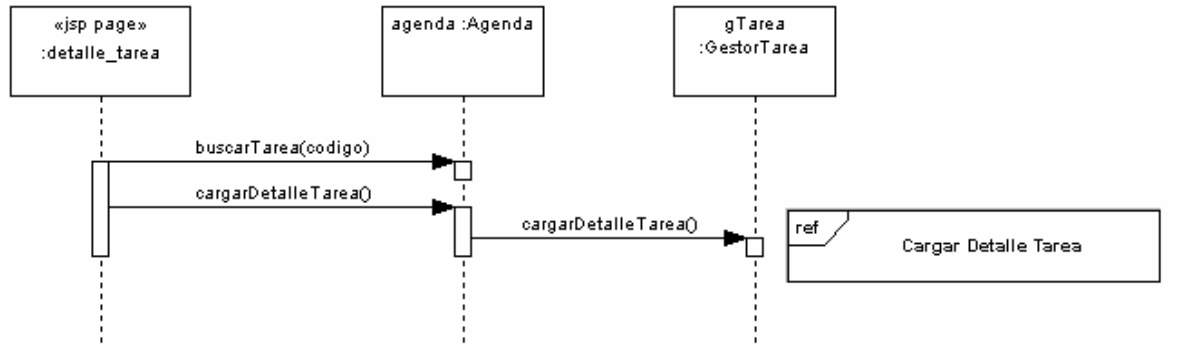
CO32	Mensaje= buscarTareas(prioridad, categoría, estado)



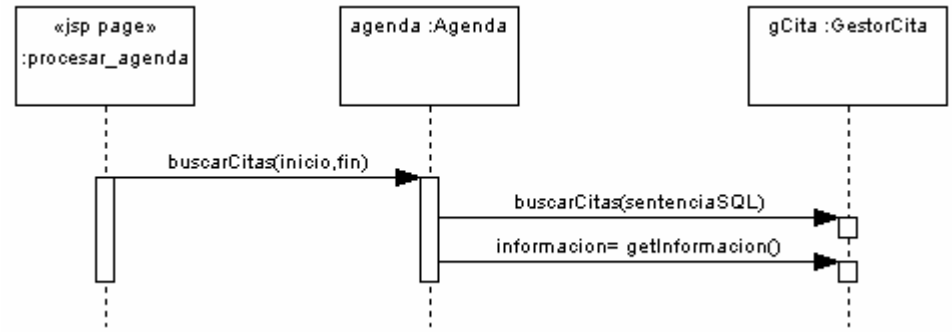
CO33	Mensaje= buscarTareas(fin, inicio)



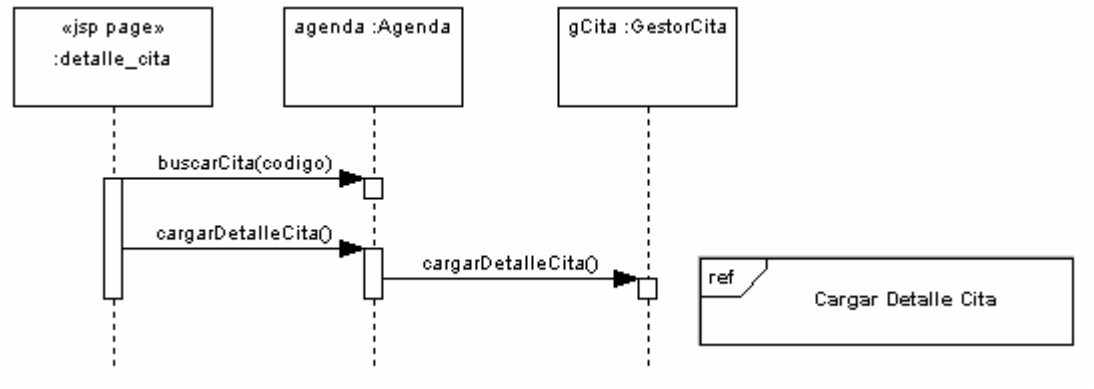
CO34	Mensaje= cargarDetalleTarea()



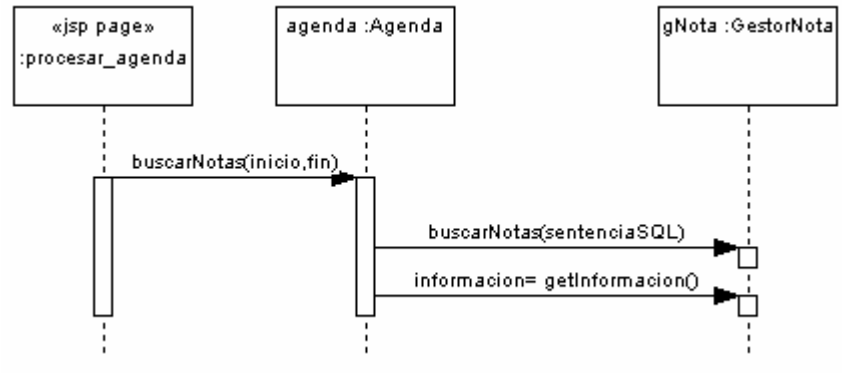
CO35	Mensaje= buscarCitas(fin, inicio)



CO36	Mensaje= cargarDetalleCita()



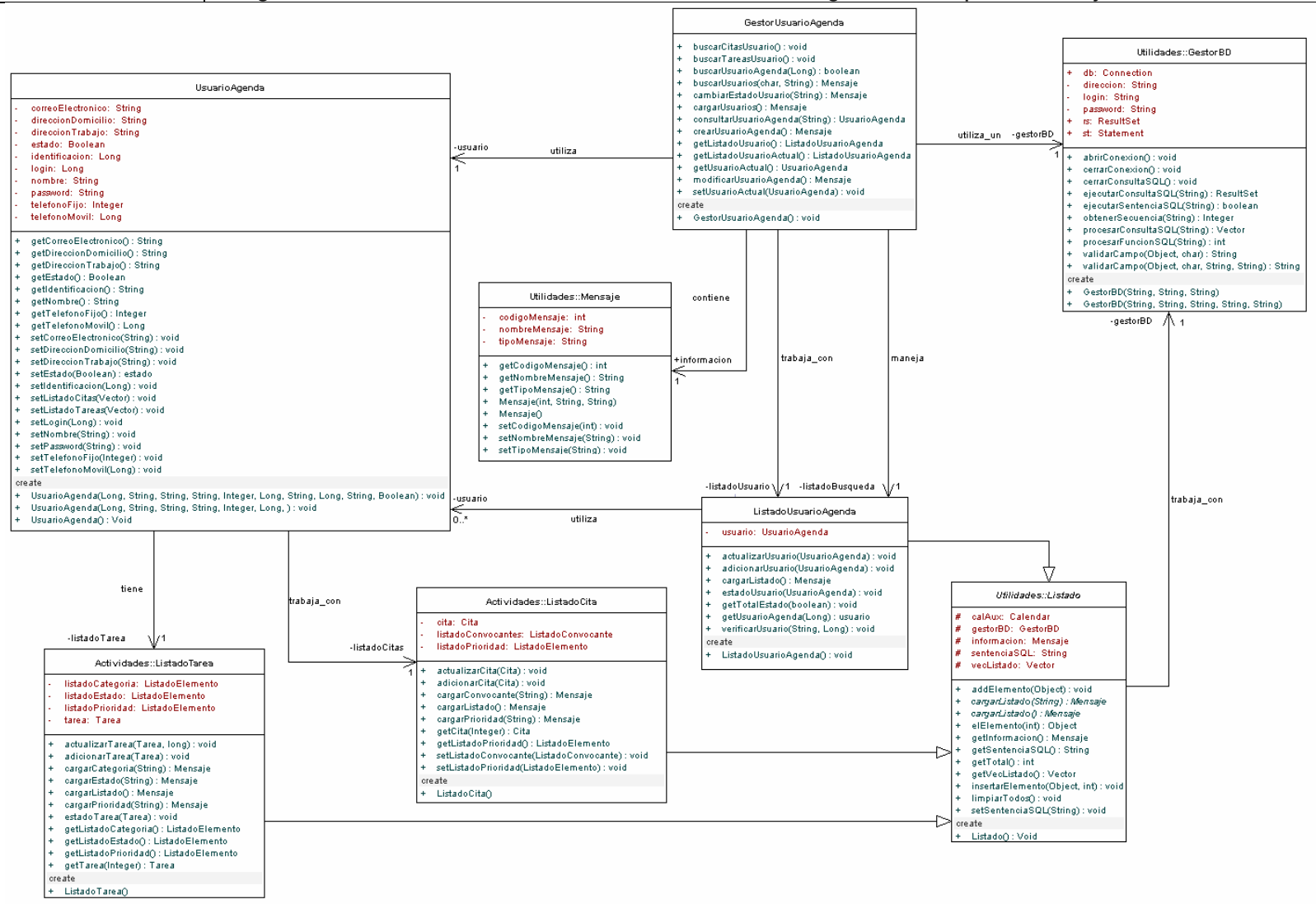
CO37	buscarNotas(fin, inicio)



