UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

SISTEMA DE GESTIÓN DE INFORMACIÓN PARA LA UNIDAD DE APRENDIZAJE AUTÓNOMO – IDIOMAS DE LA ESCUELA NACIONAL DE ENFERMERÍA Y OBSTETRICIA DE LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

T E S I S

Que para obtener el Título de

INGENIERO EN COMPUTACIÓN

Presentan:

DÍAZ NAVARRO VÍCTOR BERNARDO

ULLOA MONTOYA HÉCTOR

VERGARA ALCÁNTARA ARELY ABIGAIL

Director de Tesis: ING. JUAN JOSÉ CARREÓN GRANADOS

México, D.F. 2004.

En primer lugar quiero agradecer a cuatro personas sumamente importantes en mi vida:

A mi ***MAMÁ***, gracias por tu tiempo, paciencia, comprensión y apoyo. Gracias por dejarme ser.

A mis ***ABUELOS***, sin ustedes no hubiera llegado hasta dónde estoy hoy. Gracias por su paciencia, su apoyo, su cariño y su comprensión.

A mi ***HERMANA***, por su cariño y paciencia, aunque la distancia nos mantiene lejos, siempre estas conmigo.

A DIOS por todo lo que me ha dado y me ha permitido hacer.

A ustedes dedico este trabajo. Gracias por todo.

También quiero agradecer a Víctor Bernardo y Héctor, porque nos tuvimos una paciencia tremenda a lo largo de este trabajo, y por fortuna llegamos juntos al final. Gracias por tooooooodo.

Esta Tesis va dedicada a Jonathan, Said, Adriana, Coral, Miguel, Luis, Ana, Caro, Diego, Samantha y Andrea; porque ustedes como mis primos siempre están en mis pensamientos y esta es una muestra de que se puede legar al final de una meta; échenle ganas!!.

A Alfredo, Soledad, María de la Luz, Francisco, Lilia, Pedro, Juan Carlos, Guadalupe, Alejandro, María, Andrés, Rocío, Graciela y Jaime, porque como tíos me han enseñado gran parte de lo que soy y siempre me han apoyado.

Y por supuesto a Eric, Alejandro, Luis, Julio, Iliana, Verónica, Alfonso, Gonzalo, Gabriel; que desde el principio de la carrera estuvieron conmigo apoyándome en todo momento hasta la fecha.

A tooooodos y cada uno de mis amigos en la Facultad, prefiero omitir nombres para no olvidar a alguno, pero saben que siempre me acuerdo de ustedes: computologos, electrónicos, de tele, industriales, geofísicos, geólogos, petroleros, civiles.

A todos los amigos del CELE gracias por la paciencia y el apoyo.

A los amigos ajenos a la UNAM, gracias por su comprensión y apoyo, en especial a Elisa, Francisco, Leticia, Mónica, Mayra, Marco, Tere, Zurisadai.

Al Ing. Miguel Eduardo González Cárdenas, por el apoyo y el interés que puso en mi desde el inicio de la carrera.

ARELY ABIGAIL VERGARA ALCANTARA

A mis padres, Héctor y Ana María
sin ustedes nada de esto hubiera sido posible.

A la UNAM por formar librepensadores.

A la Facultad de Ingeniería
por todos los conocimientos dados.

A todas las personas que de manera directa
o indirecta contribuyeron a esta tesis.

Héctor Ulloa Montoya

A mis papas, gracias a ustedes lleve a cabo este sueño.

A mi tía Ofelia por su gran paciencia y sus sabios consejos.

A Mayra por todo el amor brindado.

A mis compañeros de tesis.

A la UNAM que me ha dado las herramientas para abrirme pasó en la vida.

Víctor Bernardo Díaz Navarro

Antes que nada queremos agradecerles a ***NUESTROS PADRES*** por el esfuerzo que han hecho por nosotros, que ha permitido que nos desarrollemos como personas.

De manera particular, queremos agradecer el apoyo que nos han brindado para el desarrollo de este trabajo a:

Ing. Carlos Alberto Román Zamitis, porque desde un inicio se intereso en este proyecto y nos dio su apoyo, parte de su tiempo y resolvió gran parte de nuestras dudas a lo largo del desarrollo de este trabajo.

A la ***Escuela Nacional de Enfermería y Obstetricia***, en particular a los ***Administradores*** de la ***Unidad de Aprendizaje Autónomo – Idiomas***, ya que gracias a su confianza y apoyo pudimos llevar a cabo este trabajo, confiando en haber aportado una herramienta útil para ellos.

A la ***Coordinadora***, ***Administradores***, ***Personal*** y ***Prestadores de Servicio Social*** de la ***Mediateca*** y el ***Centro de Apoyo a la Docencia*** del ***Centro de Estudio de Lenguas Extranjeras***, por habernos proporcionado información, infraestructura, apoyo y tiempo, ya que fue aquí en donde se llevo a cabo gran parte de este trabajo.

A nuestros ***Sinodales*** por habernos apoyado en el proceso final de este trabajo.

A nuestra ***Facultad de Ingeniería*** por darnos el apoyo y la infraestructura que nos permitió formarnos como Ingenieros.

A nuestra ***Universidad*** por habernos permitido ser estudiantes de esta casa de estudios.

ÍNDICE

INTRODUCCIÓN

Planteamiento y Contexto del Problema

La Unidad para el Aprendizaje Autónomo - Idiomas de la Escuela Nacional de Enfermería y Obstetricia (UAAI – ENEO) es un centro que dará apoyo en el aprendizaje de idiomas. La creación de este centro se ha basado en la Mediateca del Centro de Enseñanza de Lenguas Extranjeras de la Universidad Nacional Autónoma de México (CELE – UNAM).

Este tipo de centros requiere de un riguroso control en cuanto a la manipulación de información, la administración del lugar y el control de acceso, si carece de un sistema de control, es muy probable que se presenten problemas, por ejemplo:

* Control de Usuarios de la Mediateca:
  - Retraso en el proceso de Inscripción
  - Retraso en la elaboración de credenciales
  - Pérdida de tiempo en la revisión de la validez de inscripción de cada usuario (se revisa de manera manual).
* Registro de Materiales de Acervo:
  - Falta de integridad de los datos ya que pueden existir campos nulos o repetición de registros, que no permiten búsquedas exactas de la información.

La Unidad para el Aprendizaje Autónomo - Idiomas de la Escuela Nacional de Enfermería y Obstetricia (UAAI), pretende poner al alcance de los estudiantes de la institución un idioma, (inicialmente), desde la comodidad de su escuela y con la ventaja de un horario definido por ellos mismos; lo que les asegurará la práctica del idioma según su propio ritmo y disponibilidad.

Motivo por el cual es necesario que tanto la información ofrecida en la UAAI, como el acceso a esta se hagan de manera dinámica, esto requiere de un sistema de control que proporcione a los usuarios y administradores la seguridad y agilidad de la información al momento de encontrarse en ésta.

Entre los requisitos solicitados por la Administración de la UAAI, esta el hecho de que el sistema debe ser desarrollado en algún lenguaje de programación libre y que pueda ser ejecutado sobre cualquier plataforma. Por tanto el lenguaje de programación adoptado es: Java.

La mayoría de las tareas que el sistema realizará han sido sugeridos por el personal administrativo de la Mediateca del CELE, debido a que ellos han estado colaborando con el personal de la UAAI.

Objetivo

Desarrollar un sistema de cómputo que realice la gestión de información, los procesos administrativos y de control de acceso de la Unidad para el Aprendizaje Autónomo - Idiomas de la Escuela Nacional de Enfermería y Obstetricia.

Justificación

Debido al manejo delicado de información en la UAAI, es necesaria la creación de un sistema de gestión de información, para automatizar procesos como:

1. Inscripción y elaboración de credenciales de usuario.
2. Control de acceso de usuarios a la UAAI.
3. Registro de materiales del acervo (libros de texto, diccionarios, publicaciones periódicas, CD – ROM's, audiocassettes, videocassettes, programas televisivos y fichas de trabajo) y su posterior consulta por parte de los usuarios.
4. Generación de reportes que apoyen la planeación y toma de decisiones de la coordinación de la Unidad.
5. Evitar errores que van desde omisión de datos hasta duplicación de registros, además, existe traspapeleo, falta de disponibilidad inmediata de la información y pérdida de la misma, debido al carácter manual del registro de esta.

El desarrollo de este sistema tiene como mayor aportación la agilización de procesos de administración de información en la UAAI, lo cual se verá en el ahorro de tiempo de estos procesos para los administradores. Una de las metas a largo plazo de este proyecto es su implementación en alguna otra Unidad de las ya existentes. Por ello una de las características de este sistema será su modularidad para que en caso de que este se desee aplicar en otras Unidades de Autoacceso, cada una pueda modificarlo acorde a las necesidades de ella, sin necesidad de crear un nuevo sistema.

Por diversas características de lenguaje mismo y las restricciones establecidas, se ha optado por desarrollar en el lenguaje JAVA.

De Java se puede decir que es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems en 1991. Este lenguaje se diseñó para ser pequeño, portátil a través de plataformas y sistemas operativos, tanto a nivel de código fuente como en binario.

Entre los motivos por los cuales se ha optado por este lenguaje tenemos:

* Es independiente de la plataforma: Independencia de la plataforma es la capacidad del programa de trasladarse con facilidad de un sistema de computo a otro. Java mantiene esta independencia de la plataforma tanto a nivel del código fuente como a nivel de código binario.

El ambiente de desarrollo Java tiene dos partes: un compilador y un intérprete Java. El compilador Java toma el programa Java y en lugar de generar códigos de máquina para los archivos fuentes genera bytecode.

* Java está orientado a objetos: La técnica de programación orientada a objetos (OOP, por sus siglas en inglés) sólo es una manera de organizar programas, lo que puede conseguirse con cualquier lenguaje. Sin embargo, trabajar en un ambiente real de programación y lenguaje orientado a objetos, permite utilizar en su totalidad las ventajas de la metodología orientada a objetos, además de sus capacidades para crear programas flexibles y modulares, así como reutilizar un código.

Organización del Documento

En el presente documento se describe en seis capítulos el desarrollo del Sistema que cumpla con los requisitos de la UAAI.

En el *Capítulo I: Centros de Autoacceso (Mediateca -CELE y UAAI – ENEO)*, se hace referencia de estos, con la finalidad de analizar las necesidades de cada uno.

En el *Capítulo II: Justificación del Sistema basado en teoría de desarrollo de Software*, se hace un análisis de las diferentes metodologías para desarrollo, se indica los diferentes pasos que integran el Ciclo de Vida del Software entre otros aspectos referentes a la implementación de Sistemas.

En el *Capítulo III: Modelo de Requisitos*, se presenta un panorama general de los requisitos del Sistema para la UAAI, con un estudio, mediante el modelo de Espiral.

En el *Capítulo IV: Modelo de Análisis*, se hace referencia a un detallado análisis de lo que será el Sistema de la UAAI, basados en los requerimientos del Modelo de Requisitos.

En el *Capítulo V: Modelo de Diseño*, se lleva a cabo el diseño lógico del sistema en cuanto a la futura presentación de pantallas y secuencias de procesos que deberá seguir, basándose en detalles específicos que nos aproximen a la implementación del propio Sistema.

En el *Capítulo VI: Implementación*, se hacer referencia, a grandes rasgos, de la metodología que se tuvo para el desarrollo final del Sistema.

Finalmente, se hace alusión a las conclusiones relativas a este trabajo.

CAPÍTULO 1
UNIDADES DE AUTOACCESO (MEDIATECA-CELE Y UAII-ENEO)

1.1 UNIDADES DE AUTOACCESO

Las Unidades ó Centros de Autoacceso, son lugares donde se pretende ayudar a los estudiantes de educación superior en el aprendizaje. El objetivo de estos centros es promover la idea del aprendizaje autodirigido.

En el caso particular de esta tesis, se analizará el caso específico de un centro de Autoacceso para la Escuela Nacional de Enfermería y Obstetricia, en lo referente al área de Idiomas.

Los centros de autoacceso son espacios donde el estudiante tiene un papel activo en su formación, y cuenta con la libertad para elegir cuánto tiempo quiere dedicar al estudio, así como elegir qué estudiar y cómo hacerlo. Para esto cuenta con medios y materiales preparados para este fin, así como asesoría si la requiere.

Los centros de autoaprendizaje han evolucionado de tal forma que se puede sintetizar su evolución en la Tabla 1

Evolución de los centros de autoaprendizaje				
Indicadores	**Autonomía**	**Estrategias de aprendizaje**	**Recursos materiales**	**Servicios**
• Biblioteca e infoteca.	• El estudiante está totalmente libre.	• Autodidacta.	• Libros. • Múltiples medios en infotecas.	• Acceso a múltiples fuentes de información.
• Centros de recursos para el autoaprendizaje.	• Guiado por el maestro.	• Aprender haciendo.	• Medios para la producción. • Espacios para la difusión. • Acervos de recursos didácticos.	• Un laboratorio de aprendizaje. • Medios para la producción. • Espacios de exhibición.
• Kumon.	• Libertad para elegir tiempos y ritmos de aprendizaje.	• Enseñanza programada y asesorías.	• Fichas de trabajo, para los distintos niveles.	• Complementa y refuerza los conocimientos y habilidades lógico-matemáticas.
• Centros de autoacceso.	• Libertad para elegir material, medios, tiempos y ritmos de aprendizaje.	• Autónoma, con apoyo de un asesor. • Aprender a aprender. • Semiautónoma, combinando el trabajo en el aula con el centro.	• Paquetes instruccionales en múltiples medios. • Rutas de aprendizaje (*pathway*) y hojas de trabajo.	• Complementa y refuerza los conocimientos y habilidades en el aprendizaje de idiomas.

Tabla 1 Evolución de los Centros de Autoaprendizaje

De la Tabla anterior se observan diferencias significativas entre cada una de las etapas por las que han evolucionado los centros de autoacceso.

Así se tiene, que en el caso de las bibliotecas tradicionales son utilizadas para realizar lecturas cuya función primordial es la de posibilitar el acceso a múltiples fuentes de información.

Sin embargo, en la era de la informática no bastan los libros, muchas bibliotecas se han transformado en *Infotecas*, donde además de libros se cuenta con revistas, periódicos, videos, CD-ROM y conexión a *Internet*, con salas de lectura, video aulas y laboratorios de cómputo.

En el caso de los Centros de Aprendizaje Autónomo o Centros de Autoacceso, Pat Grounds especialista del Consejo Británico define al Centro de Autoacceso (es la traducción literal al español de Self-Access Center) como:

"Un lugar donde el usuario tiene la oportunidad de seleccionar entre una gran variedad de materiales (impresos, cassettes, videos, multimedia) que le sirven para incrementar y reforzar sus conocimientos en el aprendizaje de un idioma con la ventaja de invertir o dedicar el tiempo que considere adecuado a sus necesidades, además de establecer el ritmo de avance de su

aprendizaje. Es también un espacio donde el alumno en caso de requerirlo recibe atención de un asesor académico".

Para entender un poco más el concepto de estos centros tenemos a manera de descripción la siguiente información.

La filosofía o política que sigue este tipo de centros esta basada en que el alumno es responsable, autónomo y emprendedor; en el centro de autoacceso es necesario por tanto, apoyo para el alumno no solo en cuanto a material, sino que de verdad tomen lo que este a su alcance, donde sea, y lo utilicen en su beneficio propio. Esto es lo que se llama "aprender a aprender"...

Un centro de Autoacceso esta integrado por diversos elementos que propician el aprendizaje de manera autónoma. Por ello es necesario que se tenga una infraestructura física integrada por espacio, mobiliario, acervo, equipo tecnológico, la oferta de servicios de apoyo pedagógico con los cuales los estudiantes puedan llevar a cabo un proceso de aprendizaje sin necesidad de un profesor. También se brinda asesoría personalizada que está enfocada a la reflexión, establecimiento y logro de objetivos individuales de aprendizaje, materiales didácticos adecuados, medios de información sobre el sistema de autoacceso y formas de auto evaluación del aprendizaje.

La vigencia de la inscripción a estos centros en general tiene un período de duración de seis meses. El personal con el que se cuenta en este tipo de centros va desde los asesores hasta el caso de los responsables o encargados así como de prestadores de servicio social. Requiere de personal de apoyo técnico y administrativo para de ese modo, permitir a los asesores dedicarse al trabajo académico, la preparación de materiales, asesorías, etc.

Los usuarios que tienen derecho a estos centros son generalmente personas de la comunidad de la institución en dónde se localice el Centro de Autoacceso. Esto es, en general, pueden tener acceso a él: estudiantes, académicos y personal administrativo.

En algunos centros se lleva, la contabilización de las asistencias en formatos impresos o en cuadernillos que indican los objetivos, temas de estudio, materiales empleados, áreas visitadas y tiempo de estancia de cada usuario.

La selección de materiales (ya sea para su adquisición, manejo o consulta), es responsabilidad de coordinadores, encargados o directores de los centros, en conjunto con los asesores; en algunos casos se toma en cuenta la opinión de los usuarios y de los profesores de las clases presénciales.

En algunos centros existen catálogos descriptivos de los materiales en las diferentes áreas. La cantidad de materiales existente en cada sección del Centro es suficiente para satisfacer las necesidades de los usuarios.

1.2 MEDIATECA DEL CENTRO DE ENSEÑANZA DE LENGUAS EXTRANJERAS

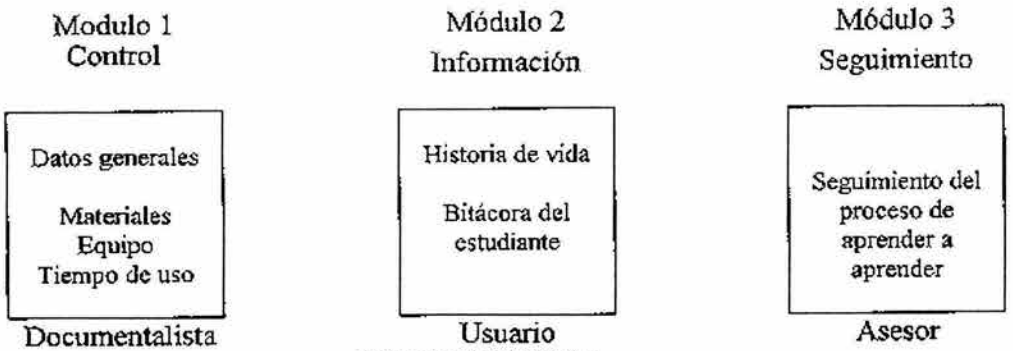
El uso y aprendizaje de lenguas extranjeras, hoy en día se ha convertido en una herramienta que permite, entre otras cosas, el acceso rápido, fácil y directo a diferentes fuentes de información; además permite la comunicación entre los individuos para propósitos diversos: académicos, financieros, de negocios, etc. Por estas razones y la gran demanda que existe en la Universidad Nacional Autónoma de México (UNAM) por parte de la comunidad estudiantil para aprender lenguas extranjeras el Centro de Enseñanza de Lenguas Extranjeras (CELE) de la UNAM ha creado un espacio diferente llamado Mediateca, en el que con apoyo profesional de un equipo interdisciplinario, integrado por lingüistas aplicados, profesores de lenguas extranjeras, pedagogos y bibliotecólogos, entre otros y con un espacio físico y mobiliario adecuado, así como equipo de cómputo, audio y video, se podrán aprender las diferentes lenguas que se enseñan en el CELE, a través del aprendizaje autodirigido.

Esta condición es la que convierte a la Mediateca del CELE en un espacio alternativo a la manera tradicional de aprender lenguas extranjeras.

La Mediateca tiene como objetivo promover y fomentar el aprendizaje autodirigido de las lenguas extranjeras que se imparten en el CELE, mediante la utilización de diferentes recursos, con el propósito de crear en el estudiante una actitud de autonomía en relación con su proceso de aprendizaje.

En la Mediateca del CELE se intenta hacer un modelo de mediateca mexicano o latinoamericano, es decir, un lugar en el que el usuario además de aprender una lengua extranjera o cualquier otra disciplina del conocimiento, sea capaz de aprender a aprender.

La Mediateca cuenta con un sistema, el cual permite conocer y mantener una forma de control y seguimiento de los usuarios en cuanto al uso del equipo de audio, cómputo y video, y de los materiales del acervo documental.



Equipo y material didáctico

Las colecciones están ordenadas bajo un sistema de clasificación local. Esta clasificación tiene la finalidad de ordenar los documentos en las lenguas que potencialmente se enseñarán en la Mediateca y en las áreas que representan su contenido, amplitud y disposición.

1.3 ESCUELA NACIONAL DE ENFERMERÍA Y OBSTETRICIA

En el caso de la Escuela Nacional de Enfermería y Obstetricia observamos que es una escuela deseosa de superación y actualización en cuanto a sus programas de estudio y los servicios que presta a su población estudiantil, por ello es necesario contar con un área de idiomas en la que la comunidad vaya poco a poco adquiriendo la habilidad de un idioma extranjero, en un inicio, el Inglés.

La Escuela Nacional de Enfermería y Obstetricia, desea que sus egresados tengan un perfil académico de alto nivel que satisfaga adecuadamente la demanda de sus servicios profesionales tanto a nivel nacional como internacional.

Es por eso que durante el proceso de formación del alumno, se le impulsa en la búsqueda de información a través de la consulta de material especializado, así como de su participación en diferentes eventos como conferencias, mesas redondas, ponencias, etc. Aunado a ello, en la escuela hay un programa de intercambio estudiantil con algunas Universidades de Canadá y Estados Unidos, lo que exige por parte de los estudiantes interesados un amplio conocimiento del idioma inglés.

Además, los alumnos interesados en realizar estudios en el extranjero a través de una beca, necesitan del conocimiento de al menos una lengua extranjera; aunado a que la literatura más actualizada en el campo de la enfermería, está escrita en lengua inglesa principalmente.

Por tales motivos se consideró necesaria la apertura de un centro que proporcione recursos al alumno para aprender una lengua extranjera y enriquecer su conocimiento. En consecuencia, se solicitó al Centro de Enseñanza de Lenguas Extranjeras de la UNAM, a través de su Mediateca, el diseño de un proyecto que respondiera a los requerimientos de esta escuela.

Este proyecto inició con la aplicación de una encuesta que tiene como finalidad conocer: si el alumno está estudiando un idioma, el nivel de conocimiento de la lengua, cuáles son los medios a través de los cuales puede aprender de forma autodirigida; qué idioma es el más útil para su formación, qué elementos del proceso de comunicación en lengua extranjera se consideran de mayor relevancia, el apoyo académico que esperan encontrar, el tipo de materiales, equipo técnico, el horario de asistencia más conveniente, la disposición y actitud hacia el aprendizaje autodirigido y finalmente se pidió su punto de vista y sugerencias sobre la creación de la Unidad para el Aprendizaje Autónomo-Idiomas de la ENEO.

Para iniciar con el proyecto, se realizó una encuesta a 304 estudiantes de una población total de 970 alumnos, lo que representa un 31%. En la selección de la muestra las autoridades de la escuela mostraron un mayor interés por los alumnos de primer ingreso porque son los que permanecen durante todo el semestre en la escuela y disponen de más tiempo en comparación los demás, los estudiantes de tercer semestre en la muestra constituyen el 25% de la muestra, mientras que los de cuarto semestre representan el 31%, los de sexto semestre el 26%, y finalmente los de séptimo semestre el 5%. Un elemento importante a considerar fue el tiempo de permanencia en la escuela.

Los datos de la muestra proporcional a la que se aplicó la encuesta, se muestran en la siguiente tabla

Semestre	Población T. Matutino De 7:00 hrs a 14:00 hrs	Población T. Vespertino De 14 hrs a 21:00 hrs	Población Total por semestre	Permanencia en la escuela	Número de personas encuestadas
Primer Semestre	157	144	301	551 Alumnos permanecen todo el semestre en la escuela	100
Tercer Semestre	162	88	250		82
Cuarto Semestre	25	20	45	8 semanas	45
Sexto Semestre	84	45	129	6 semanas	62
Séptimo Semestre	23	25	45	3 semanas	15
Octavo Semestre	120	77	197	3 semanas	N. E.E.
Total de alumnos	571	399	970	2 turnos 970 alumnos	304 Encuestados

Fuente: Secretaría General de la Escuela Nacional de Enfermería y Obstetricia. UNAM

Al finalizar el estudio mediante la aplicación de encuestas, se llegaron a las siguientes conclusiones, sobre la necesidad de la creación de un centro de Aprendizaje para el área de Idiomas.

- Con la creación del centro de autoacceso se buscará canalizar de la mejor manera el alto porcentaje de alumnos que manifiesta la importancia de aprender una lengua extranjera y de su gran disposición hacia el aprendizaje autónomo, así como, facilitar los recursos académicos que favorecen al estudiante en la adquisición de conocimiento, destacando la utilidad de hacer consciente este proceso en diferentes ámbitos no sólo en el escolar.
- El idioma con el que se iniciará será el inglés en respuesta a la demanda observada en la encuesta, con asesorías, talleres de lectura, clubes de conversación, etc; subrayando la importancia de la atención personalizada que los estudiantes desean encontrar.
- Los materiales que se pondrán a disposición de los usuarios serán audiovisuales e impresos, aunque se busca un equilibrio, con respecto al número de materiales y el nivel, existe un interés especial en los métodos comerciales de nivel básico por las características de la población.
- Los servicios adicionales así como la implementación de otra(s) lengua(s) se estimará de acuerdo a las necesidades futuras del centro.
- Se propone iniciar con un horario de las 10:00 las 19:00 hrs.

CAPÍTULO 2
TEORÍA DE DESARROLLO DE SOFTWARE

2.1 CONCEPTOS DE DESARROLLO DE SOFTWARE

Complejidad del Software

Cuanto más grandes son los sistemas de software mayor es su complejidad, por ello es necesario conocer la procedencia de esa complejidad. Se puede hablar de dos aspectos que causan esta complejidad, uno estático y otro dinámico.

El aspecto estático del software tiene que ver con la funcionalidad que el software ofrece. Cuanto mayor es su funcionalidad mayor es el número de requisitos que debe satisfacer un sistema. Esto significa que los sistemas se vuelven más grandes y más difíciles de comprender por la cantidad de información y funciones que manejan. El nivel de complejidad radica en estos aspectos intrínsicos a la aplicación. Para reducir tal complejidad habría que simplificar la funcionalidad que el sistema ofrece. Obviamente, la complejidad puede fácilmente aumentar si la aplicación no está desarrollada de manera adecuada.

El aspecto dinámico del software tiene que ver con los cambios que pudieran hacerse en un sistema en el futuro. Según una "ley" de desarrollo de software (Lehman, 1985), "todo programa que se use se modificará"; y cuando un programa se modifica, su complejidad aumenta. Sin embargo cuando un sistema es modificado y este ha sido de alguna manera desordenado, puede llegar un punto en el que el desorden logrado es tal que el sistema resultante no es económicamente justificable para continuar con él, ya que es demasiado caro modificarlo. Lamentablemente, la historia muestra que los sistemas raramente se desarrollan a tiempo, dentro del presupuesto y según las especificaciones originales. Más aún, los sistemas tienden a fallar. Sin embargo, no todo es negativo. Un aspecto importante para poder manejar la complejidad de los sistemas, es seguir un buen proceso de desarrollo de software.

Ciclo de Vida del Software

Para comprender este concepto es necesario entender un poco más en qué consiste desarrollar software. Un sistema de software tiene un ciclo de vida que comienza con la formulación de un problema, seguido por la especificación de requisitos, análisis, diseño, implementación, verificación, validación, integración y pruebas del software, continuado de una fase operacional durante la cual se mantiene y extiende el sistema. Todo desarrollo de software incluye aspectos esenciales, como la creación de las estructuras que resuelvan el problema, junto con aspectos secundarios ("accidentales"), como la codificación y las pruebas.

Según Brooks, existe una regla empírica ("thumb rule") que dice que para el desarrollo de un proyecto de software se debe asignar, 1/3 del tiempo a la planeación, 1/6 a codificación, 1/4 a pruebas de componentes, y 1/4 a pruebas del sistema, como se muestra en la Figura 1.

Esto implica que, la mitad del esfuerzo (2/4) son dedicados a pruebas lo cual también incluye la depuración y aspectos secundarios del software.

Fig. 1 Estimado general del tiempo dedicado al desarrollo de un proyecto de Software

2.2 TECNOLOGÍA ORIENTADA A OBJETOS

Dado que un aspecto primordial de este trabajo es el software orientado a objetos, es entonces necesario comprender qué significa esta tecnología. Como primer punto se hablará de los mitos y las realidades de esta tecnología; en segundo término los aspectos básicos que distinguen a la programación orientada a objetos con respecto a la manera tradicional de programación; finalmente, la motivación, y conceptos detrás de esta tecnología junto con una breve reseña de los más importantes lenguajes orientados a objetos.

Mitos y Realidades

La orientación a objetos es un buen ejemplo de cómo un "pequeño detalle" puede significar tanto.

Programación Tradicional

En la programación tradicional, conocida como *estructurada*, el objetivo es separar los datos del programa de las funciones que los manipulan. El programa o aplicación completa, consiste de múltiples datos y múltiples funciones, como se muestra en la Figura 2.

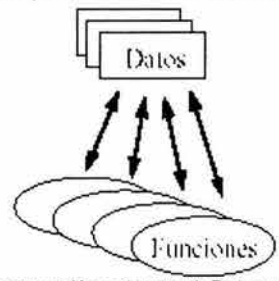


Fig. 2 Programación estructural: Datos y Funciones Globales

Esta forma de programar tiene sus orígenes en la arquitectura Von Newmann de las primeras computadoras modernas. La arquitectura básica es la misma utilizada en la actualidad a nivel comercial en las PC's y se basa de manera simplificada en una unidad central de procesamiento (CPU) y una memoria donde se carga el programa o aplicación que debe ejecutarse. (El disco duro guarda a largo plazo la aplicación para que ésta no se pierda pero no juega un papel primordial cuando la aplicación se ejecuta.) La memoria en sí se divide en una sección donde se guardan las funciones del programa, correspondiente al código que controla la lógica de la aplicación, y otra sección de datos donde se guarda la información que quiere manipularse. Dada esta separación entre funciones y datos en la memoria lo más lógico siempre ha sido utilizar una programación que se ajustara a ello dando origen a un gran número de lenguajes basados en esta estructuración.

Esta forma de programar presenta dos problemas principales:

- El primer problema es obligar a un programador a pensar como la máquina, en lugar de lo opuesto.
- El segundo problema es que toda la información presente es conocida y potencialmente utilizada por todas las funciones del programa y si se hiciera algún cambio en la estructura de alguno de los datos (se consideran todos como "globales"), potencialmente habría que modificar todas las funciones del programa para que éstas pudieran utilizar la nueva estructura.

Programación Orientada a Objetos

Para la solución de los dos problemas en el caso de la programación estructurada, la Programación Orientada a Objetos nos ayuda debido a que la unidad básica de programación es el *objeto*. Para el primer problema se observa que a nivel organizacional el concepto del objeto nos acerca más a la manera de pensar de la gente al agregar un nivel de abstracción adicional donde internamente la estructura del programa se ajusta a la arquitectura de la máquina. En cuanto al segundo problema, los datos globales desaparecen, asignando a cada objeto sus propios datos y

funciones locales, resultando en un programa o aplicación definido exclusivamente en término de objetos y sus relaciones entre sí, como se muestra en la Figura3.

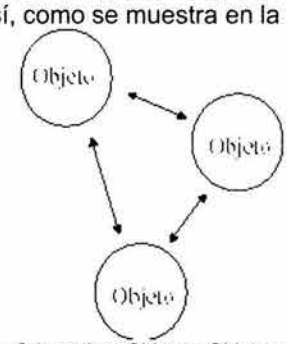


Fig. 3 Programación Orientada a Objetos: Objetos Globales

Obviamente se deben tener datos y funciones para que un programa tenga sentido, pero estos son guardados en cada objeto de manera independiente, como se muestra en la Figura 4.