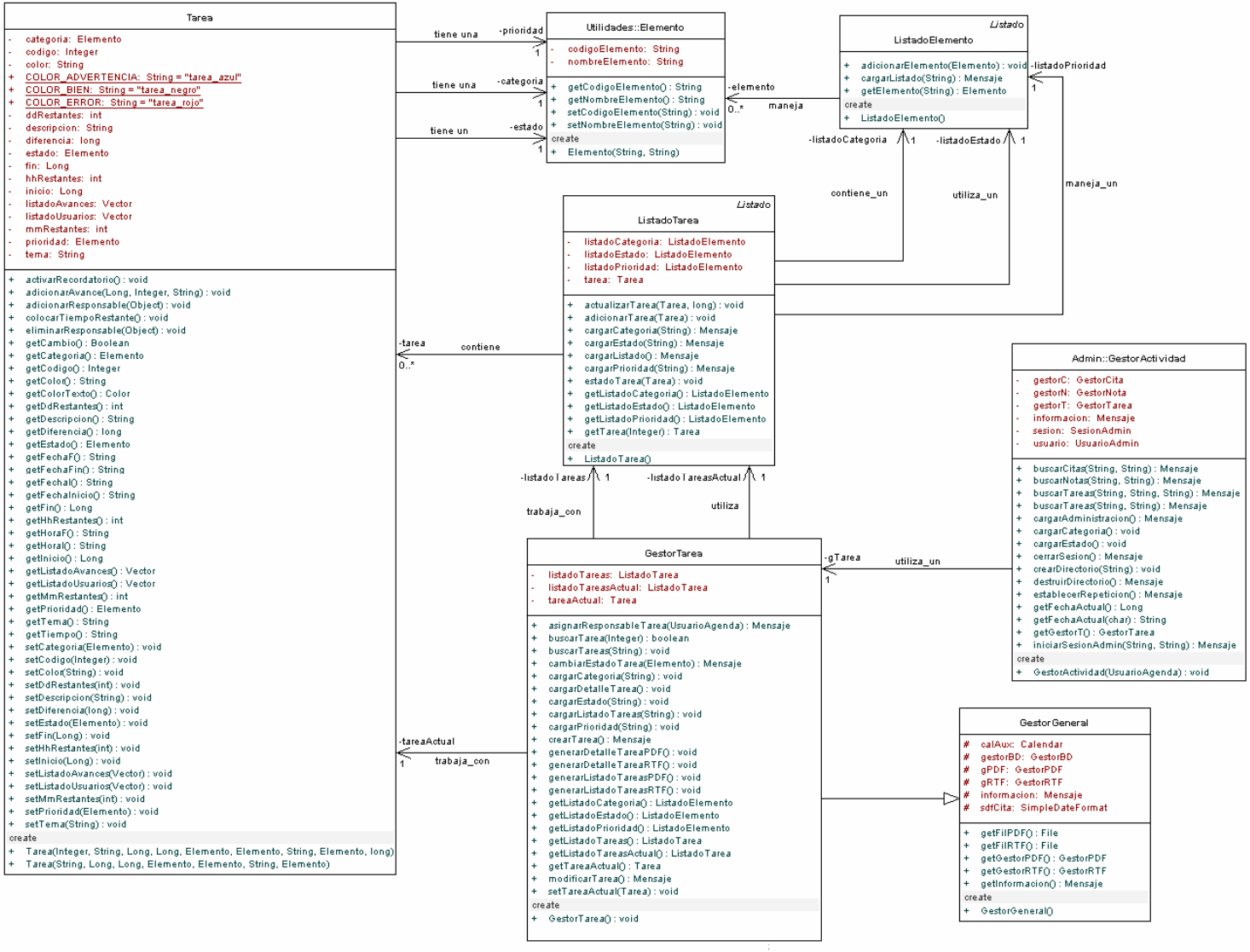
6.3.2 Diagrama de clases del diseño.

Figura 11. Diagrama de Clases del Diseño

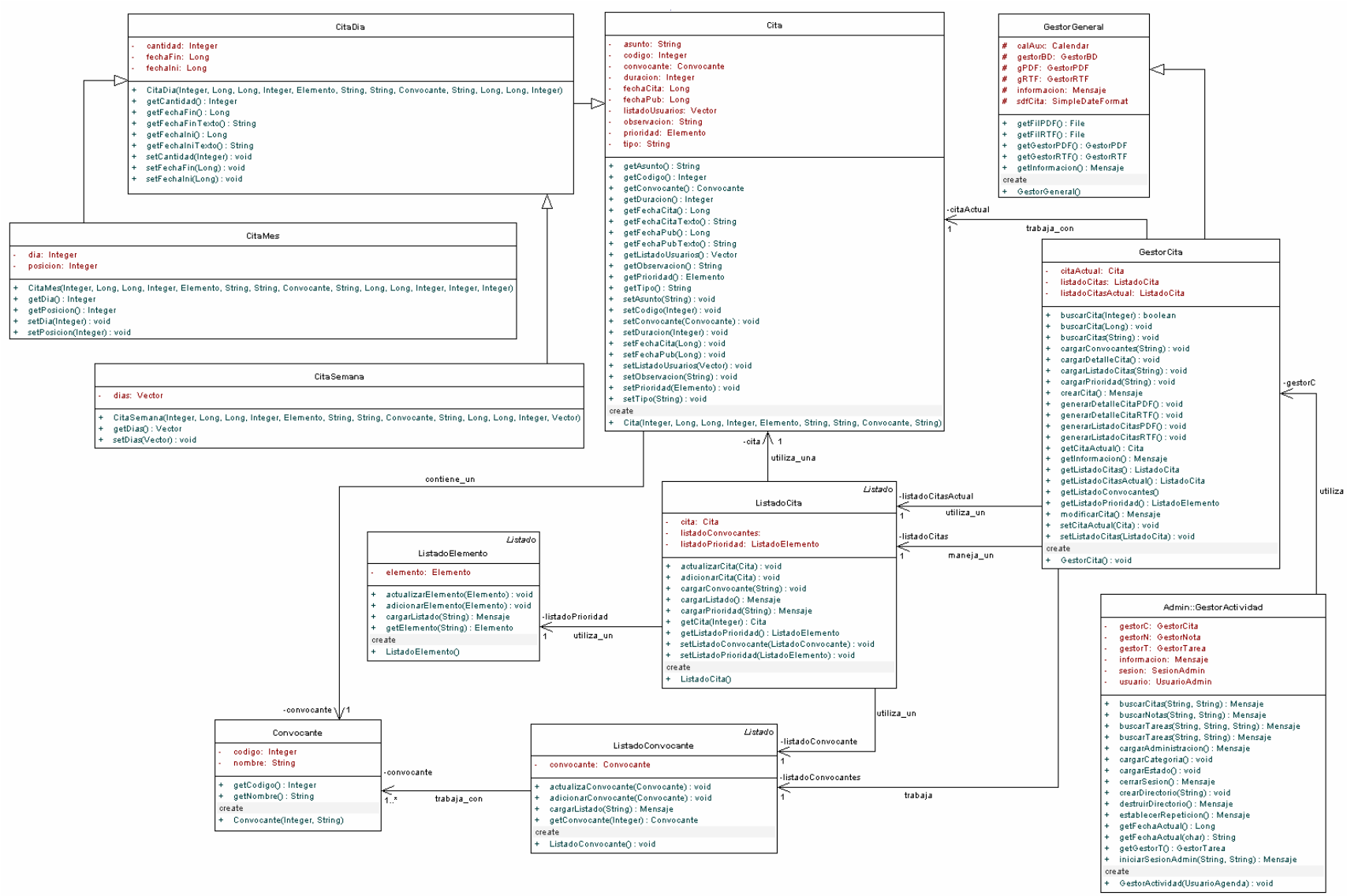
Documento	Título
DCD1	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Usuarios



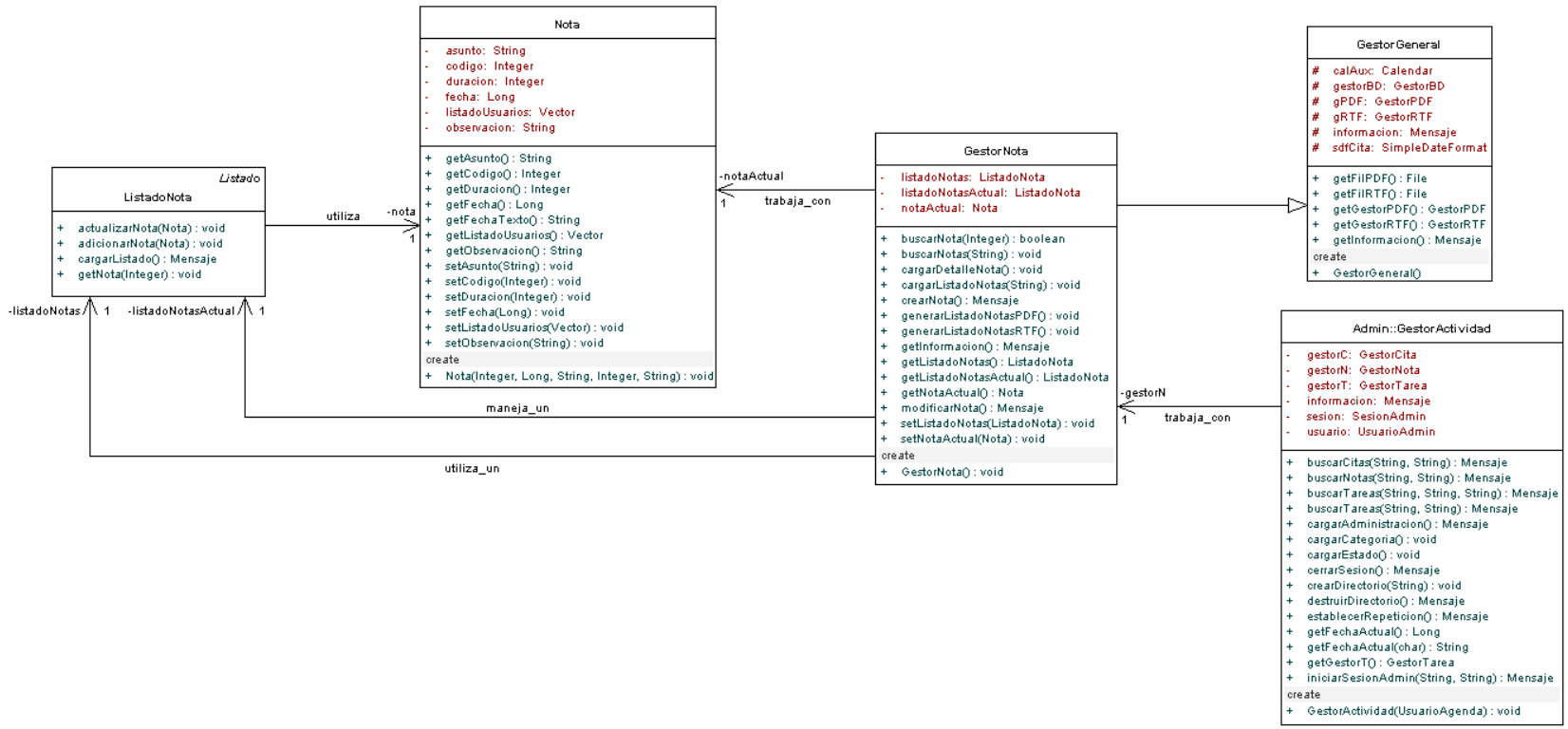
Documento	Título
DCD2	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades - Gestión de tareas