Fig. 4 Programación Orientada a Objetos: Objetos globales que contienen Datos y funciones Locales

Se observa que los datos están ubicados en el centro del objeto (un concepto puramente ilustrativo) resaltando el efecto de que un cambio en la estructura de uno de estos datos sólo afecta a las funciones del mismo objeto pero no al resto de la aplicación.

2.2.1 Programación Orientada a Objetos

El software orientado a objetos apoya ciertos aspectos que mejoran la robustez de los sistemas, este software requiere de ciertas características mínimas para considerarse orientado a objetos y finalmente debe integrarse como parte de un lenguaje de programación.

Aspectos que Mejoran la Robustez de los Sistemas

Existen razones un poco más técnicas que motivan a la orientación a objetos, como son la *abstracción*, *modularidad*, *extensiblidad* y *reutilización*.

- **Abstracción.** Consiste en reducir el nivel de datos primitivo necesarias para producir un sistema de software. De manera sencilla esto se logra mediante el uso de lenguajes de programación que contengan estructuras de datos de alto nivel. Con la programación orientada a objetos se definen dos niveles de abstracción. El nivel más alto, el de los objetos, es utilizado para describir la aplicación mientras que el nivel más bajo, el de los datos y las funciones, es utilizado para describir sus detalles. Este nivel inferior corresponde al único nivel de la programación tradicional. Esto refleja que la complejidad se maneja de mejor manera con la tecnología orientada a objetos. El caso del objeto como estructura básica sirve para separar el "qué" de una aplicación del "cómo", o sea sus detalles, al contrario de la programación tradicional donde el "qué" y el "cómo" se resuelven a la vez.

- **Modularidad.** Depende de las abstracciones básicas del sistema, lo cual permite dividir el sistema en componentes separados. Al tener abstracciones de mayor nivel la modularidad de los componentes también es de mayor nivel reduciendo el número final de componentes lo cual a su vez simplifica su manipulación y mantenimiento. Con la programación Orientada a Objetos, la modularidad del sistema se da en base a objetos, un nivel más alto que los datos y funciones tradicionales. El número final de módulos, es decir de objetos, es menor que el número original de datos y funciones. Esto reduce la complejidad de la aplicación ya que el analista piensa en menos componentes a la vez descartando detalles innecesarios.
- **Extensibilidad.** Tiene como objetivo permitir cambios en el sistema de manera modular afectando lo mínimo posible el resto del sistema. Con la programación Orientada a Objetos, los cambios se dan a dos niveles: modificación externa e interna de los objetos. Los cambios internos a los objetos afectan principalmente al propio objeto, mientras que los cambios externos a los objetos afectarán de mayor forma al resto del sistema. Dada la reducción en el número de entidades básicas en un sistema mediante abstracciones de nivel más alto, se logra un desarrollo de sistemas más estables con menor complejidad, y por lo tanto más fácilmente extensibles.
- **Reutilización.** El reuso de código reduce el tiempo del diseño, la codificación, y el costo del sistema al amortizar el esfuerzo sobre varios diseños. Mediante el reuso de código se puede aprovechar componentes genéricos para estructurar bibliotecas reutilizables, y así lograr una estandarización y simplificación de aplicaciones por medio de componentes genéricos prefabricados. Y aunque el reuso es posible en lenguajes convencionales, los lenguajes orientados a objetos aumentan substancialmente las posibilidades de tal reuso, gracias a la modularidad de los sistemas. En particular, lenguajes como Java ofrecen componentes de estructuras de datos básicas como colas, pilas, listas, árboles, junto con aquellas de más alto nivel, utilizadas por ejemplo para la construcción de interfaces de usuario facilitando el desarrollo de nuevas aplicaciones. La problemática mayor de la reutilización radica en que para construir componentes genéricos, sencillos, con interfaces bien definidas y que puedan utilizarse en varias áreas de aplicación el esfuerzo es mucho mayor que para construir componentes que serán utilizados en una aplicación. Con la orientación a objetos, el *objeto* es la unidad de reuso más pequeña, pudiéndose aprovechar definiciones similares de objetos dentro de la misma aplicación o incluso en distintas aplicaciones. Al agrupar objetos similares se puede lograr reutilización de componentes de más alto nivel. Por otro lado, se puede aprovechar objetos con estructuras de datos y funciones similares, definiendo una sola vez los aspectos comunes y *especializándolos* en objetos adicionales.

2.2.1.1 Características Mínimas de los Lenguajes Orientados a Objetos

La motivación detrás de la orientación a objetos es lograr mayor productividad en el desarrollo de software y mejorar la calidad de éste mediante niveles más altos de abstracción, apoyo a la modularidad, extensiblidad y reutilización de código.

Por otro lado hay conceptos básicos que hacen que un lenguaje sea considerado efectivamente orientado a objetos: *encapsulamiento*, *clasificación*, *generalización* y *polimorfismo*.

- **Encapsulación.** *Encapsulación* o *encapsulamiento* es la separación de las propiedades externas de un objeto, o sea su interface, correspondiente a la interface de sus funciones, de los detalles de implementación internos del objeto, o sea sus datos y la implementación de sus funciones, como se muestra en la Figura 5.

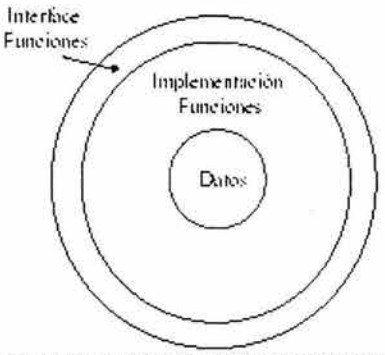


Fig. 5 Un Objeto da a conocer a los demás objetos sólo las interfaces de sus funciones

Esta separación es muy importante, realmente el conocimiento de un objeto por otros objetos en la misma aplicación es exclusivamente en base a la interface de dichos objetos. Todo el detalle, al estar encapsulado, es desconocido por el resto de la aplicación, limitando el impacto de cualquier cambio en la implementación del objeto, ya que los cambios a las propiedades internas del objeto no afectan su interacción externa. Obviamente cualquier cambio en la propia interface del objeto afectaría potencialmente a todo el resto de la aplicación. Sin embargo el porcentaje de código dedicado a las interfaces es por lo general "muchísimo" menor que el porcentaje total de líneas de código utilizados para datos e implementación de funciones. De tal manera se reduce la complejidad del sistema protegiendo los objetos contra posibles errores, y permitiendo lograr de mejor manera extensiones futuras en la implementación de los objetos.

- **Clasificación.** En toda programación orientada a objetos la *clasificación* es un aspecto fundamental, donde objetos que contienen estructuras similares, correspondiente a tipos de datos y funciones similares, se clasifican como pertenecientes a la misma *clase* de objeto. Es importante resaltar que se trata de *tipos* de datos similares, dado que los valores de los datos aún pueden cambiar en objetos de clase similar.
- **Generalización.** Si tomamos como base la clasificación, y consideramos que no sólo los objetos similares pueden clasificarse, sino también las propias clases de los objetos, entonces se define la *generalización* o *especialización* de clases. Mediante la generalización, clases de objetos con estructura y comportamiento similar se reutilizan en la definición de las nuevas clases. Estas nuevas clases se consideran clases más especializadas o *subclases* mientras que las originales se consideran clases más generales o *superclases*. El mecanismo para describir jerarquías de generalización de clases se conoce como *herencia*, un término muy utilizado en la orientación a objetos, se dice que una subclase hereda de una superclase. La herencia puede ser *sencilla*, donde una subclase hereda de una sola superclase directa, o *múltiple*, donde una subclase hereda de múltiples superclases directas. La herencia es también una forma de reutilización de código, ya que se aprovechan descripciones de clases de objetos para luego definir clases de objetos parecidos.
- **Polimorfismo.** Quizás el concepto más complicado de explicar y en cierta manera el más poderoso es el *polimorfismo*. De manera simplificada, mediante el polimorfismo se definen funciones con el mismo nombre e interfaz en distintas clases de objetos, pero bajo implementaciones distintas. El polimorfismo es útil para extender la funcionalidad existente en los objetos del sistema, a nuevos objetos aún desconocidos en ese momento. Es como definir un estándar de interfaces para los objetos la cual debe ser seguida por todos los existentes y nuevos.

2.2.2 Lenguajes de Programación

Los lenguajes orientados a objetos varían en su apoyo a los conceptos de orientación a objetos y en los ambientes de desarrollo que incorporan. Por lo general, cada lenguaje, aunque orientado a objetos, tiene un diseño particular teniendo aspectos comunes entre sí. El analista debe considerar los distintos aspectos y tomar una decisión de cuál es el lenguaje más apropiado para su aplicación. En el caso del presente trabajo el lenguaje que se maneja es Java por dos motivos principales: sus buenas características como lenguaje de programación y su gran aceptación en el mercado que lo hacen uno de los más utilizados en la actualidad.

A continuación se enlistan los lenguajes de Programación Orientada a Objetos que han estado presentes a lo largo de la historia de los lenguajes de Programación, considerando que su existencia data de hace varias décadas, como se muestra en la Tabla 1, en orden cronológico. Se observa que a partir de la década de los 80 la gran mayoría son orientados a objetos.

Año	Lenguaje	Descripción	¿OO?
1962	SIMULA	El primer sistema con objetos fue B1000 en 1961, seguido por Sketchpad en 1962, conteniendo "clones" e instancias. Sin embargo, se le atribuye como el primer lenguaje orientado a objetos conteniendo objetos y clases, a Simula I, diseñado por Ole Dahl y Kristen Nygaard del Centro de Computación de Noruega (NCC Oslo) en 1962. El lenguaje, implementado por primera vez en 1964, fue diseñado como una extensión a Algol 60 para la simulación de eventos discretos. En 1967, se introdujo el lenguaje de propósito más general Simula67 con un número mayor de tipos de datos además de apoyo a objetos. Simula se estandarizó en 1977.	Sí
1972	Smalltalk	Smalltalk diseñado por Alan Kay (quien había diseñado y construido entre 1967 y 1968 la primera computadora personal basada en programación orientada a objetos, llamada FLEX) y otros en XEROX PARC. La primera versión fue conocida como Smalltalk 72, cuyas raíces fueron Simula 67. Siguió Smalltalk 76, una versión totalmente orientada a objetos. En 1980, Smalltalk 80, fue la primera versión comercial del lenguaje, incluyendo un ambiente de programación orientado a objetos uniforme. El lenguaje Smalltalk ha influido sobre otros lenguajes como C++ y Java, y aunque no ha tenido el grado de éxito de estos dos últimos, quizás por lo tardío en volverse gratis junto a razones de eficiencia, este lenguaje tiene un gran número de seguidores en la actualidad, los cuales consideran a Smalltalk como el "mejor" lenguaje que existe.	Sí
1980	Modula	La versión original se conoció como Modula-2 desarrollada por Niklaus Wirth diseñada a mediado de los 70s como descendiente director de Pascal. El lenguaje incluía concurrencia y ciertos aspectos de la orientación a objetos. La última versión conocida como Modula-3 fue diseñada por Luca Cardelli. Dada su simpleza se desconoce por qué la falta de éxito en la utilización de este lenguaje.	Sí
1983	Ada	El lenguaje fue diseñado a partir de 1977 por el Departamento de Defensa de Estados Unidos, teniendo como autor principal a Jean Ichibah, para apoyar programación de gran escala y promover la robustez del software. Su nombre fue en honor de Lady Ada Lovelace (1815-1852), una amiga y confidente de Charles Babbage, considerado el padre de la computación por su trabajo teórico hace un siglo y medio. Aunque la versión original no era orientada a objetos, la versión actual Ada 1995 sí lo es. Existe otra versión no orientada a objetos conocida como Ada 83 o Ada Clásica.	Sí
1983	Objective-C	El lenguaje fue diseñado por Brad Cox como una extensión a C pero con orientación a objetos. El lenguaje ofrecía muchos aspectos de diseño de Smalltalk-80 como su misma naturaleza sin tipos, aunque incorporaba datos sencillos de C, como enteros y reales. Su popularidad inicial vino a raíz de su utilización en la computadora NeXT, incluyendo una interfaz de construcción como parte del ambiente NeXTSTEP, conocido luego como OpenStep, que más tarde fue adquirido por Apple como base para su nuevo sistema operativo MacOS X.	Sí
1983	Beta	Beta, desarrollado por Madsen en la Universidad de Aarhus en Dinamarca es otro lenguaje orientado a objetos inspirado por Simula con sintaxis similar a Pascal y algo parecido a C.	Sí

1985	C++	C++ diseñado por Bjarne Stoustrup, AT&T Bell Labs, entre 1982 y 1985 es uno de los lenguajes de programación más populares actualmente. El lenguaje se agrega aspectos de orientación a objetos al lenguaje de C, siendo realmente un lenguaje híbrido donde un programador puede efectivamente programar en C aunque utilizando C++. En la actualidad muchos de los seguidores de este lenguaje se han pasado a Java. La razón primordial de esto es la complejidad de C++ junto con muchos aspectos problemáticos y falta de estandarización bajo distintas plataformas.	Sí
1986	Eiffel	Eiffel, honrando a la famosa torre en París, fue diseñado por Bertrand Meyer como un lenguaje orientado a objetos con una sintaxis superficialmente similar a C. Eiffel ofrece un ambiente interpretado de "bytecode" similar a Java, aunque por eficiencia este código normalmente se traduce a C para luego ser compilado en el ambiente particular. El diseño del lenguaje apoya un enfoque de ingeniería de software conocido como "Diseño por Contrato". Aunque es un lenguaje muy sencillo y poderoso nunca logró la aceptación lograda por C++ y Java, posiblemente por la falta de compiladores gratis.	Sí
1986	Self	Self diseñado por David Ungar y Randall Smith es un lenguaje cuya sintaxis es similar a Smalltalk. Un aspecto muy novedoso es la omisión de la noción de clase en el lenguaje, donde un objeto se deriva de otro (su prototipo) por medio de copiado y refinado. Dado esto, Self es un lenguaje muy poderoso, sin embargo, es aún un proyecto de investigación requiriendo una gran cantidad de memoria y una gran máquina para ejecutar.	Sí
1988	CLOS	CLOS ("Common LISP Object System") es una extensión de CommonLisp mediante la orientación a objetos desarrollada originalmente en 1988 por David Moon (Sumbolics), Daniel Bobrow (Xerox), Gregor Kiczales (Xerox) y Richard Gabriel (Lucid), entre otros.	Sí
1990	Haskell	Haskell desarrollado por un comité (Hughes, Wadler, Peterson y otros) tiene su nombre en honor a Haskell Brooks Curry, cuyo trabajo en lógica matemática sirve como fundamento para los lenguajes funcionales. El lenguaje está altamente influenciado por Lisp aunque fue extendido con ciertos aspectos de la orientación a objetos para ser más moderno.	Sí
1992	Dylan	Dylan ('DYnamic LANguage') es un lenguaje orientado a objetos originalmente desarrollado por Apple, se parece mucho a CLOS y Scheme, aunque ha sido influenciado por Smalltalk y Self.	Sí
1995	Java	Java, diseñado por Gosling en Sun Microsystems entre 1994 y 1995 es el lenguaje orientado a objetos más utilizado en la actualidad. El lenguaje es sencillo y portátil, bastante similar a C++, aunque tomando ideas de Modula-3, Smalltalk y Objective-C, haciéndolo más robusto y seguro. Java es típicamente compilado en "bytecodes" que son luego interpretados por una máquina virtual de Java (JVM). Un aspecto primordial en el éxito del lenguaje es su integración con el Web mediante aplicaciones conocidas como "applets" que pueden ser ejecutadas desde un navegador del Web ("browser"). Otro aspecto importante es la inclusión de un gran número de paquetes y librerías que estandarizan y facilitan el desarrollo de nuevos programas.	Sí
2000	C#	Este lenguaje conocido como "C Sharp" es el último intento por parte de Microsoft de competir contra el éxito y el seguimiento que tiene Java. El lenguaje revisa muchos aspectos problemáticos de C++.	Sí

Tabla 1 Describe los lenguajes más importantes de la historia de la computación haciendo énfasis en aquellos orientados a objetos.

2.3 EL PROCESO PARA EL DESARROLLO DE SOFTWARE

Un *proceso* está definido como una serie de acciones u operaciones que conducen a un fin. En el caso del desarrollo de software, se requieren procesos que abarquen desde la creación de un sistema de software hasta su mantenimiento. Todo esto es conocido como el *ciclo de vida* del software. El desarrollo de sistemas de software es algo muy complejo. Un aspecto básico para manejar la complejidad inherente en los sistemas de software es contar con un modelo de proceso a seguir.

Modelo del Proceso

El *modelo de proceso* define un orden para llevar a cabo los distintos aspectos del proceso. El modelo se puede definir como un grupo de estrategias, actividades, métodos y tareas, que se organizan para lograr un conjunto de metas y objetivos.

En general, durante el modelo de proceso se definen las *estrategias*, *actividades* y *métodos*. Estos conceptos se describen a continuación.

- *Estrategia* es un plan para llevar a cabo un objetivo, en nuestro caso el desarrollo de software. Existen diversas estrategias para lograr mejor calidad en el software final. Una estrategia básica se relaciona con el tipo de arquitectura que se desea crear. Las estrategias básicas escogidas afectan directamente el tipo de programación y los lenguajes que se utilizarán. De cierta manera en este caso en particular la estrategia consiste en el uso de tecnología orientada a objetos.
- *Actividad* define los distintos pasos necesarios para lograr las metas y objetivos definidos en el modelo de proceso, o sea en el desarrollo de software. Las actividades dependen de la arquitectura de software y deben ser suficientemente poderosas para expresar la información requerida para modelar el sistema. Las actividades básicas necesarias para el proceso de desarrollo de software son las siguientes:
  - *Requisitos* para capturar los aspectos funcionales correspondientes, cómo un usuario interactuaría con el sistema;
  - *Análisis* para dar al sistema una estructura robusta y extensible bajo un ambiente de implementación ideal;
  - *Diseño* para adoptar y refinar las estructuras al ambiente de implementación particular;
  - *Implementación* para codificar el sistema;
  - *Pruebas* para validar y verificar el sistema;
  - *Integración* para pegar componentes del sistema;
  - *documentación* para describir los distintos aspectos el sistema y
  - *Mantenimiento* para extender la funcionalidad del sistema.
- *Método* es un procedimiento definiendo las tareas que deben llevarse a cabo para satisfacer la actividad. Existen métodos para asegurar la calidad del software, seguir el progreso del proyecto y probar el software. Una *metodología* se refiere al estudio de los métodos, existiendo un gran número de metodologías para el desarrollo de software.

Existen procesos de acuerdo al tipo de proyecto y aunque no hay límite a los diversos modelos de proceso que puedan existir, los más clásicos son: el Modelo de Cascada, el Modelo de Espiral.

2.3.1 MODELOS DESARROLLO DE SOFTWARE

Modelo Cascada

El modelo de cascada se define como una secuencia de actividades a ser seguidas en orden, donde la estrategia principal es definir y seguir el progreso del desarrollo de software hacia puntos de revisión bien definidos. La Figura 6 muestra un diagrama conceptual del modelo describiendo el orden a seguir de las actividades del desarrollo de software. No se muestra una etapa explícita de "documentación" dado que ésta se llevaba a cabo durante el transcurso de todo el desarrollo.

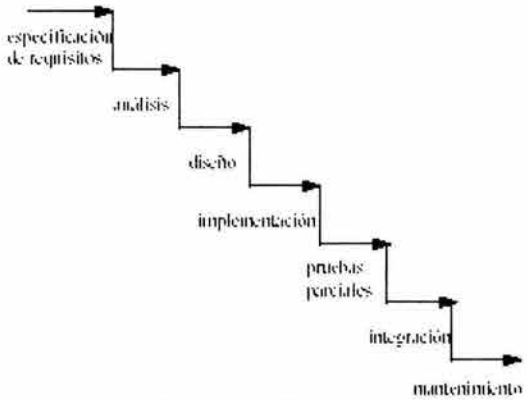


Fig. 6 Diagrama del Modelo de Cascada

Ed Yourdon, en su libro *Decline and Fall of the American Programmer*[1], discute los problemas con el Modelo de Cascada:

- Los documentos a entregar rigen el proceso de software.
- Toma demasiado tiempo ver resultados.
- Depende de requisitos estables correctos.
- Hace difícil rastrear (ver la dependencia) de los requisitos iniciales y el código final.
- Retrasa la detección de errores hasta el final.
- No promueve el reuso de software.
- No promueve el uso de prototipos.
- No se practica de manera formal.

Modelo Espiral

El modelo de espiral se basa en una estrategia para reducir riesgo, al contrario del modelo de cascada que es dirigido por documentos. El modelo enfatiza ciclos de trabajo, cada uno de los cuales estudia el riesgo antes de proceder al siguiente ciclo. Cada ciclo comienza con la identificación de los objetivos para una parte del producto, formas alternativas de lograr los objetivos, restricciones asociadas con cada alternativa, y finalmente procediendo a una evaluación de las alternativas. Cuando se identifica incertidumbre, se utilizan diversas técnicas para reducir el riesgo en escoger entre las diferentes alternativas. Cada ciclo del modelo de espiral termina con una revisión que discute los logros actuales y los planes para el siguiente ciclo, con el propósito de lograr la incorporación de todos los miembros del grupo para su continuación. La revisión puede determinar si desarrollos posteriores no van a satisfacer las metas definidas y los objetivos del proyecto. En tal caso, se terminaría el espiral.

La Figura 7 muestra un diagrama conceptual del modelo de cascada describiendo los distintos ciclos del espiral. Nuevamente, no se muestra una etapa explícita de "documentación" dado que ésta se llevaba a cabo durante el transcurso de todo el desarrollo.

[1] Yourdon, 1992

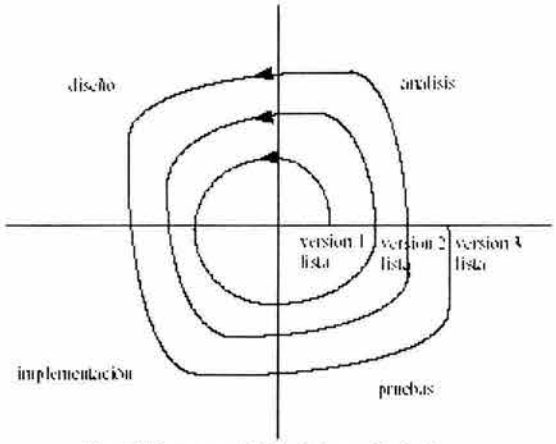


Fig. 7 Diagrama del Modelo en Espiral

Con algunas variantes, este es el modelo de proceso más importante en la actualidad.

2.4 ACTIVIDADES DEL CICLO DE VIDA DEL SOFTWARE

La Tabla 2 muestra las actividades más importantes para el ciclo de vida del desarrollo de software. Estas actividades corresponden a las mostradas en la Figura en el Modelo de Cascada y corresponden de manera similar a las actividades del Modelo de Espiral. La diferencia entre ellas radica en el proceso y orden para llevarlas a cabo junto con las estrategias y métodos utilizados para cada actividad.

Actividad	Descripción
Requisitos	Se especifica las necesidades del sistema a desarrollarse. La especificación de requisitos puede servir como base para la negociación entre los desarrolladores y clientes del sistema y también para planear y controlar el proceso de desarrollo.
Análisis	Se busca comprender los requisitos del sistema logrando la estructuración de una solución, correspondiente a la arquitectura general. Se contesta la pregunta del "qué" del sistema.
Diseño	Se transforma la arquitectura general de análisis, a una arquitectura particular y detallada del sistema que satisfaga todos los requisitos del sistema, donde las condiciones idealizadas durante el análisis se reemplazan por requisitos del ambiente de implantación particular. Se contesta la pregunta del "cómo" del sistema.
Implementación	Se expresa la arquitectura particular del sistema, en una forma aceptable para la computadora, o sea el código.
Pruebas	Se verifica y valida el sistema a nivel de componentes y la integración de ellos. Este es uno de los aspectos más críticos del desarrollo y debe ser aplicado desde el inicio, durante todas las actividades. De tal manera se busca descubrir cualquier defecto en los requisitos, análisis, diseño, implementación e integración. Las pruebas se hacen a varios niveles, desde funciones sencillas hasta el sistema completo.
Integración	Se combinan todos los componentes creados de manera independiente para formar el sistema completo.
Documentación	Se describen los aspectos sobresalientes de los requisitos, análisis, diseño, implementación, integración y pruebas. Esto servirá para usuarios externos e internos, aquellos encargados en mantener el sistema y extenderlo.
Mantenimiento	Se corrigen errores no encontrados durante el desarrollo y pruebas originales del sistema. Se extiende el sistema según existan nuevas necesidades.

Tabla 3 Actividades del Desarrollo de Software

La transición entre las distintas actividades debe ser natural, debiendo existir una continuidad o rastreabilidad de una actividad a la siguiente o la anterior.

2.5 MÉTODOS Y METODOLOGÍAS

Los métodos definen las reglas para las distintas transformaciones dentro de las actividades. Las metodologías definen el conjunto de métodos. La selección de las metodologías a utilizarse es otra de las decisiones críticas en el proceso de software.

Metodologías Estructuradas

La metodología estructurada se basa primordialmente en la división entre funciones y datos. Extendiendo este concepto básico, la metodología estructurada identifica durante el análisis las funciones del sistema, mientras que durante el diseño identifica los datos. Durante las fases de requisitos y análisis se utilizan las siguientes herramientas para describir el sistema lógico: *diagramas de flujo de datos, especificación de procesos, diccionario de datos, diagramas de transición de estados, y diagramas de entidad-relación.*

Metodologías Orientadas a Objetos

Los sistemas orientados a objetos se desarrollan alrededor de entidades del dominio del problema, lo cual resulta en desarrollos bastante estables. El análisis orientado a objeto, a diferencia del estructurado, que considera comportamiento y datos de forma separada, combina ambos. Las características principales de los orientados a objetos son que ofrecen una forma de pensar más que una forma de programar. Reducen la complejidad en el diseño de software, y permiten atacar los errores durante el diseño en lugar de durante la implementación, donde el costo de reparación es bastante mayor. Las actividades son:

- encontrar los objetos (dominio de la aplicación y problema particular);
- organizar los objetos (clases, asociaciones);
- describir cómo los objetos interactúan (escenarios, casos de uso);
- definir las operaciones de los objetos (interfaces);
- definir los objetos internamente (atributos).

Se han propuesto diversas técnicas para identificar objetos en el dominio del problema:

- *Escenarios:* Capturar comportamientos del sistema en guiones o casos de usos para derivar los roles y responsabilidades del sistema.
- *Tarjetas CRC:* Lluvia de ideas entre un grupo de expertos dando como resultado los objetos del dominio del problema que son anotadas en las tarjetas.
- *Ingeniería de información:* Identificar la estructura de la información que se guarda y mantiene por las aplicaciones, y mapearla a objetos.
- *Resaltar Texto:* Una técnica muy utilizada es subrayar nombres y verbos en la especificación de requisitos.
- *Diagramas de clases* son los más importantes describiendo los componentes básicos para las arquitecturas del análisis y diseño. A diferencia de los diagramas de flujo, que no son utilizados por la mayoría de las metodologías orientadas a objetos, los diagramas de clases muestran relación de asociación entre objetos y no flujo de datos entre ellos.
- *Diagramas de casos de uso* son los que se utilizarán, se basan en Objectory correspondientes a la especificación de requisitos. Son completados por documentos en forma de textos.
- *Diagramas de secuencia,* también conocidos como diagramas de eventos, muestran aspectos dinámicos de los objetos en relación a la comunicación con otros objetos en el tiempo.

CAPÍTULO 3
MODELO DE REQUISITOS

3.1 MODELO DE REQUISITOS

El modelo de requisitos tiene como objetivo delimitar el sistema y capturar la funcionalidad que debe ofrecer desde la perspectiva del usuario. Este modelo puede funcionar como un contrato entre el desarrollador y el cliente o usuario del sistema, y por lo tanto proyecta lo que el cliente desea según la percepción del desarrollador.

El modelo de requisitos es el primer modelo a desarrollarse, sirviendo de base para la formación de todos los demás modelos en el desarrollo de software. En general, el cualquier cambio en la funcionalidad del sistema es más fácil de hacer, y con menores consecuencias, a este nivel que posteriormente. El modelo de requisitos que se desarrollará se basa en la metodología *Objectory*[1], cuyo fundamento es el modelo de *Casos de Uso*.

El modelo de casos de uso y el propio modelo de requisitos son la base para los demás modelos y se resume así:

- Requisitos: El modelo de casos de uso sirve para expresar el modelo de requisitos, el cual se desarrolla en cooperación con otros modelos.
- Análisis: La funcionalidad especificada por el modelo de casos de uso se estructura en el modelo de análisis, que es estable con respecto a cambios, siendo un modelo lógico independiente del ambiente de implementación.
- Diseño: La funcionalidad de los casos de uso ya estructurada por el análisis es realizada por el modelo de diseño, adaptándose al ambiente de implementación real y refinándose aún más.
- Implementación: Los casos de uso son implementados mediante el código fuente en el modelo de implementación.
- Pruebas: Los casos de uso son probados a través de las pruebas de componentes y pruebas de integración.
- Documentación: El modelo de casos de uso debe ser documentado a lo largo de las diversas actividades, dando lugar a distintos documentos como los manuales de usuario, manuales de administración, etc.

El diagrama de la Figura 1 ilustra los distintos modelos. Se describirán los detalles y la notación más adelante.

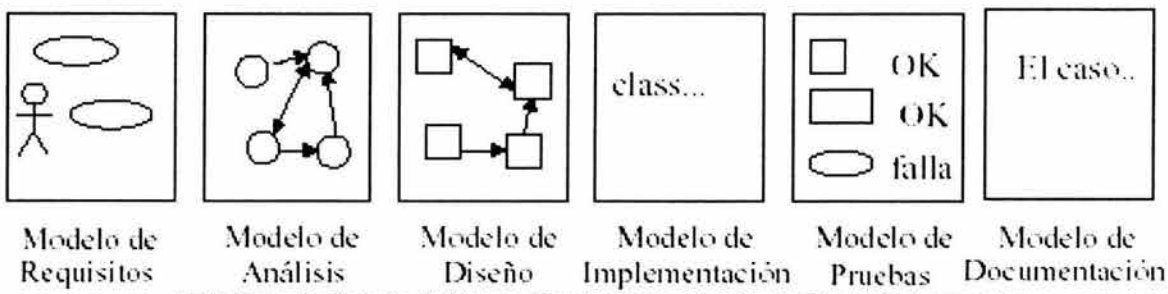


Fig 1: Dependencia de los distintos modelos del proceso de software del modelo de casos de uso

El propósito del modelo de requisitos es comprender completamente el problema y sus implicaciones. Todos los modelos no solamente se verifican contra el modelo de requisitos, sino que también se desarrollan directamente de él. El modelo de requisitos sirve también como base para el desarrollo de las instrucciones operacionales y los manuales ya que todo lo que el sistema deba hacer se describe aquí desde la perspectiva del usuario. El modelo de requisitos no es un proceso mecánico, se debe interactuar constantemente con el usuario final para completar la información faltante, y así clarificar ambigüedades e inconsistencias. Se debe separar entre los requisitos verdaderos y las decisiones

[1] Jacobson, I., Booch, G., Rumbaugh, J., 1998, *The Unified Software Development Process*, 1era Ed., Addison-Wesley

relacionadas con el diseño e implementación. Se debe indicar cuales aspectos son obligatorios y cuales son opcionales para evitar restringir la flexibilidad de la implementación. Durante el diseño se debe extender el modelo de requisitos con especificaciones de rendimiento y protocolos de interacción con sistemas externos, al igual que provisiones sobre modularidad y futuras extensiones. En ciertas ocasiones ya se puede incluir aspectos de diseño, como el uso de lenguajes de programación particulares.

En la metodología de *Objectory*, el modelo de requisitos consiste de tres modelos principales, representado por un diagrama de tres dimensiones como se muestra en la Figura 2.

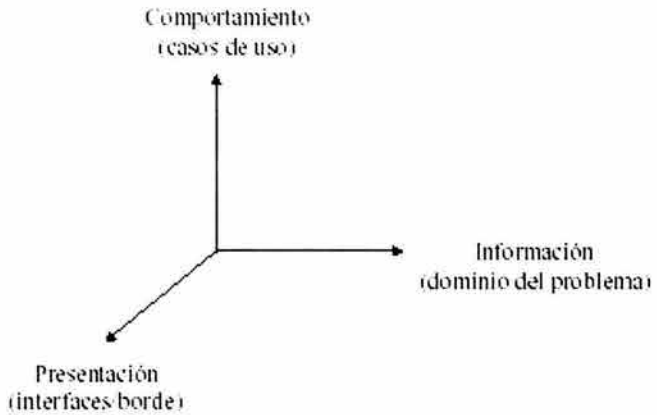


Fig.2: Los tres ejes de modelado del modelo de requisitos.

El modelo de *comportamiento*, basado directamente en el modelo de *casos de uso*, especifica la funcionalidad que ofrece el sistema desde el punto de vista del usuario. Este modelo utiliza dos conceptos claves: *actores* para representar los distintos papeles que los usuarios pueden jugar con el sistema, y *casos de uso* para representar qué pueden hacer los actores con respecto al sistema

El modelo de *presentación* o modelo de *interfaces* o *borde* especifica cómo interactúa el sistema con actores externos al ejecutar los casos de uso, en particular, en los sistemas de información ricos en interacción con el usuario, especifica cómo se verán visualmente las interfaces gráficas y que funcionalidad ofrecerá cada una de ellas.

El modelo de *información* o modelo del *dominio del problema* especifica los aspectos estructurales del sistema. Este modelo conceptualiza el sistema según los objetos que representan las entidades básicas de la aplicación.

Es importante resaltar que esta separación en tres ejes de modelado independientes es la base para una mayor estabilidad en el desarrollo del sistema, permitiendo minimizar los efectos de cada uno sobre los otros dos.

Descripción del Problema

La descripción del problema es una descripción muy preliminar de necesidades que sirve únicamente como punto de inicio para comprender los requisitos del sistema. Se trata aquí de simular una descripción preparada por el usuario final la cual debe evolucionar por medio del modelo de requisitos para lograr la especificación final del sistema a desarrollarse. La descripción del problema debe ser una descripción de necesidades y no una propuesta para una solución. La descripción inicial puede ser incompleta e informal. No hay razón para esperar que la descripción inicial del problema, preparada sin un análisis completo, sea correcta.

3.2 MODELO DE CASOS DE USO.

El *modelo de casos de uso* describe un sistema en término de sus distintas formas de utilización, cada uno de estas formas es conocida como un *caso de uso*. Cada caso de uso o flujo se compone de una secuencia de eventos iniciada por el usuario. Dado que los casos de uso describen el sistema a desarrollarse, cambios en los requisitos significarán cambios en los casos de uso. El actor y el caso de uso representan los dos elementos básicos de este modelo.

Actores

Los *actores* son entidades distintas a los usuarios, en el sentido que los usuarios son las personas reales que utilizan el sistema, mientras que los actores representan un cierto papel que una persona real puede jugar. Utilizando terminología orientada a objetos, se considera al actor como una *clase* de usuario, mientras que los usuarios se consideran como *objetos* o instancias de esa clase. Incluso, una misma persona puede aparecer como diferentes instancias de diferentes actores.

Los actores modelan cualquier entidad externa que necesite intercambiar información con el sistema. Los actores no están restringidos a ser personas físicas, pudiendo representar otros sistemas externos al actual. Lo esencial es que los actores representen entidades externas al sistema. Además, cada uno de estos actores podrá ejecutar una o más tareas del sistema.

Antes de identificar los casos de uso se identifican los actores del sistema. La razón para comenzar con la identificación de los actores es para que ellos sean la herramienta principal para luego encontrar los casos de uso. Cada actor ejecuta un número específico de casos de uso en el sistema. Al definir todos los actores y casos de uso en el sistema, se define la funcionalidad completa del sistema.

Para poder identificar los casos de uso, es necesario primero identificar los actores del sistema, comenzando por aquellos que son la razón principal del sistema, conocidos como *actores primarios*. Estos actores típicamente rigen la secuencia lógica de ejecución del sistema. Además de los actores primarios existen actores supervisando y manteniendo el sistema. Estos *actores secundarios* existen primordialmente como complemento a los actores primarios. Los actores secundarios tienden a responder a secuencias lógicas del sistema y no tanto a inicializarlas de manera propia. En particular, existe siempre la duda, por ejemplo, de si el sistema operativo o una base de datos serían actores. La decisión depende del papel que jueguen con respecto al sistema en desarrollo, si juegan un papel activo entonces deben modelarse como actores.

Cuando diferentes actores juegan roles similares ellos pueden heredar de un *actor abstracto* común, El resto de los actores se conoce como actores concretos. La ventaja de modelar actores abstractos es que expresa similitudes entre caso de uso. Si el mismo o parte del mismo caso de uso se puede ejecutar por varios actores diferentes, el caso de uso necesita ser especificado solo con respecto a un actor en lugar de varios. Por otro lado, los actores abstractos también pueden ser utilizados para especificar privilegios comunes a múltiples actores en un sistema.

Casos de Uso

Después de haber definido los actores del sistema, se define la funcionalidad propia del sistema por medio de los *casos de uso*. Utilizando terminología orientada a objetos, cada caso de uso define una clase o forma particular de usar el sistema mientras que cada ejecución del caso de uso se puede ver como una instancia del caso de uso, o sea, un objeto, con estado y comportamiento. Cada caso de uso constituye un flujo completo de eventos especificando la interacción que toma lugar entre el actor y el sistema. El actor primario es encargado de dar inicio a esta interacción, mientras que los casos de uso son instanciados como respuesta al evento anterior. Una instancia de un actor puede ejecutar varias de estas secuencias, consistiendo de diferentes acciones que a su vez deben llevarse a cabo. La instancia del caso de uso existe mientras el caso de uso siga ejecutando. La ejecución del caso de uso termina cuando el actor genere un evento que requiera un caso de uso nuevo. Las diferentes instancias de los casos de uso se conocen como *escenarios*.

Como varios casos de uso pueden comenzar de una misma forma, no es siempre posible decidir que caso de uso se ha instanciado hasta que éste se haya completado. La descripción de los casos de uso es mediante diagramas similares a los de transición de estados. Se puede ver a cada caso de uso como representando un estado en el sistema, donde un estímulo enviado entre un actor y el sistema ocasiona una transición entre estados.

–

Para identificar los casos de uso, se puede leer la descripción del problema y discutirlo con aquellos que actuarán como actores, haciendo preguntas como:

¿Cuales son las tareas principales de cada actor?

¿Tendrá el actor que consultar y modificar información del sistema?

¿Tendrá el actor que informar al sistema sobre cambios externos?

¿Desea el actor ser informado sobre cambios inesperados?

Los casos de uso pueden comenzar de forma similar y no podemos saber cual se está instanciando hasta que se termine. En otras palabras, el actor no requiere que un caso de uso se ejecute, sólo que inicie una secuencia de eventos que finalmente resulte en la terminación de algún casos de uso. Los nombres de los casos de uso deben corresponder a acciones y no tanto a sustantivos.

Existen ciertas extensiones al concepto básico de caso de uso que permiten subdividirlos de manera limitada. Para ello se permite la asignación de secciones del flujo completo de un caso de uso en casos de uso separados. Las razones para esto son aprovechar distintas variantes en los casos de uso que se repiten en la lógica del sistema de manera análoga al concepto de herencia. Lamentablemente, no es siempre obvio la funcionalidad que debe ser puesta en casos de uso separados, y qué situación es sólo una pequeña variante de un mismo caso de uso que no amerita tal división. La complejidad de los casos de uso es un factor importante para tal decisión. Existen dos enfoques distintos para expresar variantes:

- Si las diferencias entre los casos de uso son pequeñas, estos se pueden describir como variantes de un mismo caso de uso, y se pueden definir *subflujos* separados dentro de un mismo caso de uso, dando lugar al concepto de flujos *básicos* y subflujos *alternos*. Por ejemplo, en el caso de uso registrar un usuario, crear por primera vez el registro y actualizarlo se pueden considerar como dos secuencias de eventos lógicamente similares por lo cual se tratan como subflujos alternos. En general, primero se describe el *flujo básico*, el cual es el flujo más importante dentro del caso de uso. Este flujo corresponde a la secuencia de eventos más común o típica de casos de uso. Posteriormente se describen las variantes o *flujos alternos* que incluyen flujos para el manejo de variantes en la lógica. Normalmente un caso de uso tiene sólo un curso básico, pero varios cursos alternos.
- Si las diferencias entre los casos de uso son grandes, se deben describir como casos de uso separados. Cuando los casos de uso se dividen en casos de uso separados se utiliza principalmente las relaciones de *extensión*, *inclusión* y generalización.

Extensión

Un concepto importante que se utiliza para estructurar y relacionar casos de uso es la *extensión*. La extensión especifica cómo un caso de uso puede *insertarse* en otro para extender la funcionalidad del anterior. El caso de uso donde se va a insertar la nueva funcionalidad debe ser un flujo completo, por lo cual éste es independiente del caso de uso a ser insertado. De esta manera, el caso de uso inicial no requiere consideraciones adicionales al caso de uso a ser insertado, únicamente especificando su punto de inserción.

Inclusión

Una relación adicional entre casos de uso es la *inclusión*. A diferencia de una extensión, la inclusión se define como una sección de un caso de uso que es parte obligatoria del caso de uso básico. El caso de uso donde se va a insertar la funcionalidad depende del caso de uso a ser insertado. Se etiqueta la relación con "incluye" ("include").

Generalización

Una relación adicional entre casos de uso es la generalización la cual apoya la reutilización de los casos de uso. Mediante la relación de generalización es necesario describir las partes similares una sola vez en lugar de repetirlas para todos los casos de uso con comportamiento común. Se conoce a los casos de uso que se extraen como casos de uso *abstractos*, ya que no serán instanciados independientemente, sirviendo sólo para describir partes que son comunes a otros casos de uso. Se conoce a los casos de uso que realmente serán instanciados como casos de uso *concretos*. Las descripciones de los casos de uso abstractos se incluyen en las descripciones de los casos de uso concretos. Esto significa que, cuando una instancia de un caso de uso sigue la descripción de un caso de

uso concreto, en cierto punto continúa la descripción del caso de uso abstracto en su lugar. Los casos de uso abstractos también pueden ser usados por otros casos de uso abstractos. Es difícil saber cuando ya no tiene sentido seguir extrayendo más casos de uso abstractos. Una técnica para extraer casos de uso abstractos es identificar actores abstractos. Normalmente, no hay razón para crear casos de uso abstractos que se usan en un solo caso de uso. Por lo general, comportamientos similares entre casos de uso se identifica después que se han descrito los casos de uso. Sin embargo, en algunos casos es posible identificarlos antes.

Las relaciones de *extensión* e *inclusión* entre casos de uso son ambas asociaciones de clases, a diferencia de la *generalización*. El criterio importante es ver qué tanto se acoplan las funcionalidades de los casos de uso. Si el caso de uso a ser extendido es útil por si mismo, la relación debe ser descrita utilizando extensión. Si los casos de uso son fuertemente acoplados, y la inserción debe tomar lugar para obtener un curso completo, la relación debe ser descrita utilizando inclusión. Por otro lado, la generalización es utilizada cuando dos o más casos de uso comparte funcionalidad común la cual es extendida por cada uno al estilo de la generalización entre clases. También hay una diferencia en cómo se encuentran. Las extensiones se encuentran en un caso de uso existente que el usuario desea ramificar en secuencias adicionales, mientras que las inclusiones se encuentran de la extracción de secuencias comunes entre varios casos de uso diferentes. Las generalizaciones se encuentran al tratar de especializar de diferente manera un mismo caso de uso.

Finalmente, se desean diagramas con baja complejidad que no sean "telarañas" pero que muestren de manera esquemática las posibles secuencias de interacciones entre los actores y el sistema.

Documentación

Parte fundamental del modelo de casos de uso es una descripción textual detallada de cada uno de los actores y casos de uso identificados. Estos documento son sumamente críticos ya que a partir de ellos se desarrollará el sistema completo. El formato de documentación es el siguiente:

Actor:	Nombre del actor
Casos de Uso:	Nombre de los casos de uso en los cuales participa
Tipo:	*Primario* o *Secundario*
Descripción:	Breve descripción del actor

Las descripciones de los casos de uso representan todas las posibles interacciones de los actores con el sistema para los eventos enviados o recibidos por los actores. En esta etapa no se incluyen eventos internos al propio sistema ya que esto será tratado durante el análisis y únicamente agregaría complejidad innecesaria en esta etapa. El formato de documentación es el siguiente:

Caso de Uso:	Nombre del caso de uso
Actores:	Actores primarios y secundarios que interaccionan con el caso de uso.
Tipo:	Tipo de flujo: *Básico, Inclusión, Extensión, Generalización*, o algún otro.
Propósito:	Razón de ser del caso de uso.
Resumen:	Resumen del caso de uso.
Precondiciones:	Condiciones que deben satisfacerse para poder ejecutar el caso de uso.
Flujo Principal:	El flujo de eventos más importante del caso de uso, donde dependiendo de las acciones de los actores se continuará con alguno de los subflujos.
Subflujos:	Los flujos secundarios del caso de uso, numerados como (S-1), (S-2), etc.
Excepciones:	Excepciones que pueden ocurrir durante el caso de uso, numerados como (E-1), (E-2), etc.

Dado que el modelo de casos de uso está motivado y enfocado principalmente hacia los sistemas de información donde los usuarios juegan un papel primordial, es importante ya relacionarse con las interfaces a ser diseñadas en el sistema. Estas interfaces sirven para apoyar de mejor manera la descripción de los casos de uso además de servir de base para prototipos iniciales.

3.3 MODELO DE INTERFACES

El *modelo de interfaces* describe la presentación de información entre los actores y el sistema. Se especifica en detalle como se verán las interfaces de usuario al ejecutar cada uno de los casos de uso.

REQUISITOS

En el Sistema de Gestión de Información (SGI) de la UAAI de la ENEO se almacenan los datos de acervo, usuarios y catálogos. Dependiendo del usuario, este tendrá acceso a diferentes funciones en el sistema, que podrán incluir el agregar, modificar, omitir o ver nueva información en el sistema.

De acuerdo a la función dentro de la UAAI de la ENEO se pueden distinguir las siguientes personas que pueden interactuar con el sistema son:

* Coordinador de la UAAI de la ENEO
* Administrador de la UAAI de la ENEO
* Usuario de la UAAI de la ENEO

OPERACIONES

Según la persona que interactúa con el sistema, tenemos que se desarrollan diferentes actividades con el sistema, estas son:

CASO USUARIO

Para que el usuario pueda acceder al sistema debe realizar previamente su inscripción a la UAAI de la ENEO, el sistema presentará una pantalla con las opciones de inscripción o reinscripción, después de ser inscrito (o reinscrito) se le imprimirá una credencial con sus datos. Su inscripción tendrá un lapso de vigencia de 6 meses y el sistema detectará cuando se venza la credencial.

- Validar Cliente
- Validar Usuario
- Consulta de material existente

Los casos de clientes siguientes no requieren de validación de usuario para acceder a la UAAI de la ENEO, y tendrán un Login para trabajar con el sistema (Validar Cliente).

CASO ADMINISTRADOR

- *Validar Cliente*
- *Altas, bajas, actualizaciones y consultas del material existente.*
- *Altas, bajas, actualizaciones y consultas de los Usuarios existentes.*
- *Impresión de credenciales de Usuario.*
- Altas y actualización de Usuarios por áreas
- Elaboración y consulta de reportes de Acervo.
- Elaboración y consulta de reportes de Usuarios

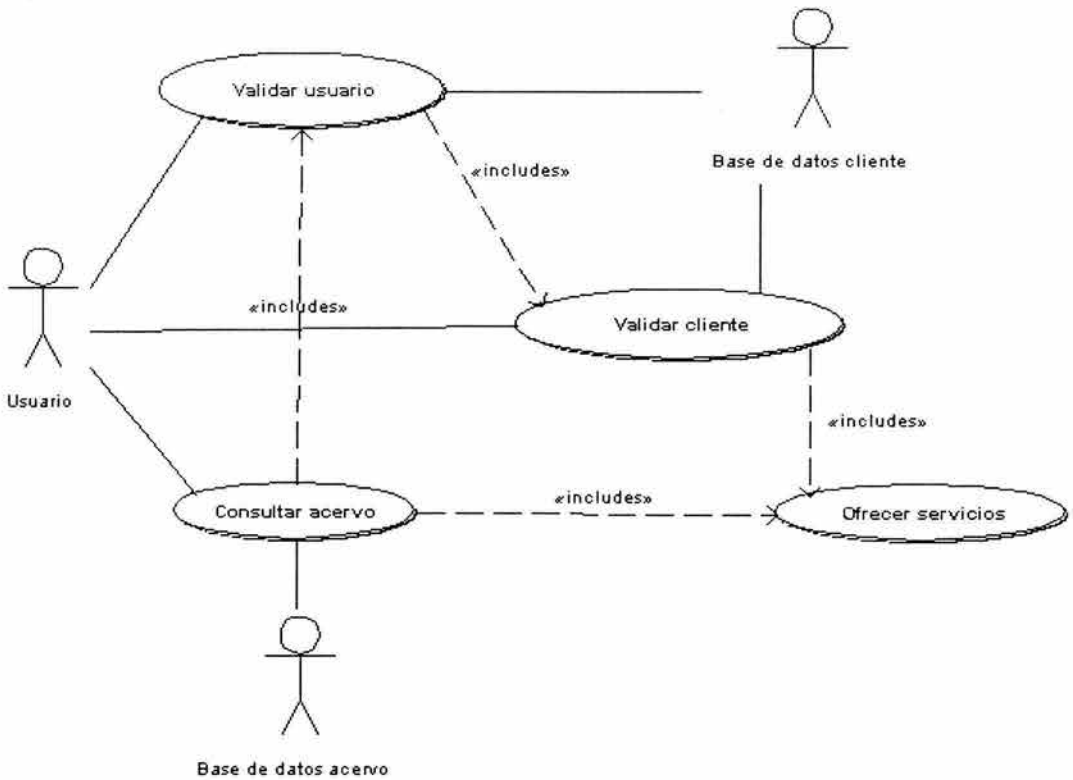
CASO COORDINADOR

- *Validar Cliente*
- *Altas, bajas, actualizaciones y consultas del material existente.*
- *Altas, bajas, actualizaciones y consultas de los Usuarios existentes.*
- *Impresión de credenciales de Usuario.*
- *Altas, bajas, actualizaciones y consultas* de Clientes.
- Altas y actualización de Usuarios por áreas
- Elaboración y consulta de reportes de Acervo.
- Elaboración y consulta de reportes de Usuarios

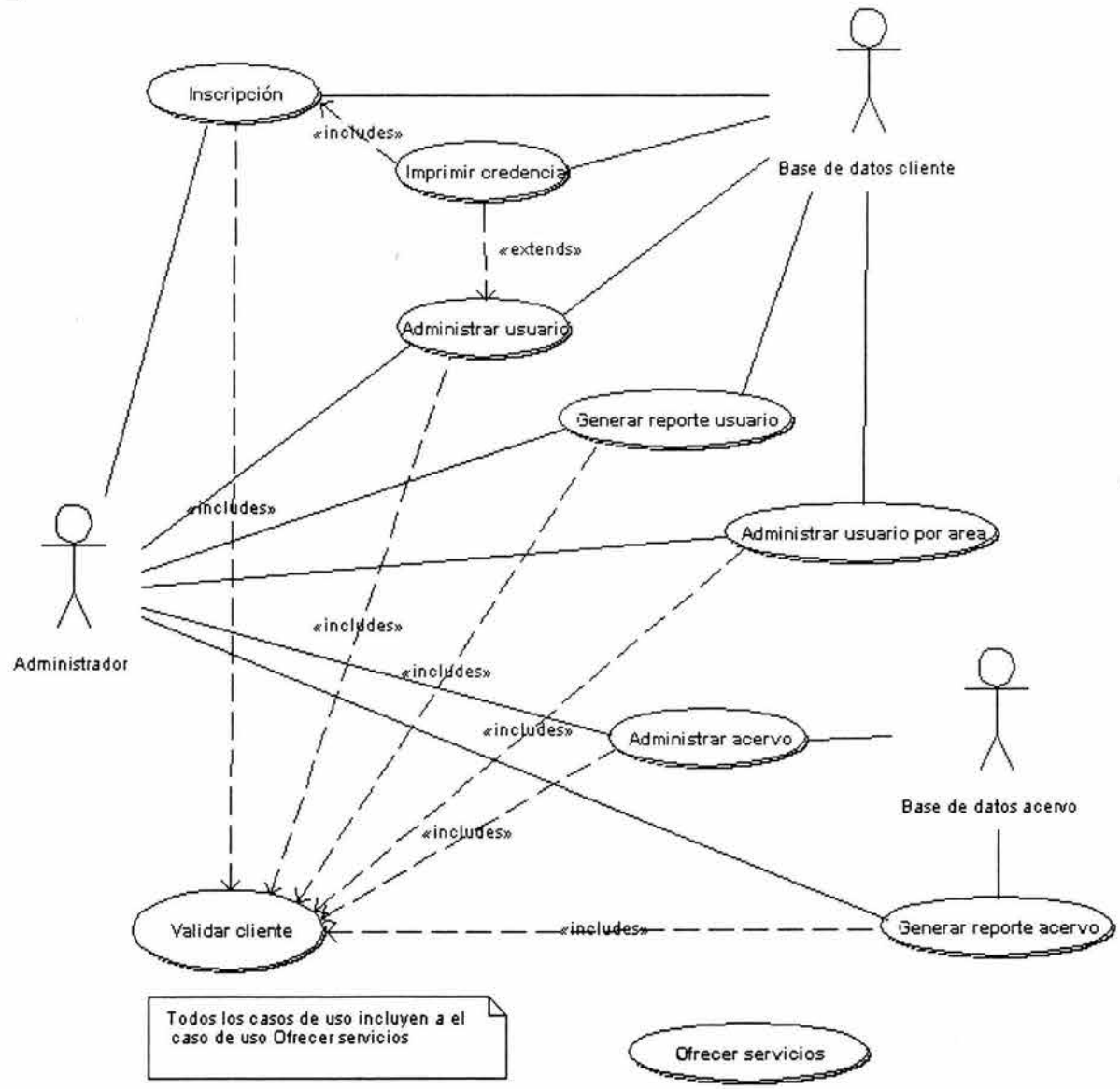
Diagramas de los casos de uso.

Se representaron dos bases de datos (base de datos clientes y base de datos acervo) para una mejor comprensión de la lógica del sistema ya que físicamente solo existirá una base de datos.

Caso Usuario.



Caso Administrador.



Caso Coordinador.

Actores del Sistema

Actores primarios: Coordinadores, Administrativos, Usuarios.
Actor secundario: Base de Datos.

Actor:	**Usuario**
Casos de Uso:	Validar Usuario, Consultar Acervo, Validar Cliente, Ofrecer Servicios.
Tipo:	Primario
Descripción:	Es un actor principal y es un Usuario de la UAAI de la ENEO

Actor:	**Administrador**
Casos de Uso:	Validar Cliente, Administrar Acervo, Ofrecer Servicios, Inscripción, Administrar Usuario, Imprimir Credencial, Administrar Acervo, Generar Reporte Acervo, Generar Reporte Usuario, Administrar Usuarios por Área.
Tipo:	Primario
Descripción:	Es un actor principal tiene un nivel de interacción con el sistema mayor que un Usuario.

Actor:	**Coordinador**
Casos de Uso:	Validar Cliente, Administrar Acervo, Ofrecer Servicios, Inscripción, Administrar Usuario, Imprimir Credencial, Administrar Acervo, Generar Reporte Acervo, Generar Reporte Usuario, Administrar Cliente, Administrar Usuarios por Área.
Tipo:	Primario
Descripción:	Es un actor principal hereda los casos de uso del actor Administrador, agregando el caso de uso, Administrar Cliente, es el actor con mayores servicios en el sistema.

Actor:	**Base de Datos**
Casos de Uso:	Validar Cliente, Validar Usuario, Inscripción, Administrar Usuario, Imprimir Credencial, Generar Reportes Usuario, Administrar Cliente, Administrar Usuarios por Área, Consultar Acervo, Administrar Acervo, Generar Reportes Acervo.
Tipo:	Secundario
Descripción:	Es un actor secundario y representa las base de datos donde se guarda toda la información relacionada con los Clientes (Usuario, Administrador y Coordinador) y el Acervo.

Casos de Uso del Sistema

Caso de Uso:	**Validar Cliente**
Actores:	Usuario, Administrador, Coordinador, Base de Datos
Tipo:	Inclusión
Propósito:	Validar a un Cliente (Usuario, Administrador, Coordinador) ya registrado para el uso del SGI-ENEO
Resumen:	Este caso es iniciado por un Cliente, validándolo mediante un *login* y *password*, así como, con la selección de una de las tres opciones presentadas (Coordinador, Administrador, Usuario) comparando estos datos con los almacenados en los registros de la Tabla Registro Cliente de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente los Casos de Uso: Inscripción (Usuario), Administrar Cliente (Administrador, Coordinador)
Flujo principal:	Se presenta al Cliente la pantalla P–1. El Cliente puede seleccionar entre las siguientes opciones: "Aceptar" y "Cancelar". Si la actividad seleccionada es "Aceptar" se valida el registro del Cliente mediante los datos proporcionados (una de las tres opciones y su *login* y *password*). Una vez validado el Cliente (Ex–1), se continua con el caso Ofrecer Servicios

	y dependiendo del tipo de Cliente, el subflujo será S–2 (Usuario), S–3 (Administrador), S–4(Coordinador). Si la actividad seleccionada es "Cancelar" se continua con el caso *Ofrecer Servicios* subflujo S–1.
Subflujos:	Ninguno
Excepciones:	EX –1 no hubo validación, la contraseña (*login* y *password*) no corresponde, se cancelan los datos proporcionados y se manda un mensaje de error, y enseguida se solicita al Cliente validarse de nuevo.

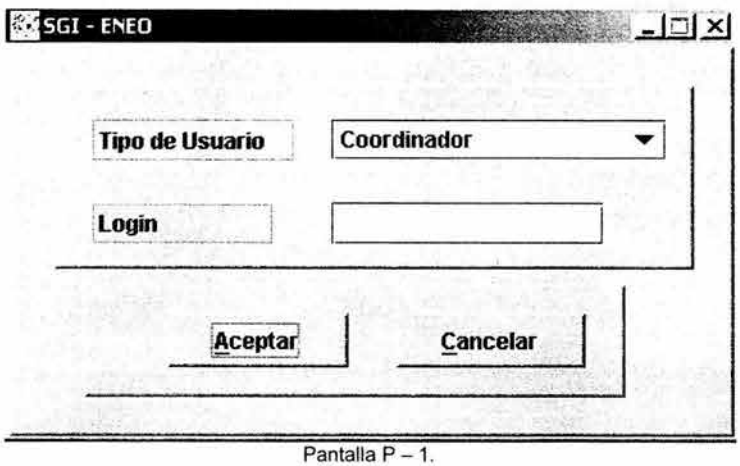
Caso de uso:	**Administrar Acervo**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Permite a un Administrador ó Coordinador modificar un registro del material contenido en el Acervo para actualizarlo.
Resumen:	Este caso es iniciado por un Administrador o Coordinador. Ellos son los que crean el registro del material en la Tabla Registro Acervo de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Validar Cliente*.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió la opción Modificar Acervo, seguirá la ejecución del Caso de Uso *Administrar Acervo* subflujo (S–1).
Subflujos:	S–1 *Administrar Registro Acervo.* Se presenta al Administrador ó Coordinador la Pantalla *Administrar Acervo(P-8(a))* con la información del primer registro de la Tabla Registro Acervo de la Base de Datos. El Administrador ó Coordinador podrá seleccionar entra las siguientes actividades: "Eliminar", "Modificar", "Actualizar"(inactivo hasta que se haga clic en "Modificar"), "Búsqueda", "Primer Registro", "Último Registro", "Siguiente", "Anterior", "Crear" y "Salir". Si el Coordinador ó Administrador presiona "Modificar" se ejecuta el subflujo *Actualizar Registro Acervo*(*S*-2). Si el Coordinador ó Administrador selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Acervo* (*S*-3). Si el Coordinador ó Administrador selecciona "Búsqueda" se ejecuta el subflujo *Búsqueda Registro* (*S*-4). Si el Coordinador ó Administrador selecciona "Primer Registro" se ejecuta el subflujo *Recupera Primer Registro* (*S*-5). Si el Coordinador ó Administrador selecciona "Último Registro" se ejecuta el subflujo *Recupera Último Registro* (*S*-6). Si el Coordinador ó Administrador selecciona "Siguiente" se ejecuta el subflujo *Recupera Registro Siguiente* (*S*-7). Si el Coordinador ó Administrador selecciona "Anterior" se ejecuta el subflujo *Recupera Registro Anterior* (*S*-8). Si el Coordinador selecciona "Crear" se ejecuta el subflujo *Crear Registro* (*S*-9). Si la actividad seleccionada es "Salir" se ejecutará el caso de uso *Ofrecer Servicios* (si aún no se ha presionado "Actualizar", la nueva información será perdida).
	S-2 Actualizar Registro Acervo Se activa el botón "Actualizar", el Coordinador ó Administrador introduce los nuevos datos, enseguida da clic en "Actualizar" el cual actualiza el registro del Acervo con la información modificada (E-1), en caso de no hacer clic en el último botón mencionado, la nueva información no será almacenada. Se continúa con el subflujo *Administrar Registro Acervo*(S–1).

	S-3 Eliminar Registro Acervo Se elimina el registro del material y se continúa con el subflujo *Administrar Registro Acervo* (*S*-1).
	S – 4 Búsqueda Registro. Se presenta la pantalla P–8(b) la cual pide los datos del registro a buscar. Se presenta las siguientes opciones: "Buscar" y "Cancelar". Si la actividad seleccionada es "Buscar". Después de obtener el registro(E-2) se continua con el subflujo *Modificar Registro Acervo* (S–1). Si la actividad seleccionada es "Cancelar", se ejecuta el caso de uso *Administrar Registro Acervo* (S–1).
	S – 5 Recuperar Primer Registro. Se presenta la pantalla P–8(a) con los datos de los campos del Primer registro de la Tabla Registro Acervo de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Acervo* (S–1).
	S – 6 Recuperar Último Registro. Se presenta la pantalla P–8(a) con los datos de los campos del Último registro de la Tabla Registro Acervo de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Acervo*(S–1).
	S – 7 Recuperar Registro Siguiente. Se presenta la pantalla P–8(a) con los datos de los campos del Siguiente registro de la Tabla Registro Acervo de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Acervo*(S–1).
	S – 8 Recuperar Registro Anterior. Se presenta la pantalla P–8(a) con los datos de los campos del Anterior registro de la Tabla Registro Acervo de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Acervo*(S–1).
	S – 9 Crear Registro. Se presenta la pantalla P–8(c) la cual tienen dos opciones: "Crear" y "Cancelar". Si la actividad seleccionada es "Crear" se crea un registro nuevo en la Tabla Registro Acervo de la Base de Datos (E–1) con los datos proporcionados anteriormente en los distintos campos, posteriormente se continua con el subflujo *Administrar Registro Acervo* (S–1). Si la actividad seleccionada es "Cancelar" se ejecuta el caso de uso *Administrar Registro Acervo* (S–1).
Excepciones:	E–1 *información incompleta.* Falta llenar información en el registro de Acervo. Se vuelve a solicitar al Coordinador ó Administrador que complete el registro. E-2 *información no encontrada.* Los datos del registro solicitado no son iguales a ningún registro alojado en la Tabla Registro Acervo de la Base de Datos.