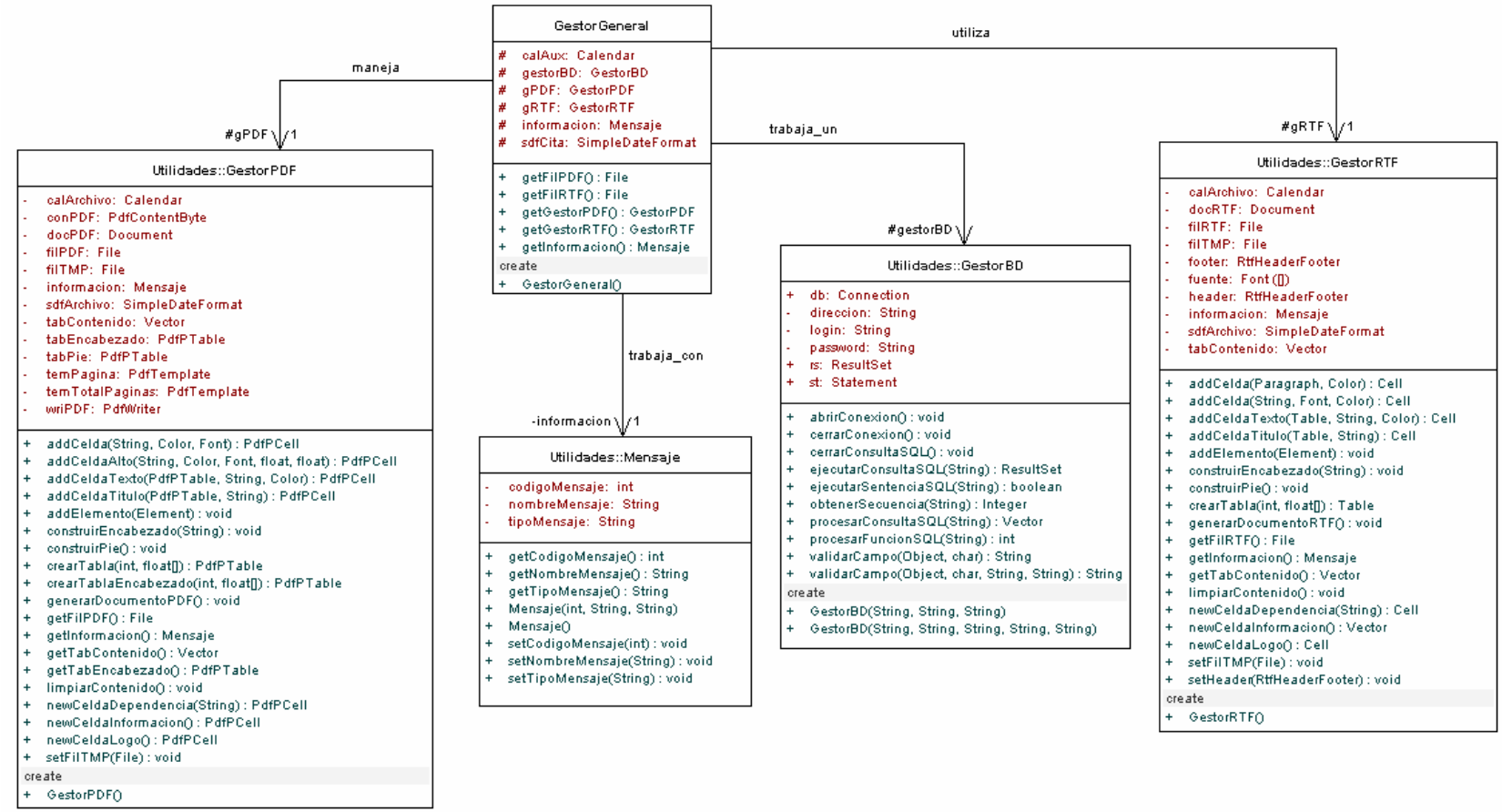
Documento	Título
DCD3	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades - Gestión de citas



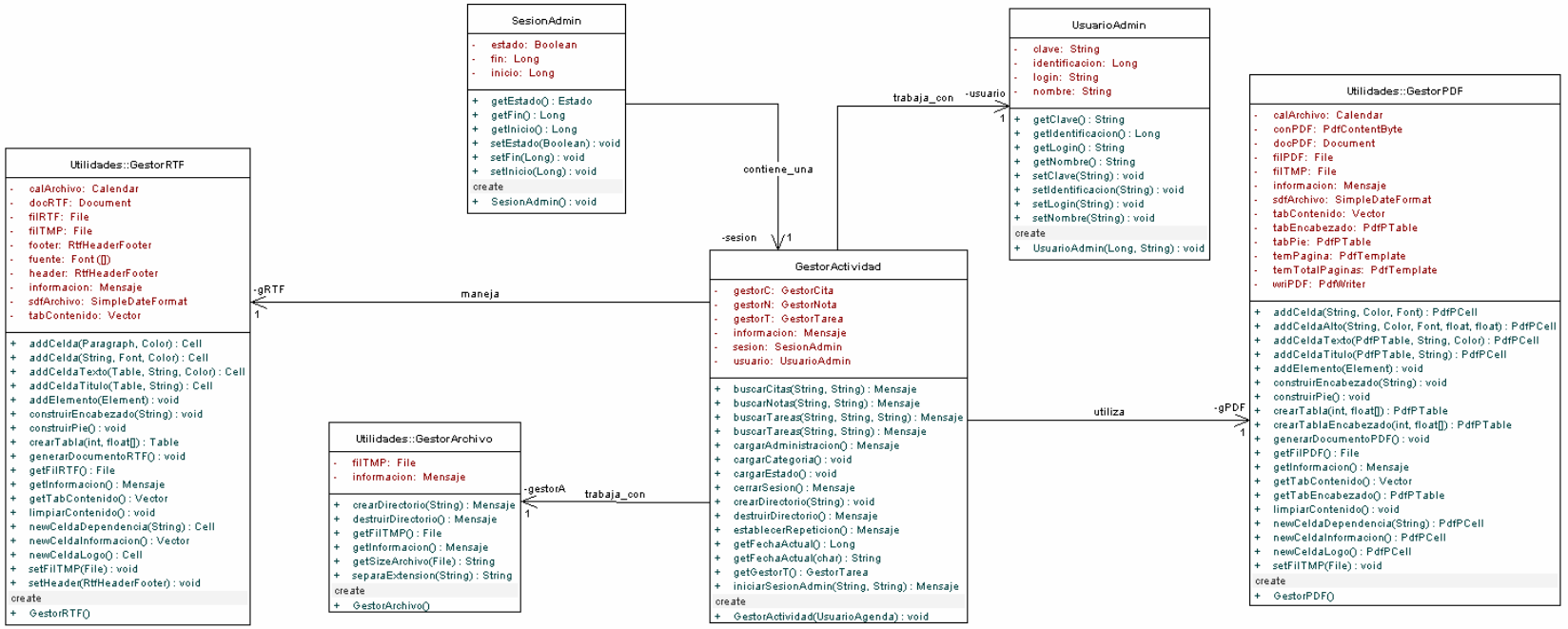
Documento	Título
DCD4	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades - Gestión de notas



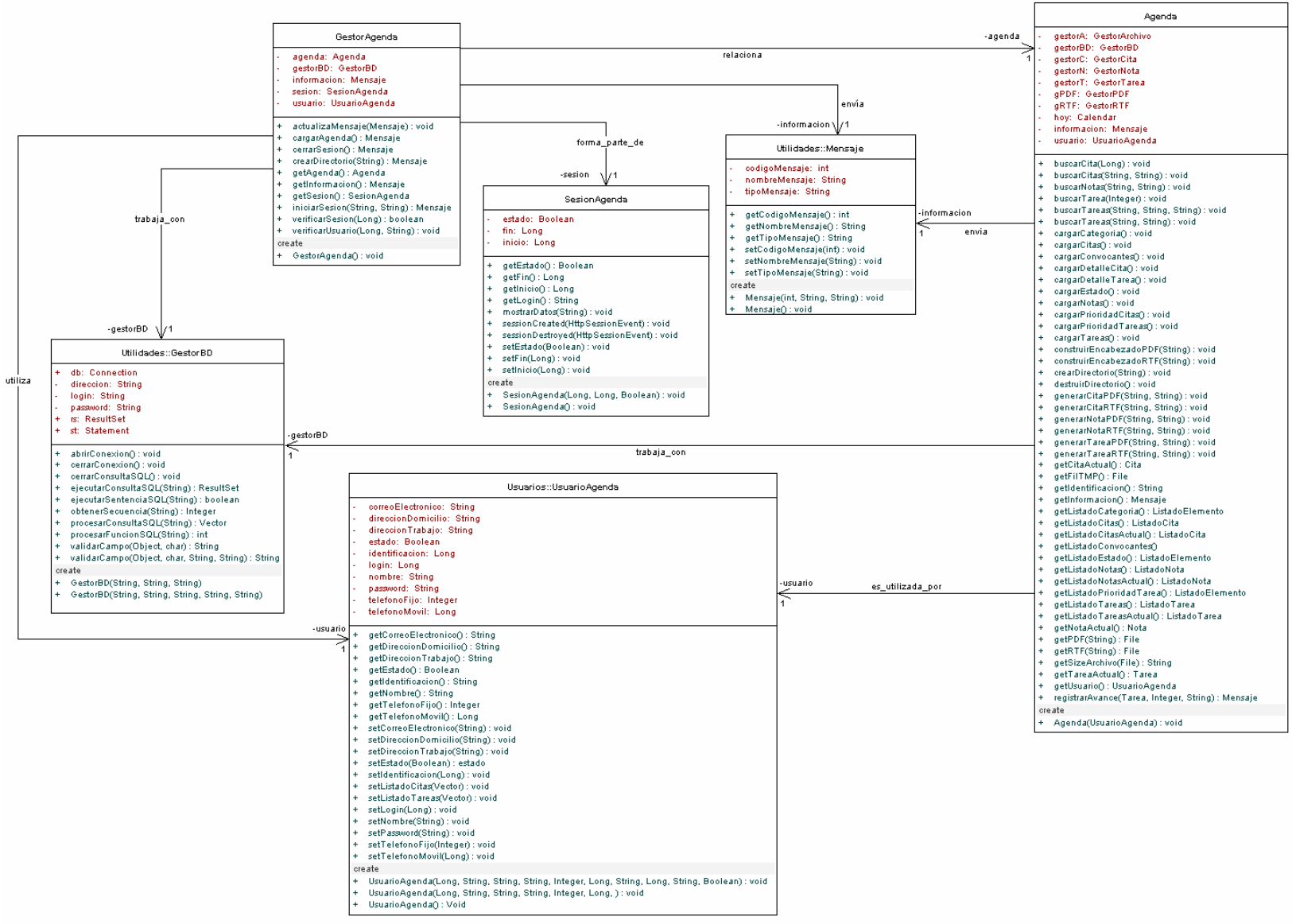
Documento	Título
DCD5	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades - Comportamiento común



Documento	Título
DCD6	Diagrama de clases - Módulo Administración de la agenda - Paquete Manejo de Actividades - Sección administración



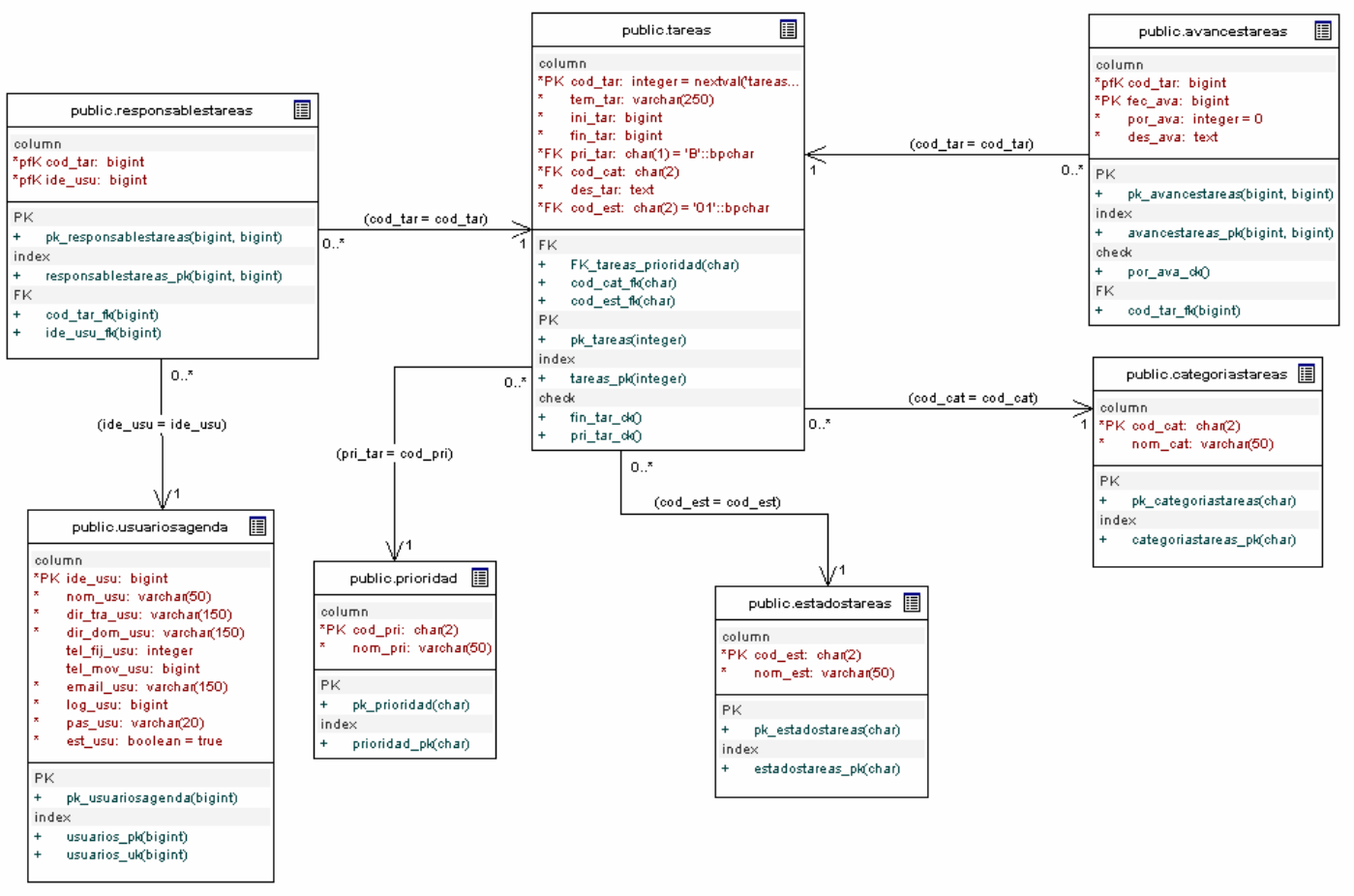
Documento	Título
DCD7	Diagrama de clases - Módulo consulta de la agenda