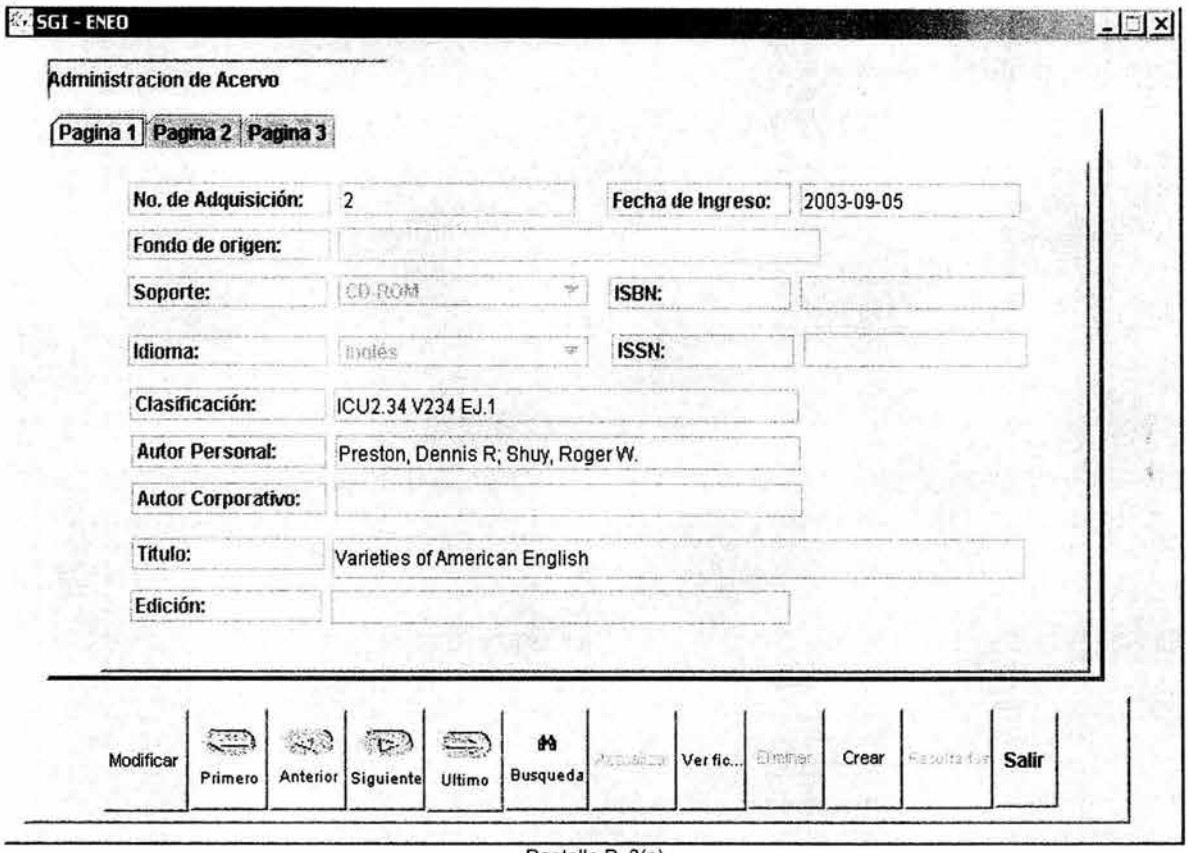
Es conveniente señalar que el SGI consta de catorce casos de uso. Los doce no mencionados en este capítulo, se encuentran en la sección de Anexos de este trabajo.

Pantallas.

Pantalla Ofrecer servicios.

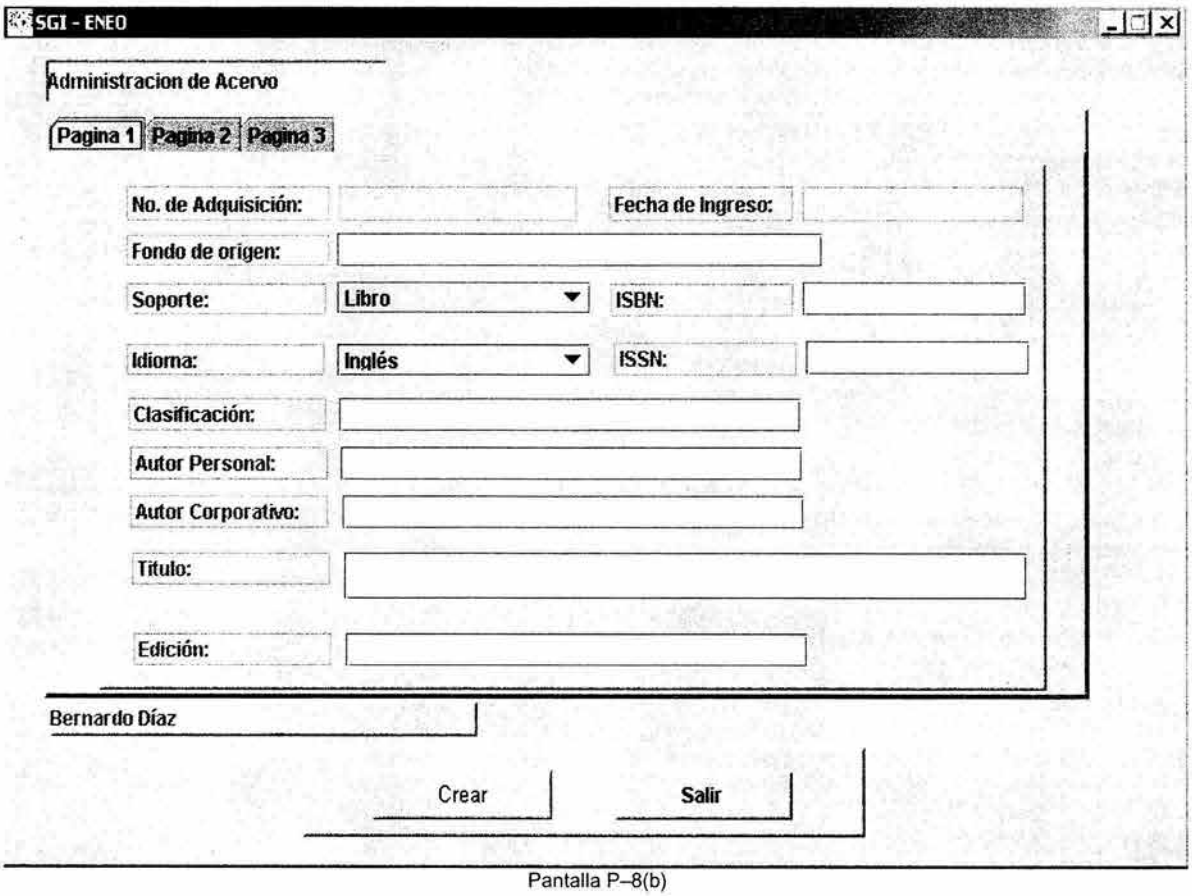


Pantalla P – 1.



Pantalla P–8(a)

Pantalla P–8(b)



Pantalla P–8(b)

3.3.1 Modelo del dominio del problema.

El *modelo del dominio del problema* define un modelo de clases común para todos los involucrados en el modelo de requisitos. Este modelo de clases consiste en los objetos del dominio del problema, es decir, objetos que tienen una correspondencia directa en el área de la aplicación. Como los usuarios y clientes deberían reconocer todos los conceptos, se puede desarrollar una terminología común al razonar sobre los casos de uso, y por lo tanto disminuyendo la probabilidad de malos entendimientos entre el analista y el usuario. Al discutirlo, se evolucionará el modelo del dominio del problema. Una técnica utilizada cuando se trabaja con tal modelo es darle al cliente un papel y un lápiz y pedirle que dibuje su visión del sistema.

Históricamente, el modelo del dominio del problema se utilizaba como el modelo de requisitos fundamental en ciertas metodologías, tales como Coad y Yourdon (1991), Booch (1991), y Rumbaugh (1991). Sin embargo, dadas sus limitaciones que impedía obtener los requisitos funcionales de un sistema, el modelo del dominio del problema dejó de ser la base única para el desarrollo completo del sistema y pasó a ser un elemento adicional en la especificación de los sistemas, como en el modelo de casos de uso. El propósito principal del dominio del problema en el modelo de requisitos de la metodología *Objetory* es formar una base común de entendimiento del desarrollo y no para definir el sistema completo. Por lo tanto, se pueden aprovechar algunas de las heurísticas de los métodos anteriores para la identificación de los objetos en el dominio del problema, logrando un *glosario* o *diccionario de clases* que sirve como común denominador a todos los componentes del sistema, incluyendo a las diversas personas involucradas a lo largo del desarrollo. A diferencia de los métodos anteriores, el modelo del dominio del problema no debe ser demasiado extenso, ya que varios grados de refinamiento serán hechos posteriormente. Y aunque es suficiente describir el dominio del problema en

término de objetos o clases, es posible refinar más aún mediante la inclusión de asociaciones, atributos, herencia y operaciones, siempre y cuando esto ayude a comprender mejor el problema y no se vuelva un esfuerzo demasiado grande durante esta etapa.

A continuación se describirá la manera de identificar las clases del dominio del problema junto con aspectos adicionales, cómo asociaciones y atributos. Sin embargo, no se identificará la herencia y las operaciones durante esta etapa. La herencia y en especial las operaciones de un sistema son los aspectos de mayor complejidad, algo que nosotros elaboraremos de manera muy cuidadosa durante el diseño del sistema.

3.3.1.1 Identificación de Clases

La identificación de clases del dominio del problema se obtiene principalmente de algún documento textual que describa el sistema. Aunque pudiéramos tomar como punto de partida los documentos desarrollados para el modelo de casos de uso, a menudo la descripción original del problema es suficiente. Se comienza este proceso mediante la identificación de las clases *candidatas*, explícitas o implícitas, a las que se refiera la descripción del problema. Para ello se extraen todos los sustantivos de la descripción del problema o de algún otro documento similar, de acuerdo a las siguientes consideraciones.

Los sustantivos en la descripción del problema son los posibles candidatos a clases de objetos. Durante esta etapa, se debe identificar entidades físicas al igual que entidades conceptuales. No se debe tratar de diferenciar entre clases y atributos durante ésta etapa.

Dado que no todas las clases se describen de manera explícita, siendo algunas implícitas en la aplicación, será necesario añadir clases que pueden ser identificadas por nuestro conocimiento del área. Se debe revisar los pronombres en la descripción del problema para asegurar que no se haya perdido ningún sustantivo descrito de forma implícita.

Para facilitar la identificación de clases, se subrayan todos los sustantivos de la descripción del problema. Se parte de la descripción del problema y subrayamos todos los sustantivos, como se ve a continuación:

REQUISITOS

En el Sistema de Gestión de Información de la UAAI de la ENEO se almacenan los datos de acervo, usuarios, catálogos. Dependiendo del usuario, este tendrá acceso a diferentes funciones en el sistema, que podrán incluir el agregar, modificar, omitir o ver nueva información en el sistema.

De acuerdo a la función dentro de la UAAI de la ENEO se pueden distinguir las siguientes personas que pueden interactuar con el sistema son:

* Coordinador de la UAAI de la ENEO
* Administrador de la UAAI de la ENEO
* Usuario de la UAAI de la ENEO

OPERACIONES

Según la persona que interactúa con el sistema, tenemos que se desarrollan diferentes actividades con el sistema, estas son:

CASO USUARIO

Para que el usuario pueda acceder al sistema debe realizar previamente su inscripción a la UAAI de la ENEO, el sistema presentará una pantalla con las opciones de inscripción o reinscripción, después de ser inscrito (o reinscrito) se le imprimirá una credencial con sus datos, un *login* y un *password,* por medio de los cuales podrá validar en el sistema su acceso al lugar. Su inscripción tendrá un lapso de vigencia y el sistema detectará cuando se venza la credencial.

Una vez dentro de la UAAI de la ENEO el usuario, podrá realizar las siguientes actividades:

- Consulta de material existente

Los casos de clientes siguientes no requieren de validación para acceder a la UAAI de la ENEO, y tendrán un Login y Password para trabajar con el sistema.

CASO ADMINISTRADOR

- *Altas, bajas, actualizaciones y consultas del material existente.*
- *Altas, bajas, actualizaciones y consultas de los Usuarios existentes.*

–

- *Impresión de etiquetas de Acervo y credenciales de Usuario.*
- Altas y actualizaciones de Usuarios por área.
- Elaboración y consulta reportes de Acervo.
- Elaboración y consulta de reportes de Usuarios.

CASO COORDINADOR

- *Altas, bajas, actualizaciones y consultas del material existente.*
- *Altas, bajas, actualizaciones y consultas de los Usuarios existentes.*
- *Impresión de etiquetas de Acervo y credenciales de Usuario.*
- Altas y actualizaciones de Usuarios por área.
- Elaboración y consulta de reportes de Acervo.
- Elaboración y consulta de reportes de Usuarios
- *Altas, bajas, actualizaciones y consultas* de Clientes existentes.

Clases Candidatas	
SGI – ENEO	Login
Datos de acervo	Password
Usuario	Altas de material
Catálogos	Bajas de material
Usuario	Actualizaciones de material
acceso	Consulta de material
funciones del sistema	Altas de usuarios
información	Bajas de usuarios
sistema	Actualizaciones de usuarios
Coordinador	Consultas de usuarios
Administrador	Impresión de etiquetas de acervo
Usuario	Impresión de credenciales usuario
Actividades	Reportes de Acervo
Inscripción	Reportes de usuarios
Pantalla	Altas de Clientes
inscripción	Bajas de Clientes
reinscripción	Actualizaciones de Clientes
Lapso de vigencia	Consultas de Clientes
Datos	Usuarios por Área
Credencial	Casos de Clientes

Clases candidatas para el SGI - ENEO. Identificadas de la descripción del problema.

3.3.1.2 Selección de Clases

A partir de las clases candidatas, se deben seleccionar las clases relevantes tomando en cuenta los siguientes consideraciones:

- Todas las clases deben tener sentido en el área de la aplicación, la relevancia al problema debe ser el único criterio para la selección.

- Como regla general, se debe escoger los nombres para las clases con cuidado, que no sean ambiguos y que mejor se apliquen al problema. Este es uno de los procesos más difíciles. Los nombres deben ser establecidos con un formato consistente (por ejemplo nombres en singular).

- No hay que preocuparse durante esta etapa sobre asociación, agregación, o herencia. Primero hay que tener las clases específicas correctas para no suprimir detalles en el intento de ajustarse a estructuras preconcebidas.

Ante la duda, se deben conservar las clases, ya que posteriormente siempre habrá oportunidad para eliminarlas.

- Se deben eliminar clases redundantes, si estas expresan la misma información. La clase más descriptiva debe ser guardada.

–

- Se deben eliminar clases irrelevantes, que tienen poco o nada que ver con el problema. Esto requiere juicio porque en un contexto una clase puede ser importante mientras que en otro contexto la clase podría no serlo.
- Se deben clarificar las clases imprecisas. Algunas clases pueden tener bordes mal definidos o demasiado generales.
- Se deben eliminar las clases que debieran ser atributos más que clases, cuando los nombres corresponden a propiedades más que entidades independientes.
- Se deben eliminar las clases que debieran ser roles más que clases, cuando los nombres corresponden al papel que juegan las clases más que entidades independientes.
- Se deben eliminar las clases que debieran ser operaciones más que clases, si las entidades representan operaciones que son aplicadas a los objetos y no entidades manipuladas por si mismas.
- Se deben eliminar las clases que corresponden a construcciones de implementación si están alejadas del mundo real, por lo cual deben ser eliminadas del análisis. No se necesita una clase para representar una entidad física. Por ejemplo: *subrutinas*, *listas, y arreglos*, son clases típicas de implementación; *Internet y World Wide Web* son el medio de implementación del sistema
- Se debe eliminar clases que correspondan aspectos de interfaces de usuario y no de la aplicación.
- Se debe eliminar clases que correspondan a todo un sistema completo.
- Se debe eliminar clases que correspondan a actores del sistema.
- Se deben agregar clases implícitas que no aparezcan en la descripción del problema.

Clases candidatas	Modificación
SGI – ENEO	Eliminada (sistema completo)
Datos de acervo	Eliminada(Imprecisa)
Usuario	Eliminada(actor)
Catálogo	Eliminada(imprecisa)
acceso	Eliminada(imprecisa)
funciones del sistema	Eliminada(imprecisa)
información	Eliminada(imprecisa
sistema	Eliminada(imprecisa)
Coordinador	Eliminada(actor)
Administrador	Eliminada(actor)
Usuario	Eliminada(actor)
Actividades	Eliminada(imprecisa)
Inscripción	Eliminada(operación)
Pantalla	Eliminada(interface)
Inscripción	Eliminada(operación)
Reinscripción	Eliminada(operación)
Lapso de vigencia	Eliminada(imprecisa)
Datos	Eliminada(imprecisa)
Credencial	Eliminada(imprecisa)
Casos de Clientes	Eliminada(imprecisa)
Login	Eliminada(atributo)
Password	Eliminada(atributo)
Altas de material	Eliminada(operación)
Bajas de material	Eliminada(operación)
Actualizaciones de material	Eliminada(operación)
Consulta de material	Eliminada(operación)
Altas de usuarios	Eliminada(operación)
Bajas de usuarios	Eliminada(operación)
Actualizaciones de usuarios	Eliminada(operación)
Consultas de usuarios	Eliminada(operación)
Altas de clientes	Eliminada(operación)
Bajas de clientes	Eliminada(operación)
Actualizaciones de clientes	Eliminada(operación)
	Eliminada(operación)

Consultas de clientes	Eliminada(imprecisa)
Reportes de acervo	Eliminada(imprecisa)
Reportes de usuario	Eliminada(operación)
Impresión de etiquetas de acervo	Eliminada(operación)
Impresión de credenciales usuarios	Renombrada: **UsuarioPorArea**
Usuarios por área	

Clases Identificadas	
RegistroAcervo	RegistroCliente
RegistroUsuario	UsuarioPorArea
Acentos	Estados
Carreras	Idiomas
Catalogadores	Secciones
CatDificultad	Soportes
Categorías	SubdivisionTematica
DivisionTematica	Subsecciones
Ubicaciones	TipoCliente

3.3.1.3 Diagrama de Clases

Después de haber identificado y seleccionado las clases, se debe construir el diagrama de clases para el dominio del problema. Puede ayudar a identificar clases adicionales, y servirá de base para encontrar las atributos y asociaciones entre ellas.

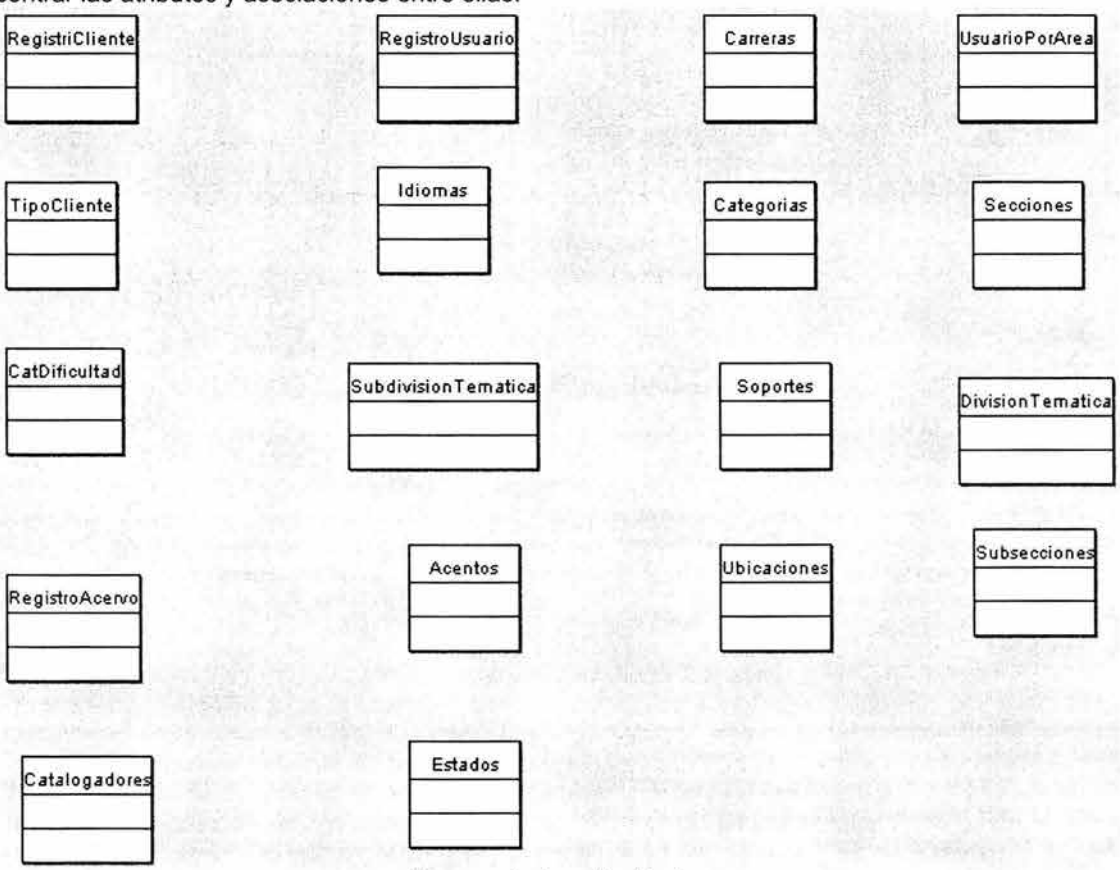


Diagrama de clases identificadas.

CAPÍTULO 4
MODELO DE ANÁLISIS

4.1 MODELO DE ANÁLISIS

El objetivo del modelo de análisis es comprender y generar una arquitectura de objetos para el sistema en base en lo especificado en el modelo de requisitos. El análisis pretende modelar el sistema bajo condiciones ideales, garantizando que la arquitectura de software resultante sea suficientemente robusta y extensible para servir de base a la estructura lógica de la aplicación pero sin consideraciones relativas al entorno de implementación que es posible que cambien incluso radicalmente. Durante esta etapa no se considera el ambiente de implementación, lo cual incluye al lenguaje de programación, manejador de base de datos, distribución o configuración de hardware, etc. Al final, el sistema tendrá que ser adaptado a las condiciones de implementación deseadas, algo que se hará durante el diseño, cuando todas las consideraciones que han sido descartadas durante el análisis sean consideradas.

Es importante enfatizar que el modelo de análisis no es una reflexión del dominio del problema sino una representación de ésta adaptada a la aplicación particular. El modelo de análisis genera una representación conceptual del sistema, consistiendo de clases de objetos. Cada una de las clases de objetos contribuye de manera especial para lograr la robustez de la arquitectura, como se muestra conceptualmente en la Figura 1.

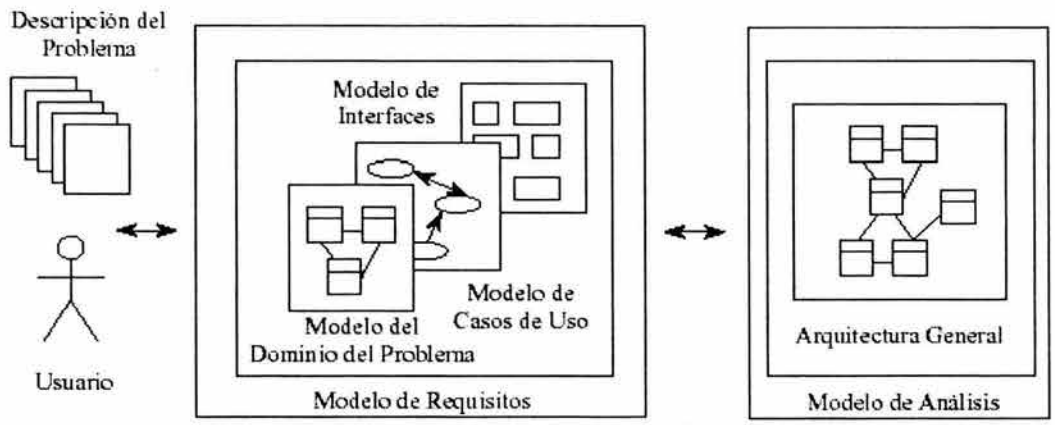


Fig. 1 El diagrama muestra conceptualmente el modelo de análisis junto con la arquitectura general de objetos en relación al modelo de requisitos anteriormente desarrollado.

Una cualidad importante entre los diferentes modelos es la *rastreabilidad* ("traceability") entre un modelo y otro, la cual relacionará sus distintos aspectos, sin importar el orden en que fueron desarrollados. Dado que la mayoría de los sistemas serán modificados a lo largo de su vida, si estos cambios surgen de cambios en los requisitos o respuestas a problemas, es necesario saber qué efectos tendrán estos sobre etapas posteriores incluyendo el código final.

Arquitectura de Clases

El modelo de análisis tiene como objetivo generar una arquitectura de objetos que sirva como base para el diseño posterior del sistema. Dependiendo del tipo de aplicación existen diversas arquitecturas que se pueden utilizar, siendo de interés aquellas arquitecturas diseñadas para el manejo de los sistemas de información, las cuales involucran bordes de usuario y accesos a base de datos como aspectos fundamentales de la arquitectura. En término de las arquitecturas, éstas se distinguen según la organización de la funcionalidad que ofrecen los objetos dentro de ellas o la *dimensión* de los objetos. Esta dimensión corresponde a los diferentes tipos de funcionalidad que manejan los objetos dentro la arquitectura. Es decir, si se tiene el caso de

funcionalidad para el manejo de bordes y base de datos, y existen tipos distintos de objetos para el manejo de cada una de estas por separado, entonces se considera que la arquitectura es de *dos* dimensiones. Pero, si todos los objetos manejan de manera indistinta los dos tipos de funcionalidades, entonces se considera que la arquitectura es de *una* sola dimensión.

Si se aplica el concepto de dimensión a los métodos estructurados, se puede ver que estos consisten de dos dimensiones: funciones y datos. Las *funciones* representan un eje de comportamiento que no guarda información, mientras que los *datos* se ubican en un eje de información que no contiene comportamiento.

Los ejes de funcionalidad corresponden a distintos tipos de funcionalidades. Con respecto al número y tipo de dimensiones para crear un sistema robusto y sensible a modificaciones, se debe aplicar el concepto de modularidad, que implica que cuanto mayor sea la modularidad de un sistema mayor es su robustez y extensibilidad. Es por esto que se debe diseñar un sistema en donde se logren ejes de funcionalidad ortogonales, esto es, el efecto de cambios en una dimensión no debe afectar a las otras dimensiones. Y aunque estas dimensiones no son del todo ortogonales, si son lo suficientemente independientes por lo que se puede limitar el efecto de posibles cambios. En relación al número de dimensiones, esto depende de la funcionalidad que la arquitectura debe manejar, algo que a su vez depende del tipo de aplicación que se está desarrollando.

Por tanto, para los sistemas de información, uno de los tipos de arquitectura más importante es la arquitectura *MVC – Modelo*, *Vista*, *Control* (*Model*, *View*, *Control*). Esta arquitectura se basa en tres dimensiones principales: *Modelo* (*información*), *Vista* (*presentación*) y *Control*(*comportamiento*) como se muestra en la Figura 2.

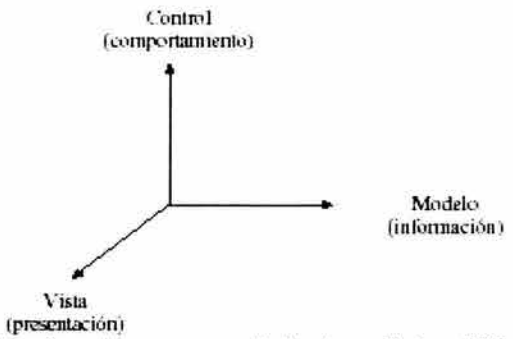


Fig. 2: Diagrama de tres dimensiones correspondiente a la arquitectura *MVC – Modelo, Vista, Control.*

La vista o presentación de la información corresponde a los bordes que se le presentan al usuario para el manejo de la información, donde existen múltiples vistas sobre un mismo modelo. La información representa el dominio del problema y es almacenada en una base de datos. El control corresponde a la manipulación de la información a través de sus diversas presentaciones. Y aunque existe dependencia entre estas tres dimensiones se considera que la manera de presentar la información es independiente de la propia información y de cómo se controla.

Es importante señalar que cada dimensión probablemente cambie a lo largo de la vida del sistema, donde el control es el más propenso a ser modificado, seguido de la vista y finalmente el modelo.

En este caso, el modelo de análisis usa como base la arquitectura *MVC* para emplear los tres aspectos que muestran la figura 3.

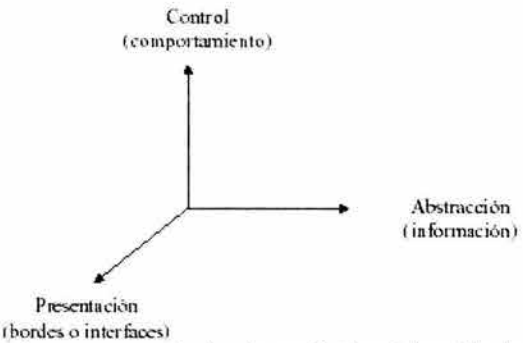


Fig. 3: Diagrama de tres dimensiones correspondiente a la arquitectura del modelo de análisis basado en el modelo de casos de uso.

En este caso las dimensiones de la arquitectura corresponden a las dimensiones usadas en el modelo de requisitos. Se manejan tres dimensiones para lograr una rastreabilidad con la arquitectura que se desarrollará en el modelo de análisis.

La arquitectura para el modelo de análisis será implementada mediante tres tipos o *estereotipos* de objetos como elementos básicos de desarrollo.

Clases con Estereotipos

El tipo de funcionalidad de un objeto dentro de una arquitectura se le conoce como su *estereotipo*. Para los sistemas de información la arquitectura del sistema en este modelo de análisis se basa en tres estereotipos básicos de objetos:

- El estereotipo *entidad* ("entity") para objetos que guarden información sobre el estado interno del sistema, a corto y largo plazo, correspondiente al dominio del problema. Todo comportamiento naturalmente acoplado con esta información también se incluye en los objetos entidad.
- El estereotipo *interface* o *borde* ("boundary") para objetos que implementen la presentación o vista correspondiente a las bordes del sistema hacia el mundo externo, para todo tipo de actores, no sólo usuarios humanos.
- El estereotipo *control* ("control") para objetos que implementen el comportamiento o control especificando cuándo y cómo el sistema cambia de estado, correspondiente a los casos de uso. Los objetos control modelan funcionalidad que no se liga con ningún otro tipo de objeto, como el comportamiento que opera en varios objetos entidad a la vez.

No hay restricciones en cuanto a los diferentes estereotipos que puedan utilizarse. La notación de UML para un estereotipo se muestra en la Figura 4.

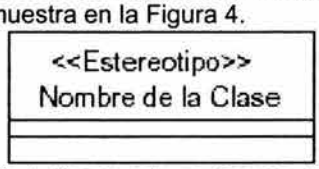


Fig. 4: Diagrama de una clase con estereotipo

Los tres estereotipos correspondientes a las tres dimensiones para la arquitectura del modelo de análisis se muestra en la Figura 5.

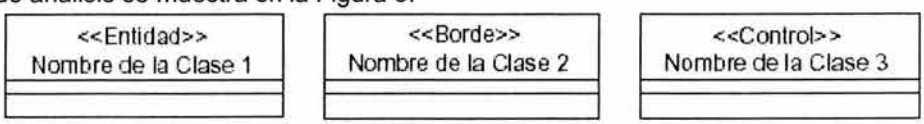


Fig. 5: Diagrama de clase para los tres estereotipo.

Los tres estereotipos se muestran como íconos en la Figura 6, es importante hacer notar que los estereotipos se deben insertar en inglés para que las herramientas CASE los reconozcan de tal manera.

Fig. 6: Diagrama de clase para los tres estereotipo.

Dado que hay interacción entre los diferentes tipos de objetos, existe cierto traslape en la funcionalidad que los objetos ofrecen; ese traslape debe minimizarse para asegurar una buena extensibilidad, donde cada tipo de objeto captura por lo menos dos de las tres dimensiones. No obstante cada uno de ellos tiene cierta inclinación hacia una de estas dos dimensiones, como se muestra en la Figura 7.