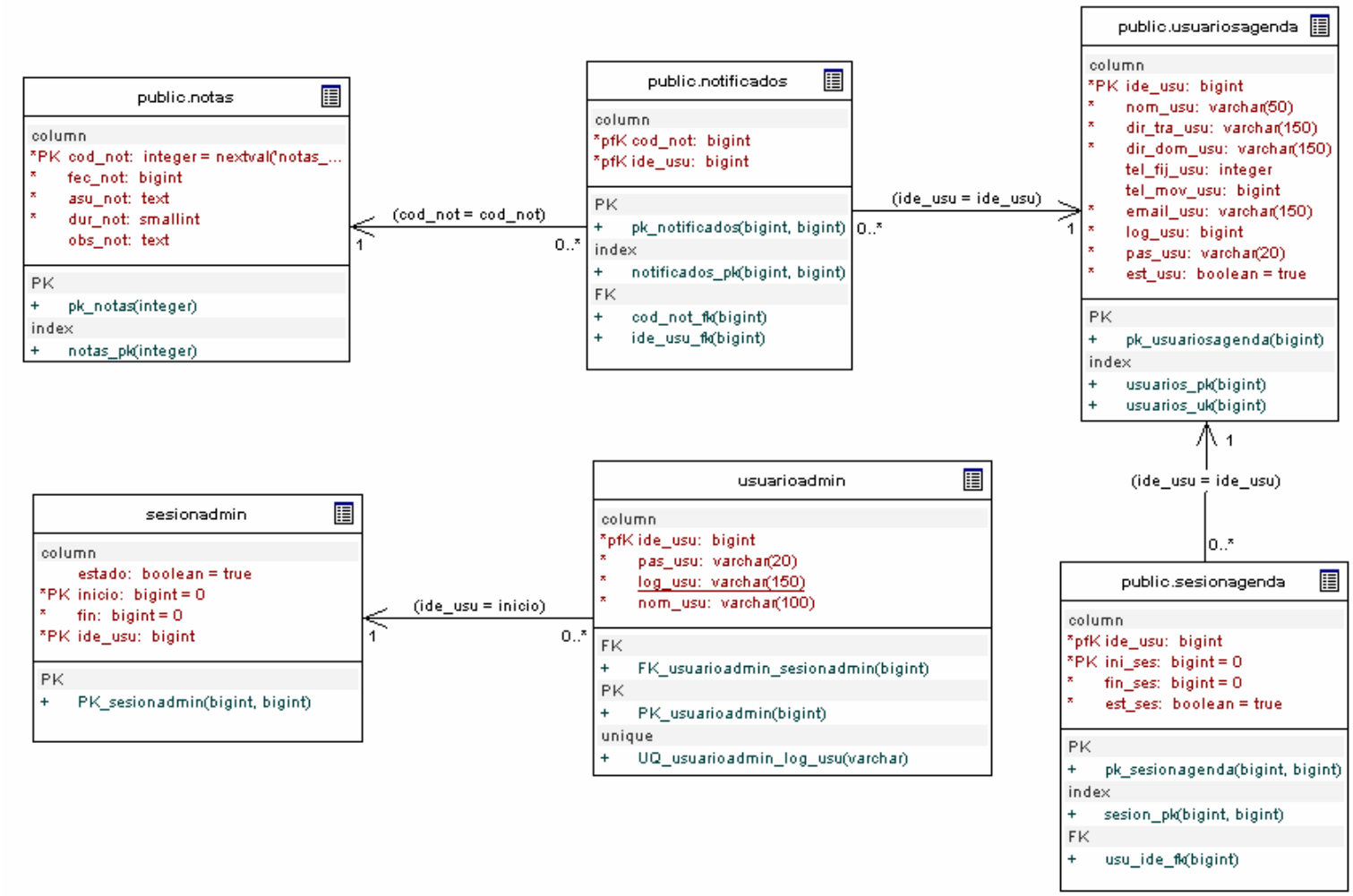

Agenda
- gestorA: GestorArchivo
- gestorBD: GestorBD
- gestorC: GestorCita
- gestorN: GestorNota
- gestorT: GestorTarea
- gPDF: GestorPDF
- gRTF: GestorRTF
- hoy: Calendar
- informacion: Mensaje
- usuario: UsuarioAgenda
+ buscarCita(Long) : void
+ buscarCitas(String, String) : void
+ buscarNotas(String, String) : void
+ buscarTarea(Integer) : void
+ buscarTareas(String, String, String) : void
+ buscarTareas(String, String) : void
+ cargarCategoria() : void
+ cargarCitas() : void
+ cargarConvocantes() : void
+ cargarDetalleCita() : void
+ cargarDetalleTarea() : void
+ cargarEstado() : void
+ cargarNotas() : void
+ cargarPrioridadCitas() : void
+ cargarPrioridadTareas() : void
+ cargarTareas() : void
+ construirEncabezadoPDF(String) : void
+ construirEncabezadoRTF(String) : void
+ crearDirectorio(String) : void
+ destruirDirectorio() : void
+ generarCitaPDF(String, String) : void
+ generarCitaRTF(String, String) : void
+ generarNotaPDF(String, String) : void
+ generarNotaRTF(String, String) : void
+ generarTareaPDF(String, String) : void
+ generarTareaRTF(String, String) : void
+ getCitaActual() : Cita
+ getFilTMP() : File
+ getIdentificacion() : String
+ getInformacion() : Mensaje
+ getListadoCategoria() : ListadoElemento
+ getListadoCitas() : ListadoCita
+ getListadoCitasActual() : ListadoCita
+ getListadoConvocantes()
+ getListadoEstado() : ListadoElemento
+ getListadoNotas() : ListadoNota
+ getListadoNotasActual() : ListadoNota
+ getListadoPrioridadTarea() : ListadoElemento
+ getListadoTareas() : ListadoTarea
+ getListadoTareasActual() : ListadoTarea
+ getNotaActual() : Nota
+ getPDF(String) : File
+ getRTF(String) : File
+ getSizeArchivo(File) : String
+ getTareaActual() : Tarea
+ getUsuario() : UsuarioAgenda
+ registrarAvance(Tarea, Integer, String) : Mensaje
create
+ Agenda(UsuarioAgenda) : void
-gestorT
maneja
1
GestorGeneral
Actividades::GestorTarea
- listadoTareas: ListadoTarea
- listadoTareasActual: ListadoTarea
- tareaActual: Tarea
+ asignarResponsableTarea(UsuarioAgenda) : Mensaje
+ buscarTarea(Integer) : boolean
+ buscarTareas(String) : void
+ cambiarEstadoTarea(Elemento) : Mensaje
+ cargarCategoria(String) : void
+ cargarDetalleTarea() : void
+ cargarEstado(String) : void
+ cargarListadoTareas(String) : void
+ cargarPrioridad(String) : void
+ crearTarea() : Mensaje
+ generarDetalleTareaPDF() : void
+ generarDetalleTareaRTF() : void
+ generarListadoTareasPDF() : void
+ generarListadoTareasRTF() : void
+ getListadoCategoria() : ListadoElemento
+ getListadoEstado() : ListadoElemento
+ getListadoPrioridad() : ListadoElemento
+ getListadoTareas() : ListadoTarea
+ getListadoTareasActual() : ListadoTarea
+ getTareaActual() : Tarea
+ modificarTarea() : Mensaje
+ setTareaActual(Tarea) : void
create
+ GestorTarea() : void
utiliza
-gestorC
1
GestorGeneral
Actividades::GestorCita
- citaActual: Cita
- listadoCitas: ListadoCita
- listadoCitasActual: ListadoCita
+ buscarCita(Integer) : boolean
+ buscarCita(Long) : void
+ buscarCitas(String) : void
+ cargarConvocantes(String) : void
+ cargarDetalleCita() : void
+ cargarListadoCitas(String) : void
+ cargarPrioridad(String) : void
+ crearCita() : Mensaje
+ generarDetalleCitaPDF() : void
+ generarDetalleCitaRTF() : void
+ generarListadoCitasPDF() : void
+ generarListadoCitasRTF() : void
+ getCitaActual() : Cita
+ getInformacion() : Mensaje
+ getListadoCitas() : ListadoCita
+ getListadoCitasActual() : ListadoCita
+ getListadoConvocantes()
+ getListadoPrioridad() : ListadoElemento
+ modificarCita() : Mensaje
+ setCitaActual(Cita) : void
+ setListadoCitas(ListadoCita) : void
create
+ GestorCita() : void
-gestorRTF
1
trabaja_con
Utilidades::GestorRTF
- calArchivo: Calendar
- docRTF: Document
- filRTF: File
- filTMP: File
- footer: RtfHeaderFooter
- fuente: Font ([])
- header: RtfHeaderFooter
- informacion: Mensaje
- sdfArchivo: SimpleDateFormat
- tabContenido: Vector
+ addCelda(Paragraph, Color) : Cell
+ addCelda(String, Font, Color) : Cell
+ addCeldaTexto(Table, String, Color) : Cell
+ addCeldaTitulo(Table, String) : Cell
+ addElemento(Element) : void
+ construirEncabezado(String) : void
+ construirPie() : void
+ crearTabla(int, float[]) : Table
+ generarDocumentoRTF() : void
+ getFilRTF() : File
+ getInformacion() : Mensaje
+ getTabContenido() : Vector
+ limpiarContenido() : void
+ newCeldaDependencia(String) : Cell
+ newCeldaInformacion() : Vector
+ newCeldaLogo() : Cell
+ setFilTMP(File) : void
+ setHeader(RtfHeaderFooter) : void
create
+ GestorRTF()
utiliza_un
-gestorN
1
GestorGeneral
Actividades::GestorNota
- listadoNotas: ListadoNota
- listadoNotasActual: ListadoNota
- notaActual: Nota
+ buscarNota(Integer) : boolean
+ buscarNotas(String) : void
+ cargarDetalleNota() : void
+ cargarListadoNotas(String) : void
+ crearNota() : Mensaje
+ generarListadoNotasPDF() : void
+ generarListadoNotasRTF() : void
+ getInformacion() : Mensaje
+ getListadoNotas() : ListadoNota
+ getListadoNotasActual() : ListadoNota
+ getNotaActual() : Nota
+ modificarNota() : Mensaje
+ setListadoNotas(ListadoNota) : void
+ setNotaActual(Nota) : void
create
+ GestorNota() : void
trabaja_con
-gPDF
1
Utilidades::GestorPDF
- calArchivo: Calendar
- conPDF: PdfContentByte
- docPDF: Document
- filPDF: File
- filTMP: File
- informacion: Mensaje
- sdfArchivo: SimpleDateFormat
- tabContenido: Vector
- tabEncabezado: PdfPTable
- tabPie: PdfPTable
- temPagina: PdfTemplate
- temTotalPaginas: PdfTemplate
- wriPDF: PdfWriter
+ addCelda(String, Color, Font) : PdfPCell
+ addCeldaAlto(String, Color, Font, float, float) : PdfPCell
+ addCeldaTexto(PdfPTable, String, Color) : PdfPCell
+ addCeldaTitulo(PdfPTable, String) : PdfPCell
+ addElemento(Element) : void
+ construirEncabezado(String) : void
+ construirPie() : void
+ crearTabla(int, float[]) : PdfPTable
+ crearTablaEncabezado(int, float[]) : PdfPTable
+ generarDocumentoPDF() : void
+ getFilPDF() : File
+ getInformacion() : Mensaje
+ getTabContenido() : Vector
+ getTabEncabezado() : PdfPTable
+ limpiarContenido() : void
+ newCeldaDependencia(String) : PdfPCell
+ newCeldaInformacion() : PdfPCell
+ newCeldaLogo() : PdfPCell
+ setFilTMP(File) : void
create
+ GestorPDF()