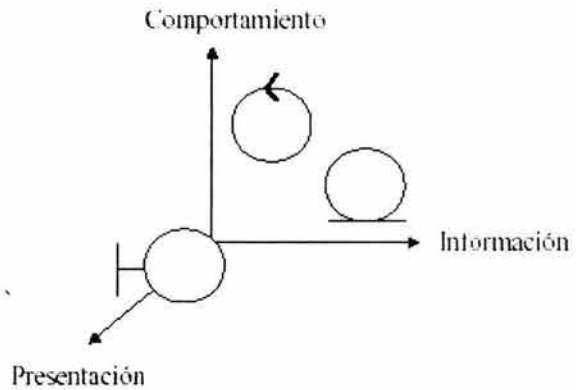


Fig. 7: Diagrama mostrando traslape en los estereotipos de los objetos.

4.1.1 Clases para Casos de Uso

En el desarrollo del modelo de análisis, en general se trabaja con un caso de uso a la vez. Para cada caso de uso se identifican los objetos necesarios para su implementación. Se identifican estos objetos según sus estereotipos para corresponder a la funcionalidad ofrecida en cada caso de uso. Se define explícitamente qué objeto es responsable de cual comportamiento dentro del caso de uso. En general, se toma un caso de uso y se comienza identificando los objetos borde necesarios, continuando con los objetos entidad y finalmente los objetos control. Este proceso se continúa a los demás casos de uso.

Dado que los objetos son "ortogonales" a los casos de uso, en el sentido de que un objeto puede participar en varios casos de uso, este proceso es iterativo; esto significa que cuando un conjunto de objetos ya existe, estos pueden modificarse para ajustarse al nuevo caso de uso.

La meta es formar una arquitectura lo más estable posible, reutilizando el mayor número de objetos posible. De tal manera, la descripción original de los casos de uso se transforma a una descripción en base a los tres tipos de objetos, como se ve en la Figura 8.

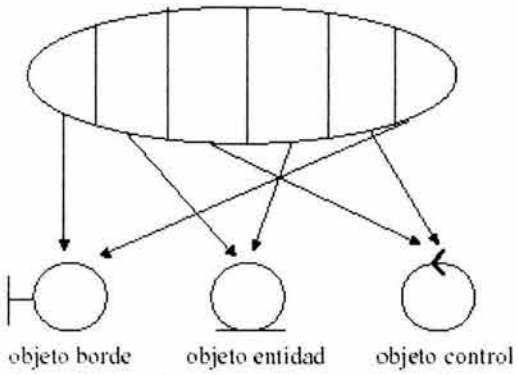


Fig. 8: La funcionalidad de cada caso de uso es asignada a objetos distintos y de acuerdo a los estereotipos de dichos objetos.

Se parte el caso de uso de acuerdo a los siguientes principios:

- La funcionalidad de los casos de uso que depende directamente de la interacción del sistema con el mundo externo se asigna a los objetos borde.
- La funcionalidad relacionada con el almacenamiento y manejo de información del dominio del problema se asigna a los objetos entidad.
- La funcionalidad específica a uno o varios casos de uso y que no se ponen naturalmente en ningún objeto borde o entidad se asigna a los objetos control. Típicamente se asigna a un sólo objeto control y si éste se vuelve muy complejo se asignan objetos control adicionales.

4.1.1.1 Identificación de Clases según Estereotipos

Para llevar a cabo la transición del modelo de requisitos al modelo de análisis se deben identificar los objetos necesarios para implementar todos los casos de uso. La arquitectura de objetos debe considerar los tres tipos de estereotipos de objetos ya mencionados.

Para lograr esto se debe identificar primero las clases borde, luego las entidad y finalmente las de control. En general, se desea asignar la funcionalidad más especializada correspondiente a la "política" de la aplicación a los objetos control, la cual depende y afecta al resto de los objetos. Por otro lado, los objetos de borde deben contener funcionalidad más bien local limitando su efecto en los demás objetos. Por tanto, una de las principales tareas es distribuir lo mejor posible el comportamiento especificado en el modelo de requisitos en los diferentes tipos de objetos de la arquitectura de análisis.

La asignación de funcionalidad es bastante difícil en la práctica afectando de gran manera la robustez y mantenimiento del sistema. En general, los cambios más comunes a un sistema son los cambios en su funcionalidad y bordes. Los cambios a los bordes deben afectar típicamente solo los objetos borde. Cambios a la funcionalidad son más difíciles, ya que la funcionalidad puede abarcar todos los tipos de objetos. Si la funcionalidad esta ligada a la información existente, tales cambios afectan al objeto entidad que representa esa información, o puede involucrar múltiples objetos incluyendo objetos control. En general, esta funcionalidad se define en uno o varios casos de uso y se asigna a uno o varios objetos control. A continuación se analiza el proceso de identificación de los tres tipos.

4.1.1.1.1 Borde

Toda la funcionalidad especificada en las descripciones de los casos de uso que depende directamente de los aspectos externos del sistema se ubica en los objetos de borde.

Es a través de estos objetos que se comunican los actores con el sistema. La tarea de una clase borde es traducir los eventos generados por un actor en eventos comprendidos por el sistema, y traducir los eventos del sistema a una presentación comprensible por el actor. Las clases borde, en otras palabras, describen comunicación bidireccional entre el sistema y los

actores. Las clases borde son bastante fáciles de identificar, donde se cuenta con al menos tres estrategias:

- Se pueden identificar en base a los actores.
- Se pueden identificar en base a las descripciones de las clases borde del sistema que acompañan al modelo de requisitos.
- Se pueden identificar en base a las descripciones de los casos de uso y extraer la funcionalidad que es específica a los bordes.

Si se maneja la estrategia correspondiente a los actores. Cada actor concreto necesita su propia clase borde para su comunicación con el sistema. En muchos casos un actor puede necesitar de varios objetos borde. Es evidente que los objetos borde no son totalmente independientes de cada uno ya que deben saber de la existencia de los demás para poder resolver ciertas tareas.

Una vez identificado los objetos borde es más fácil modificar posteriormente las clases borde de un sistema. Al tener todo lo relacionado a una clase borde en un objeto, cada cambio al borde será local a ese objeto. Como los cambios a las bordes son muy comunes, es vital que estos sean extensibles. Existen dos tipos diferentes de bordes a modelar, bordes a otros sistemas y bordes a los usuarios humanos.

- *Bordes con otros sistemas*, es muy común que la comunicación se describa mediante protocolos de comunicación. Los objetos borde pueden traducir las salidas del sistema a un protocolo de comunicación estandarizado, o simplemente enviar eventos producidos internamente sin conversiones complejas. Su principal ventaja es que si se cambia el protocolo, estos cambios serán locales al objeto borde. Un mayor problema ocurre cuando existen señales continuas del mundo externo, entonces los objetos borde deben muestrear la señal de entrada, o interrumpir cuando ciertos valores exceden un valor umbral, ya que internamente en el sistema sólo existe comunicación discreta mediante eventos. Los objetos borde deben entonces traducir la información continua a información discreta. Problemas de cuantificación pueden aparecer y deben ser resueltos.
- *Bordes con usuarios humanos*, los objetos borde pueden ser complejos para modelar. Existen muchas técnicas diferentes para un buen diseño de bordes, como el diseño de Interfaces Gráficas de Usuario (GUI-Graphical User Interface), Sistemas de Manejo de Ventanas de Usuario (UIMS-User Interface Management Systems) y sistemas de Interface de Programación de Aplicación (API). Es fundamental que el usuario tenga una imagen lógica y coherente del sistema. En las aplicaciones interactivas es común que la borde de usuario sea una parte mayor de la aplicación completa.

Aunque cada tipo de objeto tiene un propósito distinto, es evidente que los objetos borde tienen como propósito principal las presentaciones. A pesar de esto, también pueden manejar información y tener comportamiento. Cuánta información y comportamiento debe ligarse a un objeto borde debe decidirse de manera individual. En un extremo, el objeto borde solo envía el evento que recibe del actor a otros objetos en el sistema, sin participar activamente en el curso de eventos. En el otro extremo, el comportamiento del objeto borde es muy complejo donde la información se integra en el objeto borde y puede funcionar casi independiente de otros objetos.

El potencial para cambios debe afectar la decisión de qué comportamiento en el caso de uso debe ligarse a un objeto borde particular; Cualquier cambio en la funcionalidad directamente ligada a la borde debe ser local al objeto borde, mientras que otros cambios no deben afectarlo; esto implica que debe aprenderse y aplicarse en todas las actividades del modelado. Para identificar qué parte del flujo de un caso de uso debe asignarse a los objetos borde, se debe analizar las interacciones entre los actores y los casos de uso.

Esto significa buscar aspectos con una o más de las siguientes características:

- Presentación de información al actor que requiera información de regreso.
- Funcionalidad que cambie si cambia el comportamiento del actor.
- Flujo de acción que dependa de un tipo de borde particular.

A continuación se describen las clases borde necesarias para cada caso de uso de acuerdo a la documentación generada durante el modelo de requisitos. Se requieren todas las pantallas con

las cuales los casos de uso se relacionan. Es importante mencionar que a pesar de que existen múltiples ligas entre pantallas para simplificar la navegación, sólo se identifican como parte del caso de uso aquellas que se consideran "esenciales" para la ejecución del caso de uso.

Caso de uso	Actores	Clases Bordes
Validar Cliente	*Usuario, Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaValidación, InterfaceBaseDatosCliente*
Ofrecer Servicios	*Usuario, Administrador, Coordinador*	*InterfaceCliente, InterfaceUsuario, PantallaOfrecerServicio*
Validar Usuario	*Usuario, Base de Datos*	*InterfaceUsuario, PantallaValidaCredencial, InterfaceBaseDatosCliente*
Consultar Acervo	*Usuario, Base de Datos*	*InterfaceCliente, PantallaConsultaAcervo, PantallaResultadoAcervo, InterfaceBaseDatosAcervo*
Inscripción	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaInscripción, InterfaceBaseDatosCliente*
Administrar Usuario	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, InterfaceBaseDatosCliente*
Imprimir Credencial	*Cliente, Usuario, Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaImprimirCredencial, InterfaceBaseDatosCliente*
Administrar Acervo	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, InterfaceBaseDatosAcervo,*
Administrar Cliente	*Administrador, Base de Datos*	*InterfaceCliente, PantallaObtenerRegistroCliente, PantallaDatosCliente, PantallaBuscarRegistroCliente, InterfaceBaseDatosCliente*
Generar Reporte Usuario	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuario, PantallaReporteGuardarUsuario, InterfaceBaseDatosCliente*
Administrar Usuario por Área	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaGuardarUsuarioArea, InterfaceBaseDatosCliente*
Generar Reporte Acervo	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervo, PantallaReporteGuardarAcervo, PantallaReporteAdquisiciones, PantallaFichaDescriptiva InterfaceBaseDatosAcervo*
Búsquedas Generales Acervo	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaBusquedaGeneralAcervo, PantallaBusquedaGeneralAcervoGuardar, InterfaceBaseDatosAcervo*
Búsqueda General Usuario	*Administrador, Coordinador, Base de Datos*	*InterfaceCliente, PantallaBusquedaGeneralUsuario, PantallaBusquedaGeneralUsuarioGuardar InterfaceBaseDatosCliente*

4.1.1.1.2 Entidad

Se utilizan objetos entidad para modelar la información que el sistema debe manejar a corto y largo plazo. La información a corto plazo existe durante la ejecución del caso de uso, mientras que la información a largo plazo sobrevive a los casos de uso, por lo cual es necesario guardar esta información en alguna base de datos. Además, se debe incluir el comportamiento para manejar la propia información local al objeto entidad.

Los objetos entidad se identifican en los casos de uso, donde la mayoría se identifican del modelo del dominio del problema en el modelo de requisitos. Objetos entidad adicionales pueden ser más difíciles de encontrar. Es muy común que se identifiquen muchos objetos entidad, aunque se debe limitar estos objetos a los realmente necesarios para la aplicación, siendo esto lo más difícil del proceso de identificación. Es por lo tanto esencial trabajar de forma organizada cuando se modelan los objetos entidad. Las necesidades de los casos de uso deben ser las guías y solamente aquellos objetos entidad que puedan justificarse de la descripción del caso de uso deben ser incluidos.

Para definir si la información debe ser modelada como un objeto entidad o como un atributo, dependerá de cómo se use la información, si se maneja de forma separada, debe modelarse como un objeto entidad, mientras que la información que esta acoplada fuertemente a alguna otra información y nunca se usa por si misma debe modelarse como un atributo de un objeto entidad. Todo depende de cómo los casos de uso manejen la información. Cierta información puede modelarse como objeto entidad en un sistema, mientras que en otro sistema puede ser un atributo. También se debe identificar qué operaciones y cuales atributos serán incluidos dentro de los objetos entidad.

Dado que la única forma para manipular un objeto entidad es por medio de sus operaciones, las operaciones identificadas deben ser suficientes para manipular completamente al objeto entidad. La descripción detallada de los casos de uso es de nuevo un medio extremadamente valioso para encontrar las operaciones deseadas. El flujo completo de eventos que se describe en los casos de uso, permite extraer las operaciones que conciernen a los objetos entidad.

Las operaciones asignadas a los objetos entidad pueden ser más o menos complejas. En el caso menos complejo un objeto entidad consta sólo de operaciones de acceso a los valores de los atributos. En el caso más complejo un objeto entidad puede tener flujos de eventos más allá de simples accesos a los valores de los atributos. Sea cual sea la complejidad de estas operaciones, el objetivo es que éstas sólo dependan y afecten información local. La siguiente es una lista de las operaciones típicas que deben ser ofrecidas por un objeto entidad:

- Guardar y traer información
- Comportamiento que debe modificarse si el objeto entidad cambia
- Crear y remover el objeto entidad

Dada la complejidad de obtener operaciones este aspecto se analiza en la etapa de diseño. Durante la identificación de objetos entidad, se observa que objetos similares aparecen en varios casos de uso; por ello, es necesario verificar si deben ser los mismos objetos entidad o si pueden haber objetos entidad separados. Incluso si los casos de uso no interactúan de la misma manera sobre los objetos, el objeto entidad puede ofrecer operaciones que satisfagan las necesidades de diversos casos de uso. Si se considera que dos objetos entidad representan un mismo objeto, las operaciones, atributos y asociaciones también tienen que integrarse. De manera similar, se puede hacer una identificación preliminar de los atributos, aunque esto se aborda en el modelo de diseño.

Se describe por tanto las clases entidad necesarias para cada caso de uso de acuerdo a la documentación generada durante el modelo de requisitos. Se observa que las clases son obtenidas del dominio del problema generado en el modelo de requisitos. Si fueran necesarias nuevas clases entidad habría que modificar el dominio del problema anterior.

Caso de uso	Clases Entidad
Validar Cliente	RegistroCliente
Ofrecer Servicios	
Validar Usuario	RegistroUsuario
Consultar Acervo	RegistroAcervo
Inscripción	RegistroUsuario
Administrar Usuario	RegistroUsuario

Administrar Acervo	RegistroAcervo
Administrar Cliente	RegistroCliente
Generar Reporte Usuario	RegistroUsuario
Administrar Usuario por área	UsuarioPorArea
Generar Reporte Acervo	RegistroAcervo
Búsquedas Generales Acervo	RegistroAcervo
Búsquedas Generales Usuario	RegistroUsuario

4.1.1.1.3 Control

En la mayoría de los casos de uso, existe un comportamiento que no se puede asignar de forma natural a ninguno de los otros dos tipos de objetos, ya que realmente no pertenece de manera natural a ninguno de ellos. Una posibilidad es repartir el comportamiento entre los dos tipos de objetos (de borde y de entidad), como lo sugieren algunos métodos, pero la solución no es buena si se considera el aspecto de extensibilidad. Un cambio en el comportamiento podría afectar varios objetos dificultando su modificación. Por lo tanto, para evitar estos problemas tal comportamiento se asigna en *objetos control.*

Los objetos de control típicamente actúan como "pegamento" entre los otros tipos de objetos y por lo tanto proveen la comunicación entre los demás tipos de objetos. Son típicamente los más efímeros de todos los tipos de objetos, dependiendo de la existencia del propio caso de uso.

Los objetos control se identifican directamente de los casos de uso. Como primera aproximación, se asigna un objeto control a cada caso de uso, concreto y abstracto. Dado que se asigna inicialmente el comportamiento a los objetos borde y entidad para cada caso de uso, el comportamiento restante se asigna a los objetos control.

En general, la manera de asignar el comportamiento es modelar inicialmente el caso de uso sin ningún objeto control, o sea sólo utilizar objetos borde y objetos entidad. Cuando tal modelo se ha desarrollado, se verá que hay ciertos comportamientos que no se asignan de forma natural, ni en los objetos entidad ni en los objetos borde. Estos comportamientos deben ubicarse en los objetos control. Sin embargo, puede darse la situación donde no queda comportamiento restante para modelar en el caso de uso, entonces, no se necesita un objeto control. Otra situación es si el comportamiento que queda, después de distribuir el comportamiento relevante entre objetos borde y entidad, es demasiado complicado, la funcionalidad puede ser dividida en varios objetos control.

Por otro lado, si un caso de uso se acopla a varios actores esto puede indicar que existen variados comportamientos en relación a los diferentes actores y por lo tanto deben asignarse varios objetos control. La meta debe ser ligar solo un actor con cada objeto control ya que los cambios en los sistemas a menudo son originados por los actores y de tal manera se logra modularizar los posibles cambios.

La estrategia de asignación de control se debe decidir según cada aplicación. En la mayoría de los casos, sin embargo, se promueve la separación del control de un caso de uso en un objeto control que delega funcionalidad de manejo más local a los otros dos tipos de objetos.

Objetos control identificados:

Caso de uso	Clases Control
Validar Cliente	ManejadorCliente
Ofrecer Servicios	ManejadorServicio
Validar Usuario	ManejadorUsuario
Consultar Acervo	ManejadorConsultaAcervo
Inscripción	ManejadorUsuario
Administrar Usuario	ManejadorUsuario
Imprimir Credencial	ManejadorUsuario
Administrar Acervo	ManejadorAcervo
Administrar Cliente	ManejadorCliente
Generar Reporte Usuario	ManejadorReporteUsuario

Administrar Usuario por Área	ManejadorUsuarioArea
Generar Reporte Acervo	ManejadorReporteAcervo
Búsqueda General Acervo	ManejadorBusquedaGeneralAcervo
Búsqueda General Usuario	ManejadorBusquedaGeneralUsuario

4.2 Diagramas de Secuencias

Ahora se describen los casos de uso del modelo de requisitos según la lógica que deberán presentar estas clases para lograr la funcionalidad descrita en los diversos casos de uso.

Con este paso y basado en la lógica propuesta para esta descripción, se define la arquitectura de análisis tanto estructural como funcional.

Dada la complejidad y la importancia de estas descripciones, es importante probar las secuencias funcionales de los flujos de los casos de uso, revisar qué tan bien la lógica y arquitectura de clase resuelve la funcionalidad establecida.

Por tanto, se emplea el concepto de *diagramas de secuencias*, *interacción* o *eventos*, los cuales describen como los diferentes casos de uso son implementados mediante los objetos de la arquitectura recién generados. Los diagramas correspondientes muestran la interacción entre los objetos participantes a nivel de eventos que se envían entre sí, excluyendo cualquier detalle interno de ellos.

El formato de un diagrama de secuencia se muestra en la Figura 9. Es importante observar que es un diagrama exclusivamente de objetos y no de clases. Sin embargo, los objetos son instancias de clases que al no tener ningún nombre particular se muestran a través del nombre de la clase subrayada y con el prefijo de ":", correspondiente a la notación para diagramas de objetos.

Cada clase participante se representa por una barra, dibujadas como líneas verticales en el diagrama. El orden entre las barras es insignificante y debe elegirse para dar la mayor claridad. Si existieran varias instancias de las clases que interactúan entre sí, se dibujarían las instancias como barras diferentes. Además de los objetos, es importante representar entidades externas al sistema en los diagramas de secuencia. De lo contrario este tipo de diagrama no es útil en el modelo de análisis. Para este caso, las entidades externas que se incluyen como barras adicionales en el diagrama representan *instancias* de los actores.

El eje de tiempo en el diagrama de secuencia es vertical (paralelo a las barras) y avanzando hacia abajo. El comienzo del diagrama de secuencia corresponde al inicio del caso de uso. El avance en el tiempo en el diagrama es controlado por los eventos, mientras que la distancia entre dos eventos en el diagrama no tiene relación con el tiempo real entre estos eventos. Más aún el tiempo no es necesariamente lineal en el diagrama, pudiendo existir concurrencia.

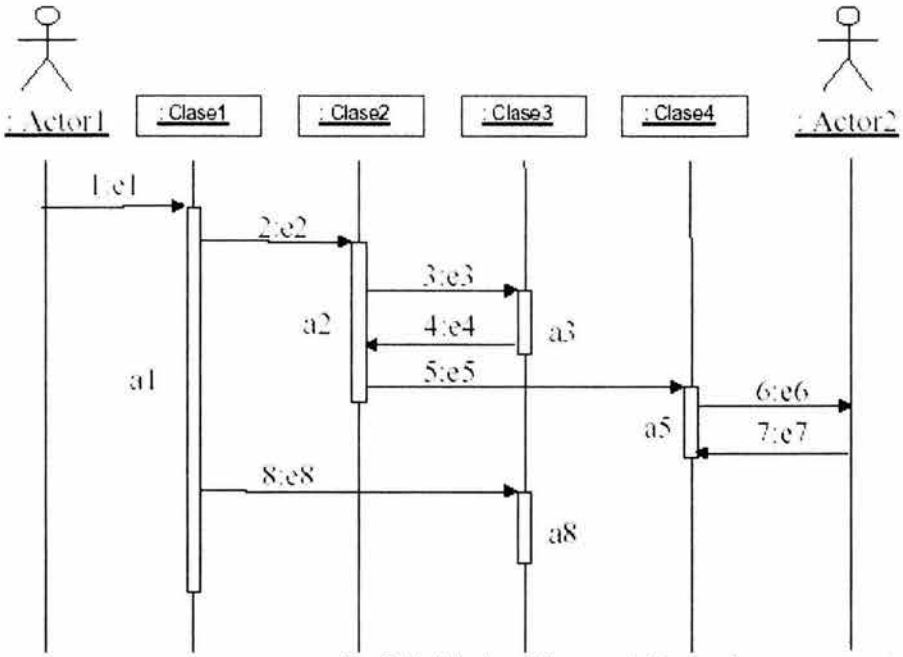


Fig. 9: Modelo de un Diagrama de Secuencia.

El diagrama de secuencia de la Figura 9 muestra un ejemplo de estos eventos.

En el diagrama de la Figura 9, las barras gruesas corresponden a *actividades* dentro del objeto, denominadas por *a*, mientras que las flechas corresponden a *eventos*, denominadas por *e*. Un evento se dibuja como una flecha horizontal que comienza en la barra correspondiente al objeto que lo envía y termina en la barra correspondiente al objeto que lo recibe.

Las actividades son iniciadas por el arribo de eventos y el tiempo que duran es sólo relevante en relación a eventos originados durante esa actividad.

Un aspecto importante que los diagramas de secuencia deben considerar es que las secuencias de eventos sean continuas y no existen interrupciones. Por otro lado la gran mayoría de los diagramas de secuencia, y por lo tanto los casos de uso, deben comenzar con un evento externo que se genera a partir de una de las instancias de los actores del sistema.

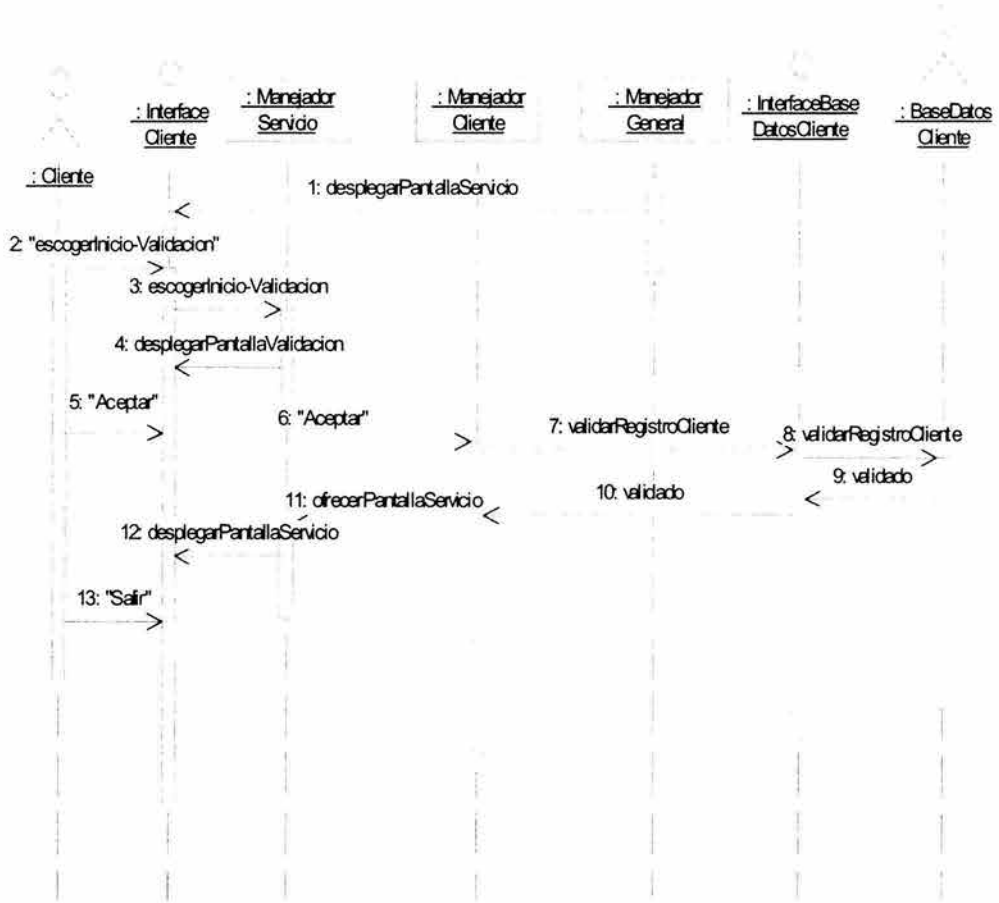
Cada nuevo evento debe generarse a partir de uno que se recibe con la única excepción de eventos iniciales que son generados por el sistema o típicamente por un actor. Se observa además, que los actores principales son los que típicamente deciden que casos de uso serán instanciados, a diferencia de los propios objetos y casos de uso secundarios que más bien responden a eventos que son recibidos.

Durante el modelo de análisis se utilizan los diagramas de secuencia para describir los *flujos principales* y *subflujos* para cada caso de uso. Esto ayuda a revisar si la lógica es correcta y consistente al relacionar los casos de uso del modelo de requisitos con la arquitectura de clases del modelo de análisis.

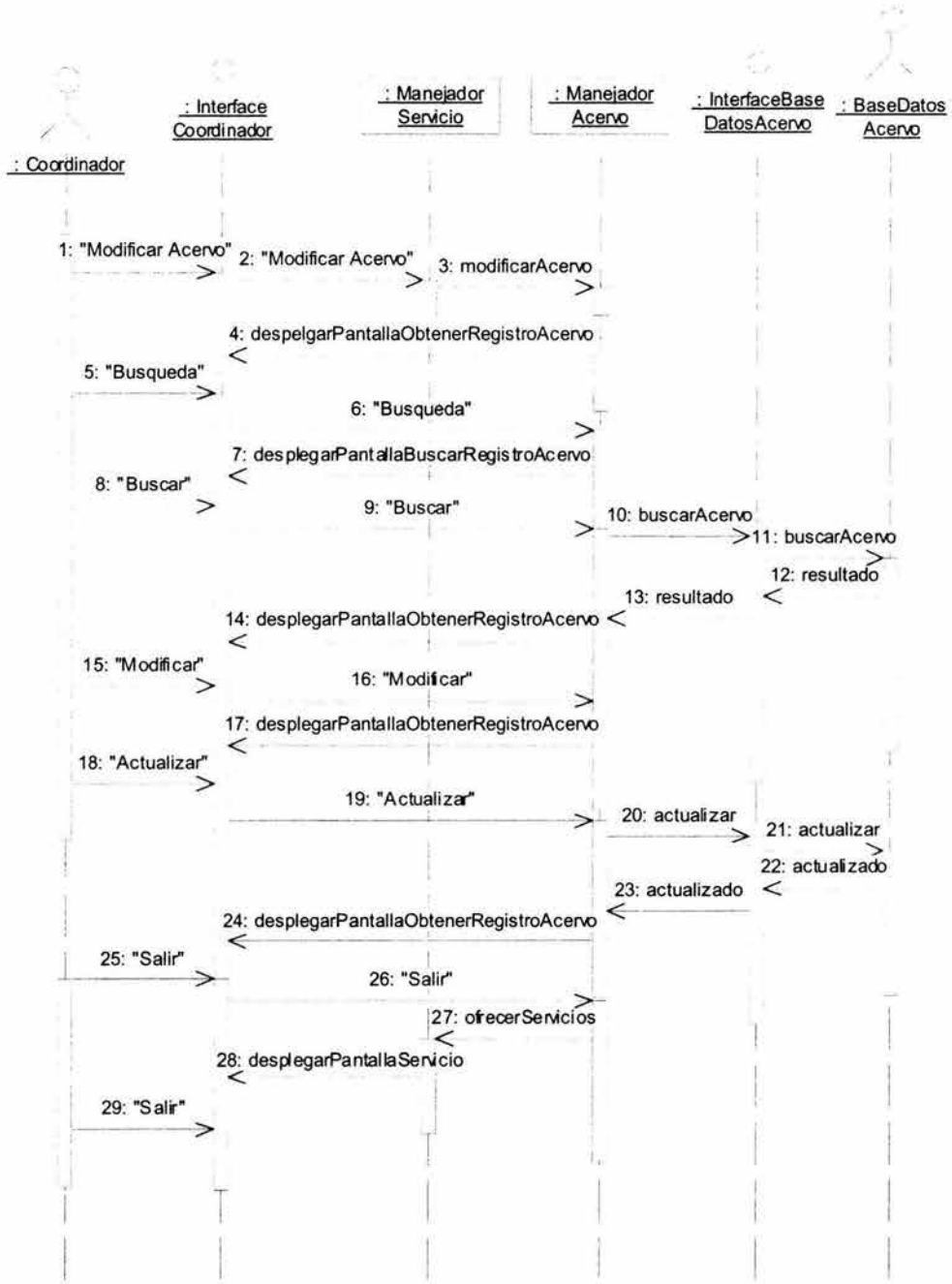
Dado que existen múltiples posibles flujos de secuencia, se define todo en los flujos principales que definan flujos completos y que incluyen subflujos intermedios, incluyendo casos de uso de inclusión. Por lo tanto, se describen los casos de uso de mayor interés. Además los inicios de muchos de estos casos se repiten, lo cual se incluye en los diversos casos de uso para no perder el flujo de eventos en el desarrollo de las diversas secuencias.

Se presentan los diagramas de secuencia para los casos de uso presentados en el capitulo Modelo de Requisitos.