6.3.3 Diseño de datos

Figura 12. Diagramas Entidad Relación



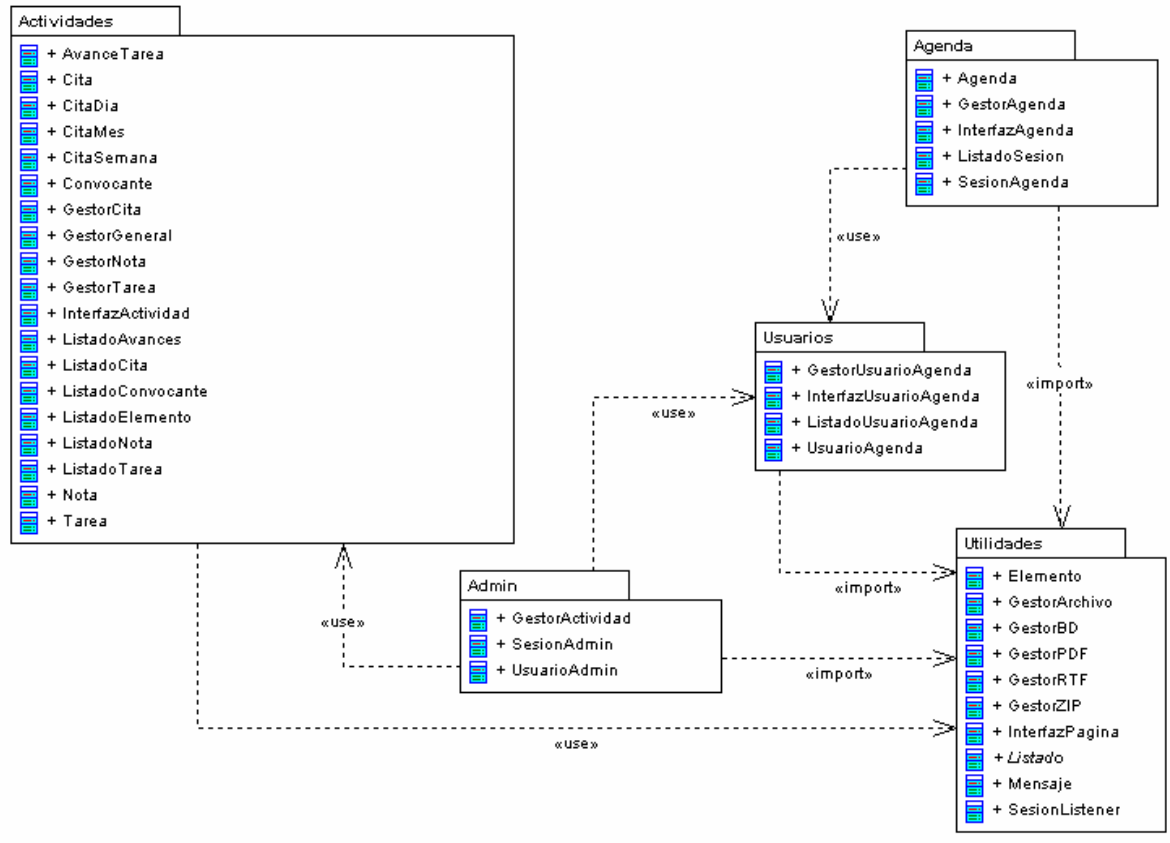

public.citas
column
*PK cod_cit: integer = nextval('citas_...
* fec_pub_cit: bigint
* fec_cit: bigint
* dur_cit: smallint
*FK pri_cit: char(1) = 'B'::bpchar
* asu_cit: text
obs_cit: text
FK cod_con: integer
* tip_cit: char(1) = 'U'::bpchar
FK
+ FK_citas_prioridad(char)
+ cod_con_fk(integer)
PK
+ pk_citas(integer)
index
+ citas_pk(integer)
check
+ dur_cit_ck()
+ fec_cit_ck()
+ pri_cit_ck()
+ tip_cit_ck()
public.citasdia
column
*pfK cod_cit: bigint
* can_cit: smallint
* fec_fin_cic: bigint
PK
+ pk_citasdia(bigint)
index
+ citasdia_pk(bigint)
FK
+ cod_cit_dia_fk(bigint)
public.prioridad
column
*PK cod_pri: char(2)
* nom_pri: varchar(50)
PK
+ pk_prioridad(char)
index
+ prioridad_pk(char)
public.citasmes
column
*pfK cod_cit: bigint
* can_cit: smallint
* dia_cit: smallint
* pos_dia: smallint
fec_fin_cic: bigint
PK
+ pk_citasmes(bigint)
index
+ citasmes_pk(bigint)
FK
+ cod_cit_mes_fk(bigint)
public.citassem
column
*pfK cod_cit: bigint
* can_cit: smallint
dia_cit:
fec_fin_cic: bigint
PK
+ pk_citassem(bigint)
index
+ citassem_pk(bigint)
FK
+ cod_cit_sem_fk(bigint)
public.convocantes
column
*PK cod_con: integer = nextval('convoc...
* nom_con: varchar(100)
PK
+ pk_convocantes(integer)
index
+ convocantes_pk(integer)
public.citados
column
*pfK cod_cit: bigint
*pfK ide_usu: bigint
FK
+ FK_citados_usuariosagenda(bigint)
+ cod_cit_fk(bigint)
+ ide_usu_fk(bigint)
PK
+ pk_citados(bigint, bigint)
index
+ citados_pk(bigint, bigint)
public.usuariosagenda
column
*PK ide_usu: bigint
* nom_usu: varchar(50)
* dir_tra_usu: varchar(150)
* dir_dom_usu: varchar(150)
tel_fij_usu: integer
tel_mov_usu: bigint
* email_usu: varchar(150)
* log_usu: bigint
* pas_usu: varchar(20)
* est_usu: boolean = true
PK
+ pk_usuariosagenda(bigint)
index
+ usuarios_pk(bigint)
+ usuarios_uk(bigint)
(cod_cit = cod_cit)
(pri_cit = cod_pri)
(cod_cit = cod_cit)
(cod_cit = cod_cit)
(cod_cit = cod_cit)
(cod_con = cod_con)
(ide_usu = ide_usu)
0..*
1