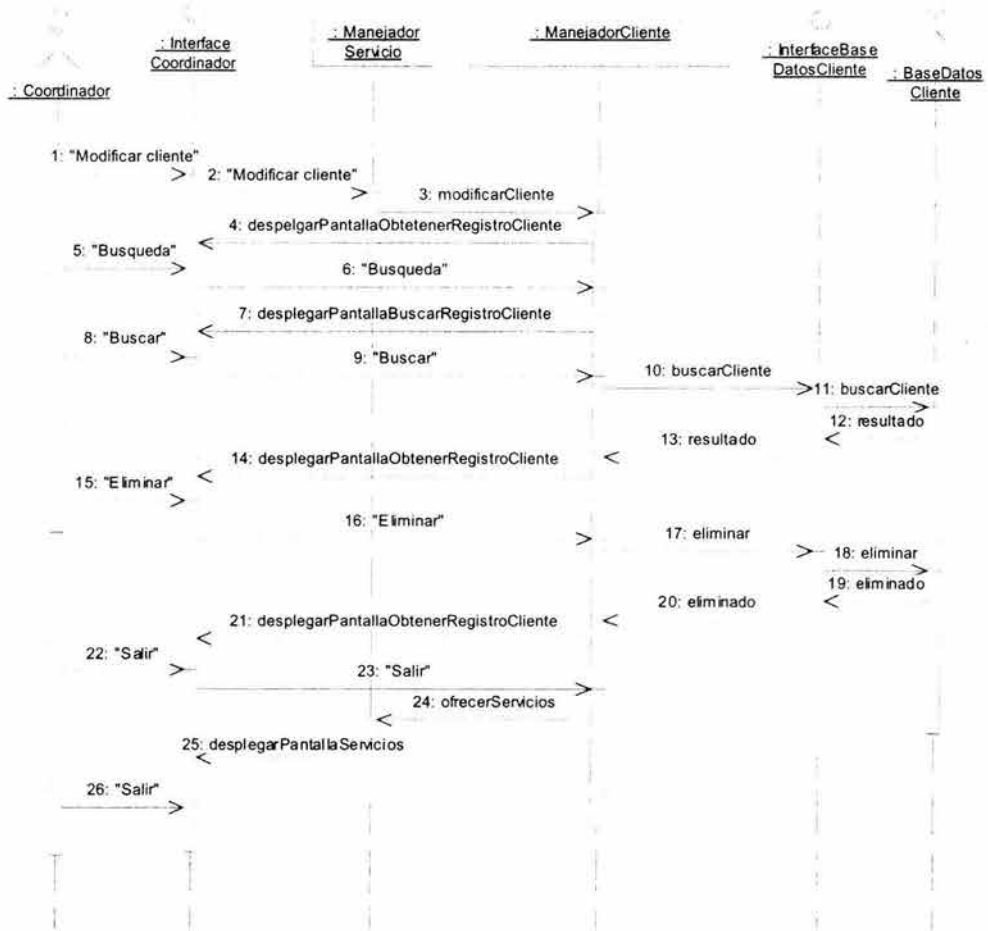
Validar cliente.



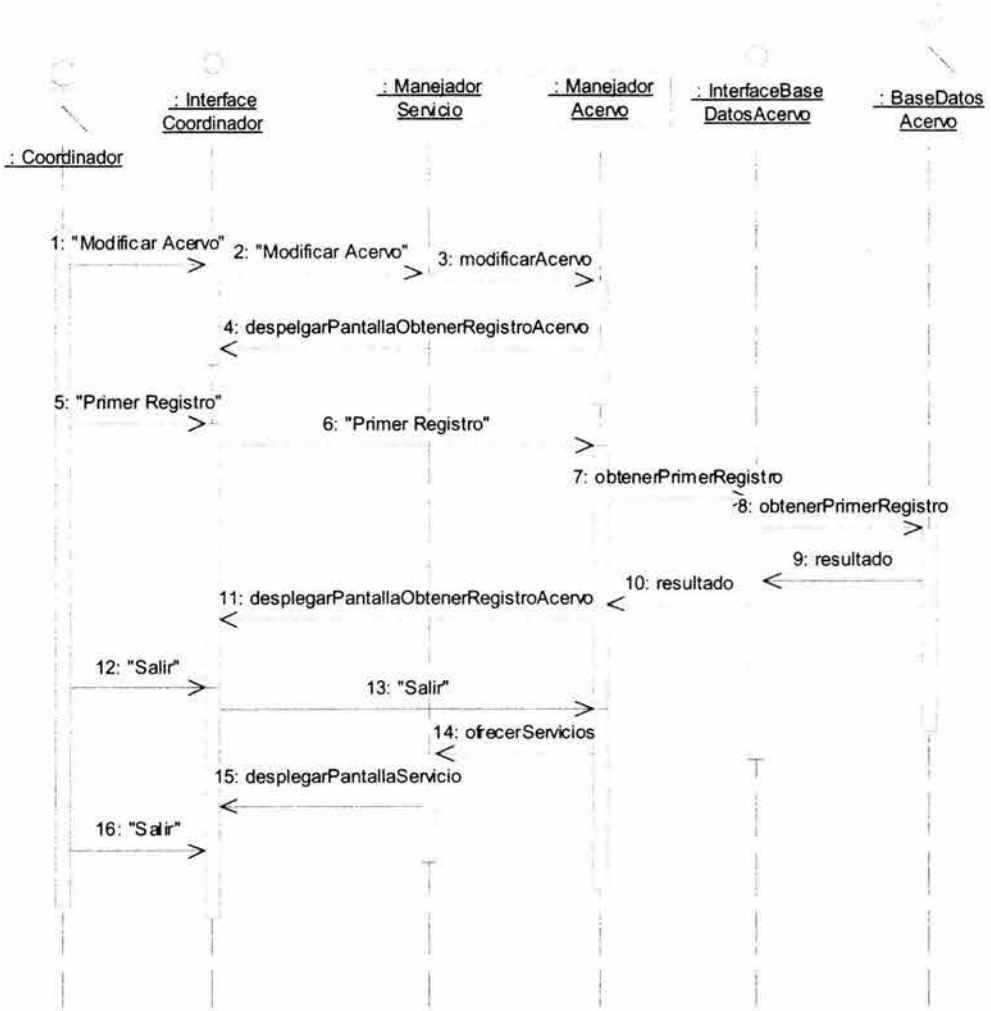
Administrar Acervo (Actualizar)



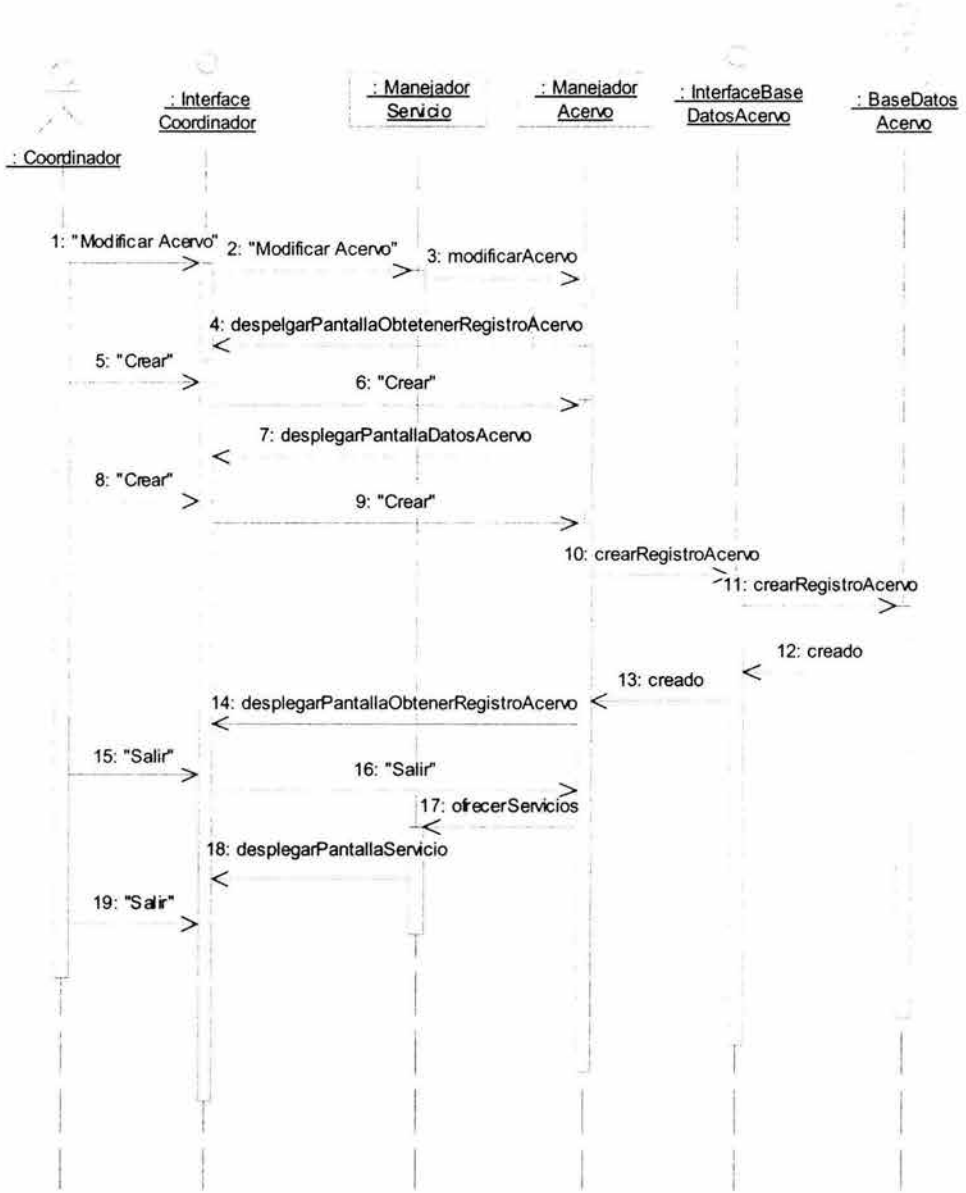
Administrar Acervo (Eliminar)



Administrar Acervo (Primer registro)



Administrar Acervo (Crear registro)



4.2.1 Documento de Casos de Uso

A partir de las diversas secuencias analizadas y descritas en los diagramas de secuencias, se puede generar una descripción casi completa de los casos de uso del sistema a nivel de análisis. Para lograrlo, se toman todas las descripciones y se insertan en los flujos o subflujos correspondientes de los diversos casos de uso. Dado que las secuencias no mencionan todos los posibles eventos sino los principales, se pueden completar los que sean necesarios. En particular se debe asegurar que no existan discontinuidades entre las secuencias de eventos.

A pesar de eso, se debe siempre mantener una buena consistencia entre los casos de uso de análisis y las secuencias anteriores que en el fondo son instancias a partir de los casos de uso. Cualquier cambio en la lógica en las secuencias o casos de uso se debe reflejar también en los diagramas de secuencia.

En las descripciones de los casos de uso, se subrayan las clases y se escribe en letras cursivas los nombres de los eventos entre clases. Las frases deben ser claras y concisas, ya que esto facilita el diseño.

Es conveniente señalar que para hacer una distinción de las tablas de la Base de Datos empleadas en el Sistema, se hará referencia a la Base y la tabla según el caso de uso es decir, cuando se maneje la leyenda BaseDatosCliente, se estará haciendo referencia a la Tabla RegistroCliente de la Base de Datos; con la leyenda BaseDatosUsuario, se hace referencia a la Tabla RegistroUsuario de la Base de Datos y, con la leyenda BaseDatosAcervo se hará referencia a la Tabla RegistroAcervo de la Base de Datos.

Caso de Uso:	**Validar Cliente**
Actores:	Usuario, Administrador, Coordinador, Base de Datos
Tipo:	Inclusión
Propósito:	Validar a un Cliente(Usuario, Administrador, Coordinador) ya registrado para el uso del SGI-ENEO
Resumen:	Este caso es iniciado por un Cliente, validándolo mediante un login (selección de una de las tres opciones presentadas: Coordinador, Administrador, Usuario) y password, que son comparados con los datos almacenados en los registros de la Tabla Registro Cliente de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente los casos de uso: Inscripción (Usuario), Administrar Cliente (Administrador, Coordinador)
Flujo Principal:	El ManejadorCliente solicita *desplegarPantallaValidacion* a la InterfaceCliente. La InterfaceCliente despliega la PantallaValidacion. La PantallaValidacion se *despliega*. El Cliente puede seleccionar entre las siguientes opciones: "Aceptar" y "Cancelar". Si la actividad seleccionada es "Aceptar" se valida el registro del Usuario mediante los datos insertados (una de las tres opciones y su contraseña) por el Cliente en la PantallaValidacion. La PantallaValidacion envía el evento "Aceptar" a la InterfaceCliente. La InterfaceCliente envía el evento "Aceptar" al ManejadorCliente. El ManejadorCliente solicita *validarRegistroCliente* a la InterfaceBaseDatosCliente. La InterfaceBaseDatosCliente solicita *validarRegistroCliente* a la BaseDatosCliente. La BaseDatosCliente valida al Cliente y devuelve *validado* a la InterfaceBaseDatosCliente. La InterfaceBaseDatosCliente devuelve *validado* al ManejadorCliente. Una vez validado el Usuario (Ex–1), el ManejadorCliente solicita *ofrecerServicios* al ManejadorServicio. Se continua con el caso de uso Ofrecer Servicios y dependiendo del tipo de Cliente, el subflujo será S–2 (Usuario), S-3(Administrador), S–4(Coordinador). Si la actividad seleccionada es "Salir", la PantallaValidacion envía el evento "Salir" a la InterfaceCliente. La InterfaceCliente envía el evento "Salir" al ManejadorCliente. El ManejadorCliente solicita *ofrecerServicios* al ManejadorServicio se continua con el caso de uso *Ofrecer Servicios* flujo principal.

Subflujos:	Ninguno
Excepciones:	EX –1 no hubo validación, la contraseña no corresponde, se cancelan los datos proporcionados y se manda un mensaje de error, y enseguida se solicita al Usuario volver a validarse.

Caso de uso:	**Administrar Acervo**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Permite a un Administrador ó Coordinador modificar un registro del material contenido en el Acervo para actualizarlo.
Resumen:	Este caso es iniciado por un Administrador ó Coordinador. Ellos son los que crean el registro del material en la Base de Datos Acervo.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Validar Cliente*.
Flujo principal:	El ManejadorServicio solicita *modificarAcervo* al ManejadorAcervo. Se continuara con el subflujo *AdministrarRegistroAcervo* (S -1)
Subflujos:	S–1 *Administrar Registro Acervo.* El ManejadorAcervo solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfaceCoordinador. La InterfaceCoordinador *despliega* a la PantallaObtenerRegistroAcervo. PantallaObtenerRegistroAcervo se despliega. En los campos de la pantalla se recuperan automáticamente los datos del Primer Registro de la Tabla RegistroAcervo de la Base de Datos. El Administrador ó Coordinador podrá seleccionar entra las siguientes actividades: "Eliminar", "Modificar", "Actualizar"(inactivo hasta que se haga clic en "Modificar"), "Búsqueda", "Primer Registro", "Último Registro", "Siguiente", "Anterior", "Crear" y "Salir". Si el Coordinador ó Administrador presiona "Modificar", se continua con el subflujo *Actualizar RegistroAcervo* (S–2). Si el Coordinador ó Administrador selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Acervo* (*S*-3). Si el Coordinador ó Administrador selecciona "Búsqueda" se ejecuta el subflujo *Búsqueda Registro Acervo*(*S*-4). Si el Coordinador ó Administrador selecciona "Primer Registro" se ejecuta el subflujo *Recupera Primer Registro* (*S*-5). Si el Coordinador ó Administrador selecciona "Último Registro" se ejecuta el subflujo *Recupera Último Registro* (*S*-6). Si el Coordinador ó Administrador selecciona "Siguiente" se ejecuta el subflujo *Recupera Registro Siguiente* (*S*-7). Si el Coordinador ó Administrador selecciona "Anterior" se ejecuta el subflujo *Recupera Registro Anterior* (*S*-8). Si el Coordinador ó Administrador selecciona "Crear" se ejecuta el subflujo *Crear Registro Acervo* (*S*-9). Si la actividad seleccionada es "Salir" la PantallaObtenerRegistroAcervo envía el evento *"Salir"* a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Salir" al ManejadorAcervo. El ManejadorAcervo solicita *ofrecerServicios* al ManejadorServicio. (Si aún no se ha presionado "Actualizar", la nueva información se perderá). Se continua con el caso de uso *ofrecer servicios* Flujo principal.

	S-2 Actualizar Registro Acervo El evento "Modificar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Modificar" al ManejadorAcervo. El ManejadorAcervo solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfaceCoordinador. La InterfaceCoordinador *despliega* a la PantallaObtenerRegistroAcervo. La PantallaObtenerRegistroAcervo se despliega. Se activa el botón "Actualizar", de la PantallaObtenerRegistroAcervo. Y el Coordinador introduce los nuevos datos, enseguida da clic en "Actualizar" el evento "Actualizar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador (E–1). La InterfaceCoordinador envía el evento "Actualizar" al ManejadorAcervo. El ManejadorAcervo solicita *actualizar* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *actualizar* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *actualizado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *actualizado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*.
	S-3 Eliminar Registro Acervo El evento "Eliminar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Eliminar" al ManejadorAcervo El ManejadorAcervo solicita *eliminar* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *eliminar* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *eliminado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *eliminado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*.
	S – 4 *Búsqueda registro.* El evento "Búsqueda" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Búsqueda" al ManejadorAcervo. El ManejadorAcervo solicita *desplegarPantallaBuscarRegistroAcervo* a la InterfaceCoordinador. La InterfaceCoordinador *despliega* a la PantallaBuscarRegistroAcervo. La PantallaBuscarRegistroAcervo se despliega. En la PantallaBuscarRegistroAcervo se pide los datos del registro a buscar. Se presenta las siguientes opciones: "Buscar" "Cancelar". Si la actividad seleccionada es "Buscar". El evento "Buscar" es enviado por la PantallaBuscarRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Buscar" al ManejadorAcervo El ManejadorAcervo solicita b*uscarAcervo* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *buscarAcervo* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve (E-2) *resultado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*. Si la actividad seleccionada es "Cancelar", el evento "Cancelar" es enviado por la PantallaBuscarRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Cancelar" al ManejadorAcervo. Se continua con el subflujo *Administrar Registro Acervo* subflujo (S–1).

	S – 5 *Recuperar primer registro.* El evento "Primer Registro" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Primer Registro" al ManejadorAcervo El ManejadorAcervo solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *obtenerPrimerRegistro* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*.
	S – 6 *Recuperar último registro.* El evento "Ultimo Registro" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Ultimo Registro" al ManejadorAcervo El ManejadorAcervo solicita *obtenerUltimoRegistro* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *obtenerUltimoRegistro* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*.
	S – 7 *Recuperar registro siguiente.* El evento "Registro Siguiente" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Registro Siguiente" al ManejadorAcervo El ManejadorAcervo solicita *obtenerRegistroSiguiente* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *obtenerRegistroSiguiente* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*.
	S – 8 *Recuperar registro anterior.* El evento "Registro Anterior" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Registro Anterior" al ManejadorAcervo El ManejadorAcervo solicita *obtenerRegistroAnterior* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *obtenerRegistroAnterior* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*.

	S – 9 Crear Registro. El evento "Crear" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Crear" al ManejadorAcervo. El ManejadorAcervo solicita *desplegarPantallaDatosAcervo* a la InterfaceCoordinador. La InterfaceCoordinador *despliega* a la PantallaDatosAcervo. La PantallaDatosAcervo se despliega. En la PantallaDatosAcervo se pide los datos del registro a buscar. Se presenta las siguientes opciones: "Crear Registro" "Cancelar". Si la actividad seleccionada es "Crear Registro" El evento "Crear Registro" es enviado por la PantallaDatosAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Crear Registro" al ManejadorAcervo El ManejadorAcervo solicita *crearRegistroAcervo* a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo solicita *crearRegistroAcervo* a la BaseDatosAcervo. La BaseDatosAcervo devuelve *creado* (E – 1) a la InterfaceBaseDatosAcervo. La InterfaceBaseDatosAcervo devuelve *creado* al ManejadorAcervo. Se continúa con el subflujo *Administrar Registro Acervo (S – 1)*. Si la actividad seleccionada es "Cancelar", el evento "Cancelar" es enviado por la PantallaDatosAcervo a la InterfaceCoordinador. La InterfaceCoordinador envía el evento "Cancelar" al ManejadorAcervo. Se continua con el subflujo *Administrar Registro Acervo* subflujo (S–1).
Excepciones:	E–1 *información incompleta.* Falta llenar información en el registro de Usuario. Se vuelve a solicitar al Usuario que complete el registro. E-2 *información no encontrada.* Los datos del registro solicitado no son iguales a ningún registro alojado en la Base de Datos Acervo.

CAPÍTULO 5
MODELO DE DISEÑO

5.1 MODELO DE DISEÑO

El Modelo de Diseño es un refinamiento y formalización adicional del *Modelo de Análisis* donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del Modelo de Diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos.

Se requiere un Modelo de Diseño ya que el *Modelo de Análisis* no es lo suficientemente formal para poder llegar al código fuente. Por tal motivo se debe refinar los objetos, incluyendo las operaciones que se deben ofrecer, la comunicación entre los diferentes objetos, los eventos que los objetos envían entre si, etc. Se desea también validar los resultados del análisis. Según el sistema crece y se formaliza, se verá qué tan bien los M*odelos de Requisitos y Análisis* describen al sistema. Durante el diseño, se puede ver si los resultados del análisis son apropiados para su implementación. Si se descubre aspectos que no están claros en alguno de los modelos anteriores, estos deben ser clarificados, quizás regresando a etapas anteriores.

Aunque esto pudiera verse como deficiencias del resultado de las fases anteriores que deben ser clarificadas aquí, esto sería una visión incorrecta de las diferentes etapas del desarrollo, ya que el propósito de los M*odelos de Requisitos y Análisis* es comprender el sistema y darle una buena estructura. También es importante comprender que las consideraciones tomadas en cuenta durante el diseño deben influir en la estructura del sistema lo menos posible. Es la propia aplicación la que controla la estructura, no las circunstancias de su implementación. Se considera el Modelo de Diseño como una formalización del espacio de análisis, extendiéndolo para incluir una dimensión adicional correspondiente al ambiente de implementación, como se puede ver en el diagrama de la Figura 1.

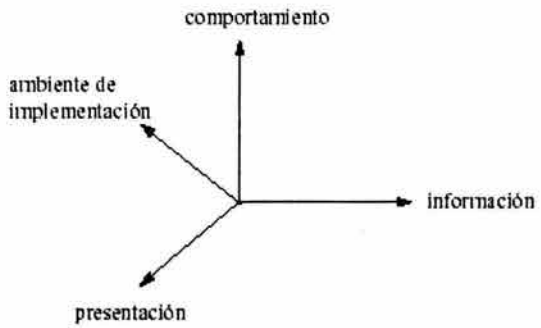


Figura1. El diseño añade el ambiente de implementación como un nuevo eje de desarrollo.

Esta nueva dimensión correspondiente al ambiente de implementación debe considerarse al mismo tiempo que el propio modelo es refinado. La meta es refinarlo hasta que sea fácil escribir código fuente. Como el *Modelo de Análisis* define la arquitectura general del sistema, se busca obtener una arquitectura detallada como resultado del Modelo de Diseño, de manera que haya una continuidad de refinamiento entre los dos modelos, como se puede ver en el diagrama de la Figura 2. En particular se puede apreciar el cambio en los modelos a partir de la introducción del ambiente de implementación.

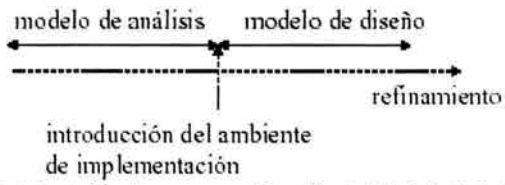


Figura 2. El Modelo de Diseño es una continuación del *Modelo de Análisis*.

La transición de análisis a diseño debe decidirse por separado para cada aplicación particular. De esta manera se puede ver el Modelo de Diseño como una especialización del *Modelo de Análisis* según un ambiente de implementación específico.

Si un cambio en el Modelo de Diseño proviene de un cambio en la lógica del sistema, entonces tales cambios deben hacerse en el *Modelo de Análisis*. Sin embargo, si el cambio es una consecuencia de la implementación, entonces tales cambios no deben ser incorporados en el *Modelo de Análisis*.

Las estructuras con las cuales se trabaja en el Modelo de Diseño son básicamente las mismas que en el *Modelo de Análisis*. Sin embargo, el punto de vista cambia, ya que se toma un paso hacia la implementación. El *Modelo de Análisis* debe ser visto como un modelo conceptual y lógico del sistema, mientras que el Modelo de Diseño debe acercarse al código fuente.

En cierta manera este enfoque es una extensión del concepto de la separación de la "política" de la implementación, donde la política fue definida durante el *Modelo de Análisis* y el Diseño tiene la responsabilidad mantener esta separación durante el diseño de métodos, aquellos que sirven para tomar decisiones (control) y aquellos que no (interface y entidad). Para llevar a cabo estos objetivos, se considera el *diseño de objetos*:

Diseño de Objetos. Se refina y formaliza el modelo para generar especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. Se describe cómo interaccionan los objetos en cada caso de uso específico, especificando qué debe hacer cada operación en cada objeto. Este paso genera las interfaces de los objetos, las cuales deben ser luego implementadas mediante métodos.

5.1.1 Diseño de Objetos

El diseño de objetos es un proceso de añadir detalles al análisis y tomar decisiones junto con diseño de sistema, o sea al ambiente de implementación, de manera que podamos lograr una especificación detallada antes de comenzar la implementación final. Algunos de los aspectos a ser resueltos diseño de objetos son determinar cómo las clases, atributos y asociaciones del *Modelo de Análisis* deben implementarse en estructuras de datos especificas. También es necesario determinar si se requiere introducir nuevas clases en el Modelo de Diseño para los cuales no se tiene ninguna representación en el *Modelo de Análisis* o si se requiere modificar o eliminar clases identificadas durante el *Modelo de Análisis*. Se debe agregar herencia para incrementar el reuso del sistema. Para el diseño de objeto se seguirá el *diseño por responsabilidades* (*RDD - Responsibility-Driven Design*). Este diseño está basado en un modelo *cliente-servidor* donde las clases son vistas como clientes cuando generan alguna petición hacia otra clase y como servidores cuando reciben peticiones de alguna otra clase. De tal manera, una misma clase puede ser vista en distintos momentos como cliente o servidor.

La funcionalidad ofrecida por las clases servidores se define en término de sus *responsabilidades*, las cuales deben ser satisfechas por lograr sus *servicios* con las demás clases. Los servicios y responsabilidades corresponderán finalmente a los métodos de la clase. A su vez, las clases servidores pueden tener *colaboraciones* con otras clases para lograr la satisfacción de responsabilidades que por si solas no pueden lograr. Como consecuencia de esto, se integran las responsabilidades y colaboraciones entre clases para definir *contratos* los cuales definen la naturaleza y alcance de las interacciones cliente-servidor. Los contratos y colaboraciones representan los requisitos de servicios entre los objetos. En resumen, se buscará identificar los contratos entre los objetos cliente y servidor diseñando su distribución entre los distintos objetos del sistema.

Más adelante se describen estos conceptos y todo lo relacionado con el diseño por responsabilidades. En las siguientes secciones se especificaran los aspectos mencionados a continuación con mayor detalle:

1. Especificación de las *tarjetas de clases*.
2. Identificación de las *responsabilidades* del sistema.
3. Identificación de las *colaboraciones* del sistema.

4. Identificación de las *jerarquías* de herencia del sistema.
5. Identificación de los *contratos* de servicio del sistema.
6. Identificación de los *subsistemas* del sistema.
7. Identificación de los *protocolos* del sistema.
8. Identificación de los *atributos* del sistema.

5.1.1.1 Tarjetas de Clase

Las *tarjetas de clases* (también conocidas como tarjetas *CRC: Clase-Responsabilidad-Colaboración*) permiten al diseñador visualizar las diferentes clases de manera independiente y detallada. El esquema para una tarjeta de clases se muestra en la Tabla 1.

Clase:	
Descripción:	
Módulo:	
Estereotipo:	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	

Tabla 1. Diagrama para la tarjeta de clase.

La tarjeta se divide en tres secciones:

1. Encabezado consistiendo del nombre de la *clase*, una *descripción* de la clase, el *módulo* al que pertenece la clase, el *estereotipo* de la clase (entidad, interface o control), las *propiedades* de la clase (abstracta o concreta), una lista de *superclases*, una lista de *subclases* y una lista de *atributos*.
2. Dos columnas debajo del encabezado, correspondientes a las *responsabilidades* (a la izquierda) y *colaboraciones* (a la derecha) de la clase. Eventualmente en la columna izquierda se incluirá información sobre los *contratos*. El número de filas en estas dos columnas es extensible y no está limitado al número que aparece en el diagrama de la Tabla 1.

Como ejemplo se considerará la clase *InterfaceCoordinador,* una de las clases identificadas durante el modelo de análisis. La tarjeta de clase sería la que se muestra en la Tabla 2.

Clase: InterfaceCoordinador.	
Descripción: Toda la interacción con el coordinador se hace por medio de esta interface	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	

Tabla 2. Diagrama para la tarjeta de clase de *InterfaceCoordinador*

De tal manera se debe especificar una tarjeta para cada clase en el sistema, donde inicialmente las tarjetas incluirán únicamente entradas para el nombre de la clase, módulo al que pertenecen y estereotipo correspondiente. Dado que aún no se ha identificado herencia, no habrán entradas para propiedades, superclase o subclase. Como se puede apreciar, inicialmente las tarjetas de clase tendrán información muy limitada, algo que a lo largo del diseño será extendido hasta alcanzar los métodos y atributos detallados. Una vez creadas estas tarjetas se proceden a identificar las responsabilidades de cada clase como se verá a continuación.

5.1.1.2 Responsabilidades

Uno de los esfuerzos más grandes del desarrollo y que involucra mayor complejidad es la especificación del comportamiento de cada una de las clases del sistema. A diferencia de la estructura interna de una clase, representada mediante sus atributos, los comportamientos corresponden a las operaciones y métodos de las clases. Dado que por lo general el número resultante de métodos en un sistema es mucho mayor que el de clases, el proceso de diseño involucra mayor complejidad que el de análisis. Si consideramos también la existencia de los atributos y las propias implementaciones de los métodos dentro de cada clase, la complejidad aumenta drásticamente. Sin embargo, esta complejidad no radica exclusivamente en el número de métodos, sino en el hecho que potencialmente todos los métodos de un objeto pueden ser llamados por todos los objetos del sistema. Esta es la verdadera fuente de la complejidad en la arquitectura del sistema, ya que cualquier cambio en uno de estos métodos afectará potencialmente a todo el resto del sistema. Por lo tanto, es necesario llevar a cabo un proceso muy cuidadoso para la identificación del comportamiento de las diversas clases.

En general, el comportamiento de las clases no debe descomponerse directamente en operaciones, sino seguir un procedimiento más natural comenzando con una descripción verbal de las *responsabilidades* o roles que tiene cada clase dentro del sistema. Obviamente, cada objeto instanciado de una misma clase tendrá responsabilidades similares a las descritas por la clase. Las responsabilidades ofrecidas por un objeto corresponden a los *servicios* apoyados por éste. A partir de estas responsabilidades se podrá eventualmente determinar las operaciones que cada objeto debe tener e incluso el conocimiento que el objeto posee.

Las responsabilidades se identifican a partir de los casos de uso generados durante el modelo de análisis. Para ello, se revisa el modelo de casos de uso, haciendo una lista o subrayando todos las frases descritas para determinar cuales de éstas representan acciones que algún objeto dentro del sistema deba ejecutar. Una de las decisiones más importantes durante la identificación de responsabilidades es a qué clase o clases se les debe asignar. Para ello se debe examinar el contexto en el cual se identificaron las responsabilidades.

5.1.1.2.1 Descripción de Responsabilidades del SGI-ENEO

Solo se agregaran las responsabilidades para los casos de uso tomados en los capítulos anteriores.

Validar Cliente

1. El ManejadorCliente solicita *desplegarPantallaValidacion* a la InterfaceCliente. La responsabilidad es "solicita *desplegarPantallaValidacion* a la InterfaceCliente" y se asigna a ManejadorCliente.
2. La InterfaceCliente *despliega* la PantallaValidacion. La responsabilidad *"despliega* la PantallaValidacion" se asigna a InterfaceCliente.
3. La PantallaValidacion se *despliega.* La responsabilidad es *"despliega*" y se asigna a PantallaValidación.
4. El Cliente puede seleccionar entre las siguientes opciones: "Aceptar" y "Cancelar". Esta frase es informativa describiendo opciones del Cliente, por lo cual no se agregan responsabilidades adicionales.