public.notas
column
*PK cod_not: integer = nextval('notas_...
* fec_not: bigint
* asu_not: text
* dur_not: smallint
obs_not: text
PK
+ pk_notas(integer)
index
+ notas_pk(integer)
(cod_not = cod_not)
1
0..*
public.notificados
column
*pfK cod_not: bigint
*pfK ide_usu: bigint
PK
+ pk_notificados(bigint, bigint)
index
+ notificados_pk(bigint, bigint)
FK
+ cod_not_fk(bigint)
+ ide_usu_fk(bigint)
(ide_usu = ide_usu)
0..*
1
public.usuariosagenda
column
*PK ide_usu: bigint
* nom_usu: varchar(50)
* dir_tra_usu: varchar(150)
* dir_dom_usu: varchar(150)
tel_fij_usu: integer
tel_mov_usu: bigint
* email_usu: varchar(150)
* log_usu: bigint
* pas_usu: varchar(20)
* est_usu: boolean = true
PK
+ pk_usuariosagenda(bigint)
index
+ usuarios_pk(bigint)
+ usuarios_uk(bigint)
1
(ide_usu = ide_usu)
0..*
sesionadmin
column
estado: boolean = true
*PK inicio: bigint = 0
* fin: bigint = 0
*PK ide_usu: bigint
PK
+ PK_sesionadmin(bigint, bigint)
(ide_usu = inicio)
1
0..*
usuarioadmin
column
*pfK ide_usu: bigint
* pas_usu: varchar(20)
* log_usu: varchar(150)
* nom_usu: varchar(100)
FK
+ FK_usuarioadmin_sesionadmin(bigint)
PK
+ PK_usuarioadmin(bigint)
unique
+ UQ_usuarioadmin_log_usu(varchar)
public.sesionagenda
column
*pfK ide_usu: bigint
*PK ini_ses: bigint = 0
* fin_ses: bigint = 0
* est_ses: boolean = true
PK
+ pk_sesionagenda(bigint, bigint)
index
+ sesion_pk(bigint, bigint)
FK
+ usu_ide_fk(bigint)

6.4 MODELO DE IMPLEMENTACIÓN

6.4.1 Diagrama de paquetes

Figura 13. Diagrama de paquetes



7. CONCLUSIONES

Como se puede apreciar en el desarrollo del proyecto, la herramienta informática que apoya a este sistema de información se considero de acuerdo a las necesidades significativas que presenta la Facultad de Ingeniería, y más precisamente el Departamento de Sistemas perteneciente a esta dependencia, en cuanto al control de actividades académicas y administrativas se refiere.

Es así como el módulo administrativo desarrollado, permite registrar, modificar, eliminar y consultar tareas, citas y notas, relacionadas con actividades que se desarrollan en el departamento de forma eficiente. Permite saber el estado de desarrollo de una actividad, proceso muy importante para conocer en que etapa se encuentra una tarea encargada a un docente o estudiante, asunto que antes era dispendioso y lento.

De igual forma, este modulo permite manejar la información de todos los integrantes del Departamento o Facultad, sean funcionarios, docentes o estudiantes, a quienes se puede asignar tareas y citas, como notas. Cada uno tiene una cuenta única con la cual puede manejar su agenda.

En el módulo de consulta de la agenda, tal como se planeó, el usuario puede consultar fácilmente y por diferentes criterios, resaltando el relacionado con periodos de tiempo, las tareas, actividades, citas y notas, guardar reportes e imprimir los que considere importantes.

La implementación considerando a la Web, como medio para establecer la comunicación entre los usuarios y la facultad, es conveniente para mantener a los interesados enterados de los sucesos que se den en la Facultad.

Este sistema como esta planteado, realizando los ajustes apropiados, puede formar parte de otros sistemas de información donde el control de actividades sea significativo como la gestión de proyectos, dependencias de empresas donde se necesite distribuir actividades y en general, como herramienta de comunicación entre los integrantes de una misma dependencia.

8. RECOMENDACIONES

El desarrollo de este proyecto deja claro muchas recomendaciones:

Realizar una segunda versión del sistema. Era inevitable que en el desarrollo del proyecto nuevos requerimientos aparecieran como el manejo de grupos de usuarios, el envío de correos electrónicos a usuarios, el registro de soportes de documentos que generan actividades, entre otros. Sin embargo, queda claro que el objetivo principal de este proyecto se cumplió totalmente y que dependiendo de la acogida que tenga este proyecto por las directivas del Departamento de Sistemas como de la Facultad, se puede realizar una segunda versión del mismo, en la cual estas funcionalidades y otras tantas, sean incorporadas.

Construir lo más pronto posible un Plan General para el Desarrollo de Software en la Universidad de Nariño, que establezca la orientación del desarrollo de software en cualquier dependencia de la institución, es decir, fije las reglas en cuanto a plataforma software y hardware, la metodología clara para el desarrollo, entre otras. Pero sobre todo, un plan que contemple a cinco o siete años cuales son las necesidades trascendentales relacionadas con sistemas de información que la Universidad manifieste, y que sobre estas necesidades se fijen proyectos de desarrollo de software que aprovechen el esfuerzo de los estudiantes al realizar sus trabajos de grado.

Implementar esta herramienta para realizar pruebas y permitir una retroalimentación eficaz.

BIBLIOGRAFÍA

D'SOUZA, Desmond y WILLS,Alan. Objects, Components, and Frameworks With Uml: The Catalysis Approach (Addison-Wesley Object Technology Series). 1998

KENDALL, Kenneth E. Y KENDALL, Julie E. Análisis y Diseño de Sistemas. 3ª Ed. México DF: McGRAW-HILL. 1992. p. 913

LARMAN, Craig. Uml y Patrones 2ª Ed. Madrid: PRENTICE HALL. PEARSON EDUCACION. S.A. 2003. p. 591

MOHR, James. LINUX Recursos para el usuario. México DF: PRENTICE HALL HISPANOAMERICANA, S.A. 1997. p. 825

PRESSMAN, S. Roger. Ingeniería del Software. Un enfoque práctico. 5ª Ed. Madrid: McGRAW-HILL/INTERAMERICANA. 2002. p. 601.

MARTRA, Pere. Introducción a UML. España. (1999-2005). Disponible desde Internet: <http://www.programacion.com/tutorial/uml/1/#uml_intro>.

WIKIPEDIA. AJAX. POSTGRESQL. TOMCAT. JAVA. Estados Unidos. El contenido está disponible bajo los términos de la GNU Free Documentation License. Disponible desde Internet: <http://es.wikipedia.org/wiki/Mysql>.