5. Si la actividad seleccionada es "Aceptar", La PantallaValidacion envía el evento "Aceptar" a la InterfaceCliente. La responsabilidad es "envía el evento "Aceptar" a la InterfaceCliente" y se asigna a PantallaValidación.
6. La InterfaceCliente envía el evento "Aceptar" al ManejadorCliente. La responsabilidad es "envía el evento "Aceptar" al ManejadorCliente" y se asigna a la InterfaceCliente. Asignamos de manera similar, la responsabilidad "maneja el evento "Aceptar"" al ManejadorCliente.
7. El ManejadorCliente solicita *validarRegistroCliente* a la InterfaceBaseDatosCliente. La responsabilidad es "solicita *validarRegistroCliente* a la InterfaceBaseDatosCliente" y se asigna a ManejadorCliente.
8. La InterfaceBaseDatosCliente solicita *validarRegistroCliente* a la BaseDatosCliente. La responsabilidad es "solicita *validarRegistroCliente* a la BaseDatosCliente" y se asigna a InterfaceBaseDatosCliente.
9. La BaseDatosCliente valida al cliente y devuelve *validado* a la InterfaceBaseDatosCliente. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
10. La InterfaceBaseDatosCliente devuelve *validado* al ManejadorCliente. Nuevamente, esta frase no agrega responsabilidades ya que describe un evento de devolución.
11. Una vez validado el usuario (Ex –1), el ManejadorCliente solicita *OfrecerServicios* al ManejadorServicio. La responsabilidad es "solicita *OfrecerServicios* al ManejadorServicio" la cual se asigna a ManejadorCliente. Asignamos de manera similar la responsabilidad *"OfrecerServicios"* a clase ManejadorServicio.
12. Se continua con el caso de uso *Ofrecer Servicios* y dependiendo del tipo de Cliente, el subflujo será S–2 (Usuario), S–3(Administrador), S–4(Coordinador). Esta es una frase que describe continuación entre casos de uso y no agrega ninguna responsabilidad.
13. Si la actividad seleccionada es "Salir", la PantallaValidacion envía el evento "Salir" a la InterfaceCliente. La responsabilidad es "envía el evento "Salir" a la InterfaceCliente" y se asigna a la PantallaValidación.
14. La InterfaceCliente envía el evento "Salir" al ManejadorCliente. La responsabilidad es "envía el evento "Salir" al ManejadorCliente" y se asigna a la InterfaceCliente. Asignamos de manera similar, la responsabilidad "maneja el evento "Salir"" al ManejadorCliente.
15. El ManejadorCliente solicita *OfrecerServicios* al ManejadorServicio. La responsabilidad es "solicita *OfrecerServicios* al ManejadorServicio" la cual se asigna a ManejadorCliente. Asignamos de manera similar la responsabilidad *"OfrecerServicios"* a clase ManejadorServicio.
16. Se continua con el caso de uso *Ofrecer Servicios* subflujo S–1.
    Esta es una frase que describe continuación entre casos de uso y no agrega ninguna responsabilidad.

Es posible también obtener responsabilidades a partir de las frases descritas en el manejo de excepciones, Sin embargo, nos concentraremos únicamente en los flujos básicos y no los alternos. En general, los flujos alternos, como los de excepción, son los menos comunes y son importantes de diseñar pero no en una primera etapa.

Administrar Acervo

Flujo Principal

17. El ManejadorServicio solicita *modificarAcervo* al ManejadorAcervo La responsabilidad es "solicita *modificarAcervo* al ManejadorAcervo" y se asigna al ManejadorServicio, adicionalmente asignamos la responsabilidad *Modificar Acervo al* ManejadorAcervo.
18. Se continuara con el subflujo *AdministrarRegistro Acervo* (S -1), Esta es una frase que describe continuación entre casos de uso y no agrega ninguna responsabilidad.

S–1 *Administrar Registro Acervo.*

-

19. El ManejadorAcervo solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfaceCoordinador. La responsabilidad es "solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfeceCoordinador". Se asigna al ManejadorAcervo.
20. La InterfaceCoordinador *despliega* a la PantallaObtenerRegistroAcervo. La responsabilidad *"despliega* la PantallaObtenerRegistroAcervo" se asigna a InterfaceCoordinador.
21. La PantallaObtenerRegistroAcervo se despliega. La responsabilidad *"despliega"* se asigna a la PantallaObtenerRegistroAcervo.
22. En los campos de la pantalla se recuperan automáticamente los datos del Primer registro en la Base de Datos Cliente: El Administrador o coordinador podrá seleccionar entra las siguientes actividades: "Eliminar", "Modificar", "Actualizar"(inactivo hasta que se haga clic en "Modificar"), "Búsqueda", "Primer registro", "Último registro", "Siguiente", "Anterior", "Crear" y "Salir". Estas frases son informativas y no describen ninguna responsabilidad particular de interés en este momento.
23. Si el coordinador presiona "Modificar". Se continua con el subflujo *Actualizar RegistroAcervo* subflujo (S–2). Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
24. Si el coordinador selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Acervo* (*S*-3). Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
25. Si el Coordinador selecciona "Búsqueda" se ejecuta el subflujo *Búsqueda registro* (*S*-4). Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
26. Si el coordinador selecciona "Primer registro" se ejecuta el subflujo *Recupera primer registro* (*S*-5). Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades. Si el coordinador selecciona "Último registro" se ejecuta el subflujo *Recupera último registro* (*S*-6) Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
27. Si el coordinador selecciona "Siguiente" se ejecuta el subflujo *Recupera registro siguiente* (*S*-7) Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
28. Si el Coordinador selecciona "Anterior" se ejecuta el subflujo *Recupera registro anterior* (*S*-8). Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
29. Si el coordinador selecciona "Crear" se ejecuta el subflujo *Crear registro* (*S*-9) Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
30. Si la actividad seleccionada es "Salir" la PantallaObtenerRegistroAcervo envía el evento *"Salir"* a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Salir"* a la InterfaceCoordinador" y se asigna a la PantallaObtenerRegistroAcervo.
31. La InterfaceCoordinador envía el evento "Salir" al ManejadorAcervo. La responsabilidad es "envía el evento "Salir" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Salir"" al ManejadorAcervo.
32. El ManejadorAcervo solicita *OfrecerServicios* al ManejadorServicio. La responsabilidad es "solicita *OfrecerServicios* al ManejadorServicio" la cual se asigna a ManejadorAcervo. Asignamos de manera similar la responsabilidad *"OfrecerServicios"* a clase ManejadorServicio.
33. Se continua con el caso de uso *Ofrecer Servicios* subflujo S–1. Esta es una frase que describe continuación entre casos de uso y no agrega ninguna responsabilidad.

S-2 Actualizar Registro Acervo

34. El evento "Modificar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Modificar" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
35. La InterfaceCoordinador envía el evento "Modificar" al ManejadorAcervo. La responsabilidad es "envía el evento "Modificar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Modificar"" al ManejadorAcervo.
36. El ManejadorAcervo solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfaceCoordinador. La responsabilidad es "solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfeceCoordinador". Se asigna al ManejadorAcervo.
37. La InterfaceCoordinador *despliega* a la PantallaObtenerRegistroAcervo. La responsabilidad *"despliega* la PantallaObtenerRegistroAcervo" se asigna a InterfaceCoordinador.
38. La PantallaObtenerRegistroAcervo se despliega. La responsabilidad *"despliega"* se asigna a la PantallaObtenerRegistroAcervo.
39. Se activa el botón "Actualizar", de la PantallaObtenerRegistroAcervo. Y el Coordinador introduce los nuevos datos. Estas frases son informativas y no describen ninguna responsabilidad particular de interés en este momento.
40. El evento "Actualizar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Actualizar"* a la InterfaceCoordinador" y se asigna a la PantallaObtenerRegistroAcervo.
41. La InterfaceCoordinador envía el evento "Actualizar" al ManejadorAcervo. La responsabilidad es "envía el evento "Actualizar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Actualizar"" al ManejadorAcervo.
42. El ManejadorAcervo solicita *actualizar* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *actualizar* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
43. La InterfaceBaseDatosAcervo solicita *actualizar* a la BaseDatosAcervo. La responsabilidad es "solicita *actualizar* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
44. La BaseDatosAcervo devuelve *actualizado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
45. La InterfaceBaseDatosAcervo devuelve *actualizado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
46. . Se continúa con el subflujo *Administrar Registro Acervo(S-1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S-3 Eliminar Registro Acervo

47. El evento "Eliminar" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Eliminar" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
48. La InterfaceCoordinador envía el evento "Eliminar" al ManejadorAcervo. La responsabilidad es "envía el evento "Eliminar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Eliminar"" al ManejadorAcervo.
49. El ManejadorAcervo solicita *eliminar* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *eliminar* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
50. La InterfaceBaseDatosAcervo solicita *eliminar* a la BaseDatosAcervo. La responsabilidad es "solicita *eliminar* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.

51. La BaseDatosAcervo devuelve *eliminado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
52. La InterfaceBaseDatosAcervo devuelve *eliminado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
53. Se continúa con el subflujo *Administrar Registro Acervo(S–1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 4 *Búsqueda Registro.*

54. El evento "Búsqueda" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Búsqueda" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
55. La InterfaceCoordinador envía el evento "Búsqueda" al ManejadorAcervo. La responsabilidad es "envía el evento "Búsqueda" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Búsqueda"" al ManejadorAcervo.
56. El ManejadorAcervo solicita *desplegarPantallaBuscarRegistroAcervo* a la InterfaceCoordinador. La responsabilidad es "solicita *desplegarPantallaBuscarRegistroAcervo* a la InterfaceCoordinador". Se asigna al ManejadorAcervo.
57. La InterfaceCoordinador *despliega* a la PantallaBuscarRegistroAcervo. La responsabilidad *"despliega* la PantallaBuscarRegistroAcervo" se asigna a InterfaceCoordinador.
58. La PantallaBuscarRegistroAcervo se despliega. La responsabilidad *"despliega"* se asigna a la PantallaObtenerRegistroAcervo.
59. En la PantallaBuscarRegistroAcervo se pide los datos del registro a buscar. Se presenta las siguientes opciones: "Buscar" "Cancelar". Estas frases son informativas y no describen ninguna responsabilidad particular de interés en este momento.
60. Si la actividad seleccionada es "Buscar". El evento "Buscar" es enviado por la PantallaBuscarRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Buscar"* a la InterfaceCoordinador" y se asigna a la PantallaBuscarRegistroAcervo.
61. La InterfaceCoordinador envía el evento "Buscar" al ManejadorAcervo. La responsabilidad es "envía el evento "Buscar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Buscar"" al ManejadorAcervo.
62. El ManejadorAcervo solicita *buscarAcervo* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *buscarAcervo* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
63. La InterfaceBaseDatosAcervo solicita *buscarAcervo* a la BaseDatosAcervo. La responsabilidad es "solicita *buscarAcervo* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
64. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
65. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
66. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
67. Si la actividad seleccionada es "Cancelar". El evento "Cancelar" es enviado por la PantallaBuscarRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Cancelar"* a la InterfaceCoordinador" y se asigna a la PantallaBuscarRegistroAcervo.

68. La InterfaceCoordinador envía el evento "Cancelar" al ManejadorAcervo. La responsabilidad es "envía el evento "Cancelar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Cancelar"" al ManejadorAcervo.
69. Se continúa con el subflujo *Administrar Registro Acervo(S–1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 5 *Recuperar Primer Registro.*

70. El evento "Primer Registro" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Primer Registro" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
71. La InterfaceCoordinador envía el evento "Primer Registro" al ManejadorAcervo. La responsabilidad es "envía el evento "Primer Registro" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Primer Registro"" al ManejadorAcervo.
72. El ManejadorAcervo solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
73. La InterfaceBaseDatosAcervo solicita *obtenerPrimerRegistro* a la BaseDatosAcervo. La responsabilidad es "solicita *obtenerPrimerRegistro* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
74. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
75. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
76. Se continúa con el subflujo *Administrar Registro Acervo(S–1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 6 *Recuperar Último Registro.*

77. El evento "Último Registro" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Ultimo Registro" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
78. La InterfaceCoordinador envía el evento "Ultimo Registro" al ManejadorAcervo. La responsabilidad es "envía el evento "Ultimo Registro" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Ultimo Registro"" al ManejadorAcervo.
79. El ManejadorAcervo solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *obtenerUltimoRegistro* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
80. La InterfaceBaseDatosAcervo solicita *obtenerUltimoRegistro* a la BaseDatosAcervo. La responsabilidad es "solicita *obtenerUltimoRegistro* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
81. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
82. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.

83. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 7 *Recuperar Registro Siguiente.*

84. El evento "Registro Siguiente" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Registro Siguiente" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
85. La InterfaceCoordinador envía el evento "Registro Siguiente" al ManejadorAcervo. La responsabilidad es "envía el evento "Registro Siguiente" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Registro Siguiente"" al ManejadorAcervo.
86. El ManejadorAcervo solicita *obtenerRegistroSiguiente* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *obtenerRegistroSiguiente* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
87. La InterfaceBaseDatosAcervo solicita *obtenerRegistroSiguiente* a la BaseDatosAcervo. La responsabilidad es "solicita *obtenerRegistroSiguiente* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
88. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
89. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
90. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 8 *Recuperar Registro Anterior.*

91. El evento "Registro Anterior" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Registro Anterior" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
92. La InterfaceCoordinador envía el evento "Registro Anterior" al ManejadorAcervo. La responsabilidad es "envía el evento "Registro Anterior" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Registro Anterior"" al ManejadorAcervo.
93. El ManejadorAcervo solicita *obtenerRegistroAnterior* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *obtenerRegistroAnterior* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
94. La InterfaceBaseDatosAcervo solicita *obtenerRegistroAnterior* a la BaseDatosAcervo. La responsabilidad es "solicita *obtenerRegistroAnterior* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
95. La BaseDatosAcervo devuelve *resultado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
96. La InterfaceBaseDatosAcervo devuelve *resultado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
97. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

S – 9 Crear Registro.

98. El evento "Crear" es enviado por la PantallaObtenerRegistroAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento "Crear" a la InterfaceCoordinador" y se asigna a PantallaObtenerRegistroAcervo.
99. La InterfaceCoordinador envía el evento "Crear" al ManejadorAcervo. La responsabilidad es "envía el evento "Crear" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Crear"" al ManejadorAcervo.
100. El ManejadorAcervo solicita *desplegarPantallaDatosAcervo* a la InterfaceCoordinador. La responsabilidad es "solicita *desplegarPantallaDatosAcervo* a la InterfaceCoordinador". Se asigna al ManejadorAcervo.
101. La InterfaceCoordinador *despliega* a la PantallaDatosAcervo. La responsabilidad *"despliega* la PantallaDatosAcervo" se asigna a InterfaceCoordinador.
102. La PantallaDatosAcervo se despliega. La responsabilidad *"despliega"* se asigna a la PantallaDatosAcervo.
103. En la PantallaDatosAcervo se pide los datos del registro a buscar. Se presenta las siguientes opciones:"Crear Registro" "Cancelar". Estas frases son informativas y no describen ninguna responsabilidad particular de interés en este momento.
104. Si la actividad seleccionada es "Crear Registro". El evento "Crear Registro" es enviado por la PantallaDatosAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Crear Registro"* a la InterfaceCoordinador" y se asigna a la PantallaDatosAcervo.
105. La InterfaceCoordinador envía el evento "Crear Registro" al ManejadorAcervo. La responsabilidad es "envía el evento "Crear Registro" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Crear Registro"" al ManejadorAcervo.
106. El ManejadorAcervo solicita *crearRegistroAcervo* a la InterfaceBaseDatosAcervo. La responsabilidad es "solicita *crearRegistroAcervo* a la InterfaceBaseDatosAcervo" y se asigna a ManejadorAcervo.
107. La InterfaceBaseDatosAcervo solicita *crearRegistroAcervo* a la BaseDatosAcervo. La responsabilidad es "solicita *crearRegistroAcervo* a la BaseDatosAcervo" y se asigna a InterfaceBaseDatosAcervo.
108. La BaseDatosAcervo devuelve *creado* a la InterfaceBaseDatosAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
109. La InterfaceBaseDatosAcervo devuelve *creado* al ManejadorAcervo. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
110. Se continúa con el subflujo *Administrar Registro Acervo(S–1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.
111. Si la actividad seleccionada es "Cancelar". El evento "Cancelar" es enviado por la PantallaDatosAcervo a la InterfaceCoordinador. La responsabilidad es "envía el evento *"Cancelar"* a la InterfaceCoordinador" y se asigna a la PantallaDatosAcervo.
112. La InterfaceCoordinador envía el evento "Cancelar" al ManejadorAcervo. La responsabilidad es "envía el evento "Cancelar" al ManejadorAcervo" y se asigna a la InterfaceCoordinador. Asignamos de manera similar, la responsabilidad "maneja el evento "Cancelar"" al ManejadorAcervo.
113. Se continúa con el subflujo *Administrar Registro Acervo(S – 1)*. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

MANEJADORES CONTROL

Se agregan las tarjetas de los objetos para los casos de uso Validar Cliente y Administrar Acervo.

–

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encargara de controlar las distintas pantallas para los diferentes tipos de clientes	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
OfrecerServicios (11)	
OfrecerServicios (15)	
solicita *desplegarPantallaValidaCredencial* a la *InterfaceUsuario* (17)	
solicita *consultarAcervo* al ManejadorConsultaAcervo. (28)	
ofrecerServicios (49)	
solicita *modificarAcervo* al ManejadorAcervo (51)	
OfrecerServicios (66)	
solicita *modificarCliente* al ManejadorCliente (162)	
OfrecerServicios (177)	
solicita *modificarUsuario* al Manejador*Usuario* (259)	
OfrecerServicios (274)	
solicita *inscripcion* al Manejador*Usuario* (347)	
OfrecerServicios (358)	
OfrecerServicios(362)	
solicita *desplegarPantallaOfrecerServicio* a la InterfaceCliente (377)	
maneja el evento "Validacion" (382)	
solicita *validacion* al ManejadorCliente (383)	
maneja el evento "Validacion" (391)	
solicita *validacion* al ManejadorCliente (392)	
maneja el evento "Validar *Usuario*" (396)	
maneja el evento "Consultar Acervo" (399)	
maneja el evento "Validacion" (403)	
solicita *validacion* al ManejadorCliente (404)	
maneja el evento "Modificar Acervo" (411)	
maneja el evento "Incripcion" (414)	
maneja el evento "Modificar *Usuario*" (417)	
maneja el evento "Reporte *Usuario*" (420)	
maneja el evento "Reporte Acervo" (423)	
maneja el evento "Administrar *Usuarios* por Area" (426)	
maneja el evento "Validacion" (430)	
solicita *validacion* al ManejadorCliente (431)	
maneja el evento "Modificar Acervo" (438)	
maneja el evento "Inscripcion" (441)	
maneja el evento "Modificar *Usuario*" (444)	
maneja el evento "Reporte *Usuario*" (447)	

maneja el evento "Reporte Acervo" (450)	
maneja el evento "Administrar *Usuarios* por Area" (453)	
maneja el evento "Modificar Cliente" (456)	
solicita *administrarUsuarioPorArea* al ManejadorUsuarioArea (458)	
OfrecerServicios (467)	
solicita *generarReporteUsuario* al ManejadorReporte*Usuario* (492)	
OfrecerServicios (505)	
solicita *generarReporteAcervo* al ManejadorReporteAcervo (581)	
OfrecerServicios (591)	

Clase: ManejadorCliente	
Descripción: El manejador Clientes se encarga de controlar los procesos asociados con el cliente.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
solicita *desplegarPantallaValidacion* a la InterfaceCliente (1)	
solicita *validarRegistroCliente* a la InterfaceBaseDatosCliente (7)	
solicita *ofrecerServicios* al ManejadorServicio (11)	
maneja el evento "Salir" (14)	
solicita *ofrecerServicios* al ManejadorServicio (15)	
solicita *desplegarPantallaObtenerRegistroCliente* a la InterfeceCoordinador (164)	
maneja el evento "Salir" (176)	
solicita *ofrecerServicios* al ManejadorServicio (177)	
maneja el evento "Modificar" (180)	
solicita *desplegarPantallaObtenerRegistroCliente* a la InterfeceCoordinador (181)	
maneja el evento "Actualizar" (186)	
solicita *actualizar* a la InterfaceBaseDatosCliente (187)	
maneja el evento "Eliminar" (193)	
solicita *eliminar* a la InterfaceBaseDatosCliente (194)	
maneja el evento "Busqueda" (200)	
solicita *desplegarPantallaBuscarRegistroCliente* a la InterfaceCoordinador (201)	
maneja el evento "Buscar" (206)	
solicita *buscarCliente* a la InterfaceBaseDatosCliente (207)	
maneja el evento "Cancelar" (213)	
maneja el evento "Primer Registro" (216)	

solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosCliente (217)	
maneja el evento "Ultimo Registro" (223)	
solicita *obtenerUltimoRegistro* a la InterfaceBaseDatosCliente (224)	
maneja el evento "Registro Siguiente" (230)	
solicita *obtenerRegistroSiguiente* a la InterfaceBaseDatosCliente (231)	
maneja el evento "Registro Anterior" (237)	
solicita *obtenerRegistroAnterior* a la InterfaceBaseDatosCliente (238)	
maneja el evento "Crear" (244)	
solicita *desplegarPantallaDatos*Cliente a la InterfaceCoordinador (245)	
maneja el evento "Crear Registro" (250)	
solicita *crearRegistro*Cliente a la InterfaceBaseDatosCliente (251)	
maneja el evento "Cancelar" (257)	
maneja el evento "Salir Sistema" (386)	
solicita *cerrarElSistema (387)*	
maneja el evento "Salir Sistema" (407)	
solicita *cerrarElSistema (408)*	
maneja el evento "Salir Sistema" (434)	
solicita *cerrarElSistema (435)*	

Clase: ManejadorAcervo	
Descripción: El manejador Acervo controla los registros almacenados en el acervo.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfeceCoordinador (53)	
maneja el evento "Salir" (65)	
solicita *ofrecerServicios* al ManejadorServicio (66)	
maneja el evento "Modificar" (70)	
solicita *desplegarPantallaObtenerRegistroAcervo* a la InterfeceCoordinador (71)	
maneja el evento "Actualizar" (76)	
solicita *actualizar* a la InterfaceBaseDatosAcervo (77)	
maneja el evento "Eliminar" (83)	
solicita *eliminar* a la InterfaceBaseDatosAcervo (84)	
maneja el evento "Busqueda" (90)	
solicita *desplegarPantallaBuscarRegistroAcervo* a la InterfaceCoordinador (91)	
maneja el evento "Buscar" (96)	
solicita *buscarAcervo* a la InterfaceBaseDatosAcervo (97)	
maneja el evento "Cancelar" (103)	
maneja el evento "Primer Registro" (106)	
solicita *obtenerPrimerRegistro* a la InterfaceBaseDatosAcervo (107)	
maneja el evento "Ultimo Registro" (113)	
solicita *obtenerUltimoRegistro* a la InterfaceBaseDatosAcervo (114)	
maneja el evento "Registro Siguiente" (120)	

solicita *obtenerRegistroSiguiente* a la InterfaceBaseDatosAcervo (121)	
maneja el evento "Registro Anterior" (127)	
solicita *obtenerRegistroAnterior* a la InterfaceBaseDatosAcervo (128)	
maneja el evento "Crear" (134)	
solicita *desplegarPantallaDatosAcervo* a la InterfaceCoordinador (135)	
maneja el evento "Crear Registro" (140)	
solicita *crearRegistroAcervo* a la InterfaceBaseDatosAcervo (141)	
maneja el evento "Cancelar" (147)	
maneja el evento "Cancelar"" al ManejadorAcervo. (160)	

INTERFACES

Clase: InterfaceCliente	
Descripción: Toda la interacción con el Cliente se hace por medio de la interface cliente.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega la PantallaValidacion (2)	
envía el evento "Aceptar" al ManejadorCliente (6)	
envía el evento "Salir" al ManejadorCliente (14)	
despliega la PantallaOfrecerServicio (378)	
envía el evento "Validacion" al ManejadorServicio (382)	
envía el evento "Salir sistema" al ManejadorCliente (386)	

Clase: InterfaceBaseDatosCliente	
Descripción: La información de cada *Usuario* se almacena en la base de datos cliente la cual se accesa mediante la interface de la base de datos cliente. Esto permite validar a los distintos *Usuarios* además de guardar información sobre estos.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
solicita *validarRegistroCliente* a la BaseDatosCliente (8)	
solicita *validarTiempo* a la BaseDatosCliente (24)	
solicita *actualizar* a la BaseDatosCliente (188)	
solicita *eliminar* a la BaseDatosCliente (195)	
solicita *buscarCliente* a la BaseDatosCliente (208)	
solicita *obtenerPrimerRegistro* a la BaseDatosCliente (218)	

solicita *obtenerUltimoRegistro* a la BaseDatosCliente (225)	
solicita *obtenerRegistroSiguiente* a la BaseDatosCliente (232)	
solicita *obtenerRegistroAnterior* a la BaseDatosCliente (239)	
solicita *crearRegistroCliente* a la BaseDatosCliente (252)	
solicita *actualizar* a la BaseDatosCliente (285)	
solicita *eliminar* a la BaseDatosCliente (292)	
solicita *buscarUsuario* a la BaseDatosCliente (305)	
solicita *obtenerPrimerRegistro* a la BaseDatosCliente (315)	
solicita *obtenerUltimoRegistro* a la BaseDatosCliente (322)	
solicita *obtenerRegistroSiguiente* a la BaseDatosCliente (329)	
solicita *obtenerRegistroAnterior* a la BaseDatosCliente (336)	
solicita *obtenerRegistroAnterior* a la BaseDatosCliente (342)	
solicita *crearNuevoRegistroUsuario* a la BaseDatosCliente (355)	
solicita *guardar* a la BaseDatosCliente (472)	
solicita *obtenerDatos* a la BaseDatosCliente (485)	
solicita *obtenerDatosInsCat* a la BaseDatosCliente (510)	
solicita *obtenerDatosInsCatG* a la BaseDatosCliente (526)	
solicita *obtenerDatosInsCat* a la BaseDatosCliente (536)	
solicita *obtenerDatosUsuArea* a la BaseDatosCliente (547)	
solicita *obtenerDatosUsuAreaG* a la BaseDatosCliente (563)	
solicita *obtenerDatos* a la BaseDatosCliente (573)	

Clase: InterfaceBaseDatosAcervo	
Descripción: La información de cada registro acervo se almacena en la base de datos acervo la cual se accesa mediante la interface de la base de datos acervo. Permite hacer consultas, crear, actualizar registrosdel acervo.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
solicita *buscarMaterial* a la BaseDatosAcervo (36)	
solicita *actualizar* a la BaseDatosAcervo (78)	
solicita *eliminar* a la BaseDatosAcervo (85)	
solicita *buscarAcervo* a la BaseDatosAcervo (98)	
solicita *obtenerPrimerRegistro* a la BaseDatosAcervo (108)	

solicita *obtenerUltimoRegistro* a la BaseDatosAcervo (115)	
solicita *obtenerRegistroSiguiente* a la BaseDatosAcervo (122)	
solicita *obtenerRegistroAnterior* a la BaseDatosAcervo (129)	
solicita *crearRegistroAcervo* a la BaseDatosAcervo (142)	
solicita *obtenerDatosAcerExist* a la BaseDatosAcervo (596)	
solicita *obtenerDatosAcerExistG* a la BaseDatosAcervo (612)	
solicita *obtenerDatosAcerExist* a la BaseDatosAcervo (622)	
solicita *obtenerDatosAdq* a la BaseDatosAcervo (639)	
solicita *obtenerDatosAdqG* a la BaseDatosAcervo (655)	
solicita *obtenerDatosAdq* a la BaseDatosAcervo (665)	

Clase: InterfaceCoordinador.	
Descripción: Toda la interacción con el coordinador se hace por medio de la interface coordinador.	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega la PantallaObtenerRegistroAcervo (54)	
Envía el evento "Salir" al ManejadorAcervo (65)	
Envía el evento "Modificar" al ManejadorAcervo (70)	
despliega la PantallaObtenerRegistroAcervo (72)	
Envía el evento "Actualizar" al ManejadorAcervo (76)	
Envía el evento "Eliminar" al ManejadorAcervo (83)	
Envía el evento "Busqueda" al ManejadorAcervo (90)	
despliega la PantallaBuscarRegistroAcervo (92)	
Envía el evento "Buscar" al ManejadorAcervo (96)	
Envía el evento "Cancelar" al ManejadorAcervo (103)	
Envía el evento "Primer Registro" al ManejadorAcervo (106)	
Envía el evento "Ultimo Registro" al ManejadorAcervo (113)	
Envía el evento "Registro Siguiente" al ManejadorAcervo. (120)	
Envía el evento "Registro Anterior" al ManejadorAcervo (127)	
Envía el evento "Crear" al ManejadorAcervo (134)	
despliega la PantallaDatosAcervo (136)	
Envía el evento "Crear Registro" al ManejadorAcervo (140)	
Envía el evento "Cancelar" al ManejadorAcervo (147)	
despliega la PantallaObtenerRegistroCliente (165)	
Envía el evento "Salir" al ManejadorCliente (176)	
Envía el evento "Modificar" al ManejadorCliente (180)	
despliega la PantallaObtenerRegistroCliente (182)	
Envía el evento "Actualizar" al ManejadorCliente (186)	
Envía el evento "Eliminar" al ManejadorCliente (193)	

Envía el evento "Busqueda" al ManejadorCliente (200)	
despliega la PantallaBuscarRegistroCliente (202)	
Envía el evento "Buscar" al ManejadorCliente (206)	
Envía el evento "Cancelar" al ManejadorCliente (213)	
Envía el evento "Primer Registro" al ManejadorCliente (216)	
Envía el evento "Ultimo Registro" al ManejadorCliente (223)	
Envía el evento "Registro Siguiente" al ManejadorCliente. (230)	
Envía el evento "Registro Anterior" al ManejadorCliente (237)	
Envía el evento "Crear" al ManejadorCliente (244)	
despliega la PantallaDatosCliente (246)	
Envía el evento "Crear Registro" al ManejadorCliente (250)	
Envía el evento "Cancelar" al ManejadorCliente (257)	
Despliega la PantallaObtenerRegistro*Usuario* (262)	
Envía el evento "Salir" al Manejador*Usuario* (273)	
Envía el evento "Modificar" al Manejador*Usuario* (277)	
Despliega la PantallaObtenerRegistro*Usuario* (279)	
Envía el evento "Actualizar" al Manejador*Usuario* (283)	
Envía el evento "Eliminar" al Manejador*Usuario* (290)	
Envía el evento "Busqueda" al Manejador*Usuario* (297)	
Despliega la PantallaBuscarRegistro*Usuario* (299)	
Envía el evento "Buscar" al Manejador*Usuario* (303)	
Envía el evento "Cancelar" al Manejador*Usuario* (310)	
Envía el evento "Primer Registro" al Manejador*Usuario* (313)	
Envía el evento "Ultimo Registro" al Manejador*Usuario* (320)	
Envía el evento "Registro Siguiente" al Manejador*Usuario*. (327)	
Envía el evento "Registro Anterior" al Manejador*Usuario* (334)	
Envía el evento "Reinscribir" al Manejador*Usuario* (341)	
Despliega la PantallaIncripcion (349)	
Envía el evento "Inscripcion" al Manejador*Usuario* (353)	
Envía el evento "Salir" al Manejador*Usuario* (361)	
Envía el evento "Imprimir" al Manejador*Usuario* (365)	
Despliega la PantallaImprimirCredencial (367)	
Envía el evento "Imprimir" al Manejador*Usuario*" y se asigna a la InterfaceCoordinador. (371)	
Envía el evento "Cancelar" al Manejador*Usuario*	
Envía el evento "Validacion" al ManejadorServicio (430)	
Envía el evento "Salir sistema" al ManejadorCliente (434)	
Envía el evento "Modificar Acervo" al ManejadorServicio (438)	
Envía el evento "Inscripcion" al ManejadorServicio (441)	
Envía el evento "Modificar *Usuario*" al ManejadorServicio (444)	
Envía el evento "Reporte *Usuario*" al ManejadorServicio (447)	
Envía el evento "Reporte Acervo" al ManejadorServicio (450)	
Envía el evento "Administrar *Usuarios* por Area" al ManejadorServicio (453)	
Envía el evento "Modificar Cliente" al ManejadorServicio (456)	
despliega la PantallaDatos*Usuario*Area (460)	

Envía el evento "Salir" al Manejador*Usuario*Area (466)	
Envía el evento "Guardar" al Manejador*Usuario*Area (470)	
Envía el evento "Modificar" al Manejador*Usuario*Area (477)	
despliega la PantallaModificar*Usuario*Area (479)	
Envía el evento "Modificar" al Manejador*Usuario*Area (483)	
Envía el evento "Regresar" al Manejador*Usuario*Area (490)	
despliega la PantallaReporteInicial*Usuario* (495)	
Envía el evento "Salir" al ManejadorReporte*Usuario* (504)	
Envía el evento "Inscripción por categoría" al ManejadorReporte*Usuario* (508)	
despliega la PantallaReporteResultado*Usuario*IC (514)	
Envía el evento "Guardar" al ManejadorReporte*Usuario* (518)	
despliega la PantallaReporteGuardar*Usuario*IC (520)	
Envía el evento "Guardar" al ManejadorReporte*Usuario* (524)	
Envía el evento "Cancelar" al ManejadorReporte*Usuario* (531)	
Envía el evento "Imprimir" al ManejadorReporte*Usuario* (534)	
Envía el evento "Regresar" al ManejadorReporte*Usuario* (542)	
Envía el evento "*Usuario*s por Area" al ManejadorReporte*Usuario* (545)	
despliega la PantallaReporteResultado*Usuario*UA (551)	
Envía el evento "Guardar" al ManejadorReporte*Usuario* (555)	
despliega la PantallaReporteGuardar*Usuario*UA (557)	
Envía el evento "Guardar" al ManejadorReporte*Usuario* (561)	
Envía el evento "Cancelar" al ManejadorReporte*Usuario* (568)	
Envía el evento "Imprimir" al ManejadorReporte*Usuario* (571)	
Envía el evento "Regresar" al ManejadorReporte*Usuario* (579)	
despliega la PantallaReporteInicialAcervo (584)	
Envía el evento "Salir" al ManejadorReporteAcervo (590)	
Envía el evento "Acervo existente" al ManejadorReporteAcervo (594)	
despliega la PantallaReporteResultadoAcervoAE (600)	
Envía el evento "Guardar" al ManejadorReporteAcervo (604)	
despliega la PantallaReporteGuardarAcervoAE (606)	
Envía el evento "Guardar" al ManejadorReporteAcervo (610)	
Envía el evento "Cancelar" al ManejadorReporteAcervo (617)	
Envía el evento "Imprimir" al ManejadorReporteAcervo (619)	
Envía el evento "Regresar" al ManejadorReporteAcervo (628)	
Envía el evento "Adquisiciones" al ManejadorReporteAcervo (631)	
despliega la desplegarPantallaReporteAdquisiciones (633)	
envía el evento "Aceptar" al ManejadorReporteAcervo (637)	
despliega la PantallaReporteResultadoAcervoAdq (643)	
envía el evento "Guardar" al ManejadorReporteAcervo (647)	
despliega la PantallaReporteGuardarAcervoAdq (649)	
envía el evento "Guardar" al ManejadorReporteAcervo (653)	
envía el evento "Cancelar" al ManejadorReporteAcervo (660)	
envía el evento "Imprimir" al ManejadorReporteAcervo (663)	
envía el evento "Regresar" al ManejadorReporteAcervo (671)	
envía el evento "Cancelar" al ManejadorReporteAcervo (660)	

PANTALLAS *USUARIO*

Clase: PantallaValidacion	
Descripción: Pantalla por medio de la cual se valida el tipo de *Usuario* que desea ingresar al sistema.	
Módulo: Pantallas	
Estereotipo: Borde	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega (3)	
envía el evento "Aceptar" a la InterfaceCliente (5)	
envía el evento "Salir" a la InterfaceCliente (13)	

PANTALLAS ADMINISTRACIÓN ACERVO

Clase: PantallaObtenerRegistroAcervo.	
Descripción: Pantalla principal para realizar la administración de los registros acervo.	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega (55)	
envía el evento *"Salir"* a la InterfaceCoordinador (64)	
envía el evento "Modificar" a la InterfaceCoordinador (69)	
despliega (73)	
envía el evento *"Actualizar"* a la InterfaceCoordinador (75)	
envía el evento "Eliminar" a la InterfaceCoordinador (82)	
envía el evento "Busqueda" a la InterfaceCoordinador (89)	
envía el evento "Primer Registro" a la InterfaceCoordinador (105)	
envía el evento "Ultimo Registro" a la InterfaceCoordinador (112)	
envía el evento "Registro Siguiente" a la InterfaceCoordinador (119)	
envía el evento "Registro Anterior" a la InterfaceCoordinador (126)	
envía el evento "Crear" a la InterfaceCoordinador (133)	
envía el evento "Imprimir" a la InterfaceCoordinador (149)	

Clase: PantallaBuscarRegistroAcervo.
Descripción: Pantalla de busqueda para materiales del acervo, utilizada por un Coordinador o Administrador.
Módulo: Interfaces
Estereotipo: Borde.
Propiedades:
Superclase:
Subclases:

Atributos:	
despliega (93)	
envía el evento *"Buscar"* a la InterfaceCoordinador (95)	
envía el evento *"Cancelar"* a la InterfaceCoordinador (102)	

Clase: PantallaDatosAcervo.	
Descripción: Pantalla de captura de datos de un nuevo registro del material Acervo.	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega (137)	
envía el evento *"Crear Registro"* a la InterfaceCoordinador (139)	
envía el evento *"Cancelar"* a la InterfaceCoordinador (146)	

Clase: PantallaOfrecerServicio.	
Descripción: Pantalla que le ofrece al *Usuario*, administrador y coordinador las distintas operaciones que pueden realizar con el sistema	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades:	
Superclase:	
Subclases:	
Atributos:	
despliega (379)	
envía el evento "Validacion" a la InterfaceCliente (381)	
envía el evento "Salir Sistema" a la InterfaceCliente (385)	
envía el evento "Validacion" a la Interface*Usuario* (390)	
envía el evento "Validar *Usuario*" a la Interface*Usuario* (395)	
envía el evento "Consultar Acervo" a la Interface*Usuario* (398)	
envía el evento "Validacion" a la InterfaceAdministrador (402)	
envía el evento "Salir Sistema" a la InterfaceAdministrador (406)	
envía el evento "Modificar Acervo" a la InterfaceAdministrador (410)	
envía el evento "Inscripcion" a la InterfaceAdministrador (413)	
envía el evento "Modificar *Usuario*" a la InterfaceAdministrador (416)	
envía el evento "Reporte *Usuario*" (419)	
envía el evento "Reporte Acervo" a la InterfaceAdministrador (422)	
envía el evento "Administrar *Usuario*s por Area" a la InterfaceAdministrador (425)	
envía el evento "Validacion" a la InterfaceAdministrador (429)	
envía el evento "Salir Sistema" a la InterfaceAdministrador (433)	
envía el evento "Modificar Acervo" a la InterfaceAdministrador (437)	

envía el evento "Inscripcion" a la InterfaceAdministrador (440)	
envía el evento "Modificar *Usuario*" a la InterfaceAdministrador (443)	
envía el evento "Reporte *Usuario*" (446)	
envía el evento "Reporte Acervo" a la InterfaceAdministrador (449)	
envía el evento "Administrar *Usuarios* por Area" a la InterfaceAdministrador (452)	
envía el evento "Modificar Cliente" a la Interface Coordinador (455)	

5.1.1.3 Colaboraciones

Es indispensable que los objetos dentro de un programa colaboren entre si o de lo contrario el programa consistirá de múltiples "mini-programas" independientes. Las colaboraciones entre objetos se dan en base a las relaciones entre las responsabilidades de las distintas clases. Dado la infinidad de posibles relaciones entre clase, las colaboraciones son la fuente principal de complejidad en el diseño de objetos.

Las colaboraciones representan solicitudes de un objeto *cliente* a un objeto *servidor*. Los objetos pueden jugar el papel de clientes o servidores dependiendo de su actividad en ese momento. Un objeto juega el papel de servidor cuando satisface una solicitud hecha por un objeto cliente. Desde el punto de vista del cliente, cada una de sus solicitudes de servicio o colaboración se asocia con una responsabilidad implementada por algún servidor. A su vez, la clase que ofrece el servicio puede solicitar una colaboración con alguna otra clase servidor. Esta cadena de solicitudes y servicios puede extenderse según sea necesario.

El proceso de identificación de colaboraciones se basa en la funcionalidad de la aplicación y de la arquitectura de clases propuesta, algo que fue hecho durante la etapa de análisis. Para identificar colaboraciones se analiza las responsabilidades de cada clase, decidiendo si cada una de estas clases es capaz de satisfacer sus responsabilidades por si misma o si requiere de otra clase para lograrlo. Se analiza qué tanto sabe cada clase y qué otras clases necesitan su conocimiento o funcionalidad.

Es importante recordar que las colaboraciones son en una sola dirección, y que ciertos tipos de clases son típicamente clientes o servidores, ubicándose las clase en uno u otro extremo de la colaboración. En la mayoría de los casos, las clases correspondientes al estereotipo de *borde* y *entidad* no son clientes de otros objetos en el sistema, si no que existen para ser servidores, donde otras clases colaboran con ellas, típicamente las clases de *control*.

Durante el proceso de identificación de las colaboraciones se toma la tarjeta de clase que juega el papel de cliente, escribiendo en la columna derecha el nombre de la clase que juega el papel de servidor o sea la clase que colabora para la responsabilidad particular. Se escribe ese nombre directamente a la derecha de la responsabilidad a la cual la colaboración ayuda a satisfacer. Aunque una responsabilidad pudiera requerir de varias colaboraciones, se escribe el nombre de la clase principal como servidor de la colaboración. Las demás clases típicamente pasan a ser parámetros dentro de las llamadas de métodos al momento de implementarse, por lo cual se pueden mantener como parte de la responsabilidad general.

5.1.1.4 Jerarquías

El diseño de las jerarquías de herencia es uno de los aspectos de programación más importantes de la orientación a objetos. Mediante la herencia se puede lograr una buena reutilización del código del sistema, logrando arquitecturas de clases más compactas lo cual puede reducir radicalmente el tamaño del sistema final.

La herencia se identifica a partir de las responsabilidades y colaboraciones obtenidas anteriormente. De manera general, existen dos formas de aprovechar la herencia. La forma más común es la creación de superclases que guarden responsabilidades comunes a múltiples clases. La forma adicional y más avanzada está relacionada con el *polimorfismo* y busca aprovechar no sólo responsabilidades comunes sino también colaboraciones comunes entre clases. Estos dos

enfoques también sirven de base para la extensibilidad de la arquitectura de clases. Por ejemplo, si varias clases definen una responsabilidad similar, se puede introducir una superclase de la cual estas clases hereden la responsabilidad común. Las nuevas superclases típicamente son clases abstractas. Se necesita solo una responsabilidad para definir una superclase abstracta, pero se necesita por lo menos dos subclases antes de esperar diseñar una abstracción general útil.

En esta etapa se debe revisar las tarjetas de clases y clasificar cada clase según si es concreta o abstracta. Se debe además agregar tarjetas para las superclases (abstractas o concretas) según sea necesario y reasignar responsabilidades para corresponder al nuevo diseño. Mediante la reasignación de responsabilidades se busca producir jerarquías de clases que puedan reutilizarse y extenderse mas fácilmente. Aunque llevaremos a cabo una sola etapa de diseño de jerarquías, la herencia debe aplicarse continuamente a lo largo del diseño de objetos. Se listan en la tarjeta de clase las responsabilidades propias no heredadas y aquellas responsabilidades que se sobrescriben. No es necesario volver a escribir las responsabilidades que se heredan de las superclases. Nótese en las siguientes secciones que se agregan nuevas superclases, para lo cual se agregan las correspondientes tarjetas de clases, y se está agregando información en las secciones de subclases y superclases, además de la especificación de si la clase es concreta o abstracta.

Para comenzar se analizan las diversas responsabilidades y colaboraciones asignadas a las distintas interfaces que interactúan con los actores primarios del sistema, estas interfaces son: Usuario, Administrador y Coordinador. En todas estas interfaces podemos apreciar que hay dos grupos de responsabilidades, aquellos correspondientes a "despliega" y las correspondientes a "envía el evento...". En el caso de la responsabilidad "despliega" la responsabilidad se llama también "despliega" para todas las clases colaboradoras:
PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones.

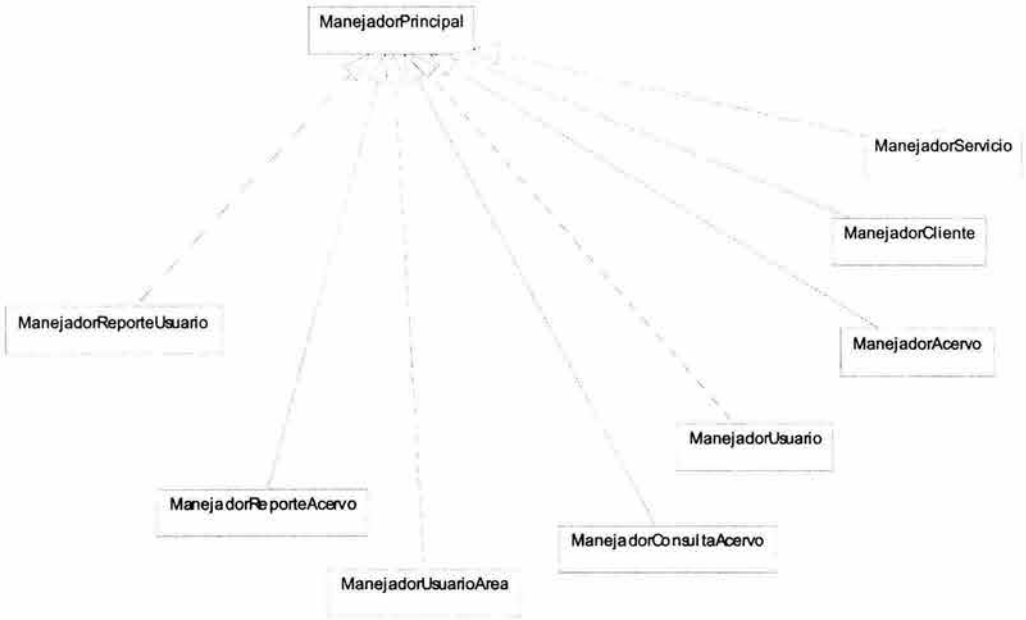
Esto es sumamente importante si se desea aprovechar el polimorfismo.

En el caso de los manejadores se observa que ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea, todos tienen una responsabilidad común "manejarEvento" correspondiente a la responsabilidad inicial "envía el evento ...". En este momento se hace la pregunta cómo se pueden simplificar estas responsabilidades y colaboraciones. Analizando las dos situaciones. En el caso de "despliega", las colaboraciones están relacionadas con diferentes pantallas, mientras que en el caso de "envía el evento ...", las colaboraciones están relacionadas con diferentes manejadores.

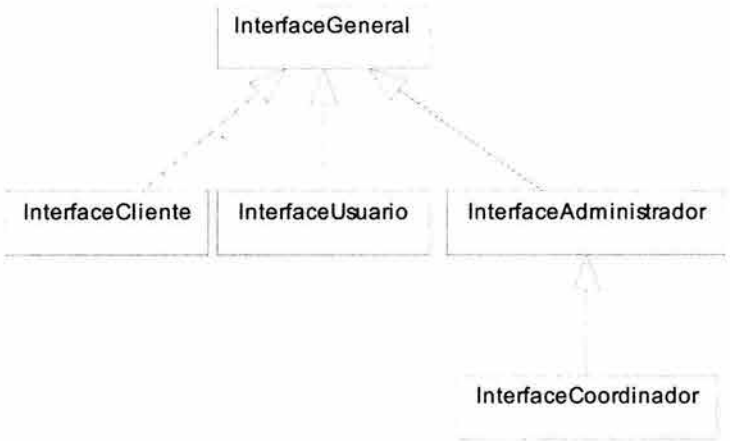
Esta es una típica situación de polimorfismo, donde una misma responsabilidad es aprovechada por diversas clases colaboradoras que funcionalmente son similares. La solución es crear nuevas superclases *PantallaGeneral y ManejadorPrincipal;* además de una superclase para las interfaces llamada *InterfaceGeneral.*

Hay que destacar que se creo la superclase PantallaGeneral que hereda a todas las pantallas del sistema.

El diagrama muestra la introducción de la superclase *PantallaGeneral* y sus respectivas subclases.



El diagrama muestra la introducción de la superclase *ManejadorPrincipal* y sus respectivas subclases.



El diagrama muestra la introducción de la superclase *InterfacePrincipal*. y sus respectivas subclases.

Es importante resaltar que se tendrá que agregar tres nuevas tarjetas de clases correspondientes a las nuevas clases *PantallaGeneral, ManejadorPrincipal e InterfaceGeneral* recién introducidas. Dichas tarjetas deberán incluir responsabilidades correspondientes a "despliega" y "manejarEvento" que serán sobrescritas por las diversas pantallas y manejadores en la jerarquía de herencia.

Se muestran a continuación las tarjetas de clase *PantallaGeneral, ManejadorPrincipal e InterfaceGeneral*:

Clase: PantallaGeneral
Descripción: superclase principal de todas las clases pantalla.
Módulo: Interfaces
Estereotipo: Borde.
Propiedades: Abstracta

Superclase:	
Subclases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones	
Atributos:	
DesplegarPantalla	
EnviarEvento	InterfaceGeneral: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador

Clase: InterfaceGeneral	
Descripción: superclase principal de todas las clases Interface.	
Módulo: Interfaces	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclase:	
Subclases: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Atributos:	
desplegarPantalla	PantallaGeneral: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones
enviarEvento	ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,

Clase: ManejadorPrincipal
Descripción: superclase principal de todas las clases Manejadores.
Módulo: ClaseControl

Estereotipo: Control	
Propiedades: Abstracta	
Superclase:	
Subclases: ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,	
Atributos:	
ManejarEvento	
DesplegarPantalla	InterfaceGeneral: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador
OfrecerServicio	ManejadorServicio

5.1.1.5 Contratos

Un *contrato* es un mecanismo de diseño para agrupar las distintas responsabilidades de una clase que están relacionadas lógicamente entre si. Los contratos sirven como indicadores de los diversos servicios provistos por cada clase. El objetivo final del contrato es ser un elemento de abstracción adicional en el manejo de la complejidad del sistema, dado que la funcionalidad completa del sistema dada a bajo nivel por las responsabilidades, puede ser vista a alto nivel como un grupo de servicios o contratos.

El contrato no es simplemente otro nombre para la responsabilidad, ya que la responsabilidad corresponde a una acción específica, mientras que un contrato define un conjunto de responsabilidades cercanas una de la otra. Cada responsabilidad puede ser parte de un sólo contrato, aunque no tiene que ser necesariamente parte de algún contrato. Esto ocurre cuando las responsabilidades representan comportamiento que una clase debe tener, pero que son privadas a los propios objetos.

Un contrato entre dos clases representa una lista de servicios que una instancia de una clase puede solicitar de una instancia de otra clase. Las responsabilidades correspondientes al contrato deben ser ofrecidas públicamente. Por lo tanto, para un mejor manejo de la complejidad del sistema, se debe posponer la definición de los aspecto privados de una clase y concentrarse inicialmente en las responsabilidades públicas.

En general, un contrato entre un cliente y un servidor no especifica cómo se hacen las cosas, solo qué se hace. Los contratos especifican quien colabora con quien, y qué se espera de la colaboración. El contrato cliente-servidor divide los objetos en dos categoría aquellos que proveen servicios (servidores), y aquellos que piden servicios (clientes).

Se puede determinar qué responsabilidades pertenecen a cuales contratos siguiendo ciertos criterios. Una forma de encontrar responsabilidades relacionadas es buscar responsabilidades que serán usadas por el mismo cliente.

En general, la mejor manera de reducir el número de contratos es buscar responsabilidades similares que puedan generalizarse. Esto también resultará en jerarquías de clases más extensibles. Una técnica para definir contratos es comenzar definiendo primero los contratos de las clases más arriba en las jerarquías. Posteriormente, se definen nuevos contratos para las subclases que agregan nueva funcionalidad. Se debe examinar las responsabilidades para cada subclase y determinar si estas representan nueva funcionalidad o si son simplemente formas específicas de expresar responsabilidades heredadas, en cuyo caso serían parte del contrato heredado.

En cada tarjeta de clase se divide la sección de responsabilidades en dos. En la parte superior se especifica una sección para los contratos, mientras que en la parte inferior se especifica una sección para las responsabilidades privadas. Cada contrato debe incluir un nombre y un número de contrato. Se debe listar cada contrato para el cual esta clase es un servidor.

También se debe listar contratos heredados, e indicar la clase de la cual se heredan, aunque no hay necesidad de repetir los detalles del contrato. Para cada contrato, se debe listar las responsabilidades de la clase en la cual se basan, asignando cada responsabilidad al contrato

correspondiente. Si la responsabilidad requiere colaborar con una clase que define varios contratos, se debe indicar el número del contrato correspondiente entre paréntesis en la columna de la colaboración, para así simplificar el seguimiento de qué responsabilidad se relaciona con qué contrato.

5.1.1.6 Subsistemas

Como se ha podido apreciar hasta el momento, la complejidad del sistema aumenta a medida que se incorporan nuevos detalles en el diseño, algo que por lo general es inevitable. Para lograr un mejor manejo de esta complejidad se introduce el concepto de *subsistemas*, el cual permite dividir el sistema completo en diversas partes. Los subsistemas permiten agrupar objetos relacionados para lograr cierta funcionalidad en "mini-sistemas". El sistema completo se compone de estos "mini-sistemas" o subsistemas y cada subsistema puede ser subdividido en subsistemas adicionales o ser definido en términos de los objetos finales.

La organización en subsistemas se logra a partir de los contratos identificadas anteriormente entre los objetos.

Externamente, los subsistemas son mecanismos de encapsulamiento, donde sus objetos cooperan para proveer una unidad de funcionalidad claramente delimitada por el subsistema. Internamente, los subsistemas pueden tener estructuras complejas, con clases colaborando entre si para satisfacer sus distintas responsabilidades contribuyendo al objetivo general del subsistema,

Se busca tener la mínima comunicación entre los diferentes subsistemas.

Un subsistema bien diseñado tiene pocas clases o subsistemas que directamente apoyan contratos, y un número mayor de colaboraciones entre clases y subsistemas internos.

Es importante distinguir entre el concepto de *módulo* y *subsistema*. Un módulo agrupa clases, correspondiente a *bibliotecas*, o sea, organizaciones estáticas definidas para almacenar y facilitar el acceso a las clases. Esto es similar al concepto de *directorios* en una computadora.

Por otro lado, el subsistema agrupa objetos, siendo una abstracción dinámica correspondiente a la funcionalidad del proceso.

Los subsistemas deben incluirse completos o no ser incluidos, de manera que un sistema pueda ser ofrecido con o sin ciertos subsistemas. Dada la dependencia entre subsistemas, si se incluye un subsistema, se debe también entregar el subsistema del cual depende.

Típicamente, los objetos de las clases entidad son reutilizados en los diversos subsistemas, mientras que los objetos de control y por lo general las de borde son propias a cierto subsistema.

Cabe resaltar que los subsistemas no existen durante la ejecución de la aplicación, son puramente abstracciones del sistema.

Para determinar los contratos apoyados por un subsistema, se tiene que encontrar todas los objetos o clases que proveen servicios a clientes fuera del subsistema. Al igual que las clases, se debe describir los contratos ofrecidos por cada subsistema. Esto se hace introduciendo el concepto de tarjeta de subsistemas de manera análoga al de clases. Dado que los subsistemas son solamente entidades conceptuales, estos no pueden directamente satisfacer ninguno de sus contratos. En su lugar, los subsistemas delegan cada contrato a una clase interna que lo satisface.

Los subsistemas se describen en *tarjetas de subsistemas*, un subsistema por tarjeta. Cada tarjeta incluye un nombre en la primera línea y dos columnas, como se puede ver en la Tabla

Subsistema: Nombre del Subsistema	
Descripción: Descripción del Subsistema	
Clases: Grupo de clases (instanciadas) que participan en este subsistema.	

Tabla Tarjeta para subsistemas.

En la columna izquierda se muestran los contratos ofrecidos de manera externa al subsistema, mientras que en la columna derecha, al lado de cada contrato se escribe la clase interna que ofrece el servicio, en otras palabras, a quien se le delega el contrato. Es importante

notar que las clases abstractas no se asignan a ningún subsistema ya que no podrían ser instanciados directamente. En su lugar asignamos únicamente clases concretas.

Antes de definir los diferentes subsistemas para el SGI - ENEO, describiremos los diagramas de colaboración a este nivel. Una de las maneras de hacerlo, y como la haremos aquí, es describir el subsistema como un rectángulo y cada uno de los contratos del subsistema como círculos. Estos contratos son asociados gráficamente con la clase servidor que implementa el contrato real mediante una relación de *realización*, como se muestra en la Figura.

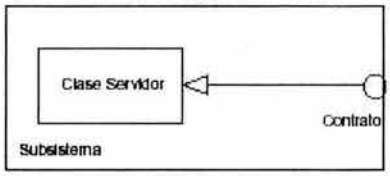


Figura Diagrama de subsistema con un contrato asociado con la clase servidor que lo implementa.

Dado que los subsistemas son puramente abstracciones, los contratos se dan entre clases clientes solicitando servicios a los subsistemas a través de sus respectivas clases servidores utilizando una flecha del ciente al servidor, como se muestra en la Figura.

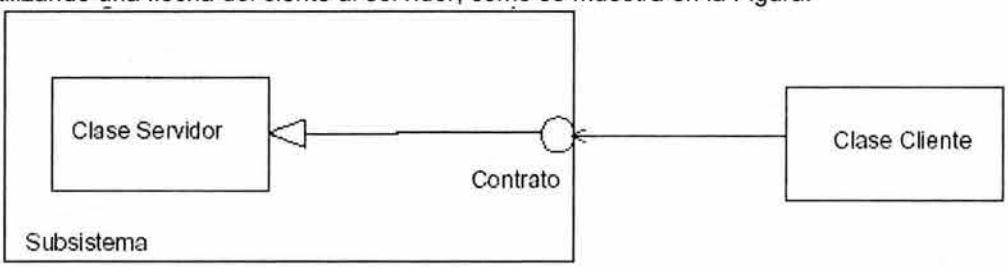


Diagrama de colaboración donde *Clase Cliente* llama al *Contrato* del *Subsistema* implementado por la *Clase Servidor.*

Si dos clases hacen solicitudes a un mismo contrato, se dibuja múltiples flechas al mismo círculo.

A continuación se identifican los subsistemas para el SGI-ENEO. Se analizaron estas consideraciones en relación a este sistema para poder identificar los subsistemas relevantes. Viendo el sistema a un alto nivel se pueden definir dos grupos de clases generales: aquellos relacionados con lo relacionado a registros y todo lo relacionado con los propios servicios del sistema.

5.1.1.7 Sistemas

Una de las ventajas de definir subsistemas es la posibilidad de visualizar el sistema completo a partir de estas.

Al incluir los subsistemas es necesario modificar las tarjetas de clase para hacer referencias a estos subsistemas y no a las clases que éstas encapsulan. Si una clase externa a un subsistema colabora con una clase dentro de un subsistema, se debe cambiar esto a una colaboración con el subsistema y no directamente con la clase. Por lo tanto, no modificaremos las tarjetas de clase con excepción a aquellas donde las colaboraciones sean con clases en otros subsistemas.

Subsistemas detectados:

Subsistema: SubsistemaInterface
Descripción: Este subsistema agrupa todos los objetos involucrados para el manejo de servicios ofrecidos
Clases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaImprimirEtiquetas, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario,

PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA,PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones, InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Contratos	Servidor
1. Desplegar Pantalla	InterfaceCliente(1), InterfaceUsuario(1), InterfaceAdministrador(1), InterfaceCoordinador(1)

Subsistema: SubsistemaServicios	
Descripción: Este subsistema agrupa los objetos generales para el manejo de los servicios	
Clases: ManejadorServicio	
Contratos	Servidor
1. Manejar evento	ManejadorServicio(1)
2.Ofrecer Servicio	ManejadorServicio(2)

Subsistema: SubsistemaManejadores	
Descripción: Este subsistema agrupa los objetos generales para el manejo eventos del sistema (excepto ManejadorServicio)	
Clases: ManejadorCliente, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea	
Contratos	Servidor
1. Manejar evento	ManejadorCliente(1), ManejadorConsultaAcervo(1), ManejadorAcervo(1), ManejadorUsuario(1), ManejadorReporteUsuario(1), ManejadorReporteAcervo(1), ManejadorUsuarioArea(1)

Tarjetas de clase después de haber agregado contratos y subsistemas.

Clase: PantallaGeneral
Descripción: superclase principal de todas las clases pantalla.
Módulo: Interfaces
Estereotipo: Borde.
Propiedades: Abstracta
Superclase:
Subclases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones

Atributos:	
Contratos	
1.Desplegar Pantalla	
DesplegarPantalla	
Responsabilidades Privadas	
EnviarEvento	InterfaceGeneral(2): InterfaceCliente(2), InterfaceUsuario(2), InterfaceAdministrador(2), InterfaceCoordinador(2)

Clase: InterfaceGeneral	
Descripción: superclase principal de todas las clases Interface.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades: Abstracta	
Superclase:	
Subclases: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Atributos:	
Contratos	
1. Desplegar Pantalla	
DesplegarPantalla	PantallaGeneral(1): PantallaOfrecerServicio(1), PantallaValidacion(1), PantallaValidaCredencial(1), PantallaConsultaAcervo(1), PantallaResultadoAcervo(1), PantallasAdministracionAcervo(1), PantallaObtenerRegistroAcervo(1), PantallaBuscarRegistroAcervo(1), PantallaDatosAcervo(1), PantallaObtenerRegistroCliente(1), PantallaBuscarRegistroCliente(1), PantallaDatosCliente(1), PantallaObtenerRegistroUsuario(1), PantallaBuscarRegistroUsuario(1), PantallaInscripcion(1), PantallaImprimirCredencial(1), PantallaReporteResultadoUsuarioUA(1), PantallaReporteGuardarUsuarioUA(1), PantallaReporteResultadoAcervoAdq(1), PantallaReporteGuardarAcervoAdq(1), PantallaDatosUsuarioArea(1), PantallaModificarUsuarioArea(1), PantallaReporteInicialUsuario(1), PantallaReporteResultadoUsuarioIC(1), PantallaReporteGuardarUsuarioIC(1), PantallaReporteInicialAcervo(1), PantallaReporteResultadoAcervoAE(1), PantallaReporteGuardarAcervoAE(1), PantallaReporteAdquisiciones(1)
2. Enviar evento	
enviarEvento	ManejadorPrincipal(1): SubsistemaServicio(1), SubsistemaManejadores(1)

Clase: ManejadorPrincipal
Descripción: superclase principal de todas las clases Manejadores.
Módulo: ClaseControl
Estereotipo: Control
Propiedades: Abstracta
Superclase:
Subclases: ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,

Atributos:	
Contratos	
1. Manejar Evento	
ManejarEvento	
Propiedades privadas	
desplegarPantalla	InterfaceGeneral(1): InterfaceCliente(1), InterfaceUsuario(1), InterfaceAdministrador(1), InterfaceCoordinador(1)
ofrecerServicio	SubsistemaServicios(2)

5.1.1.8 Protocolos

Una vez completadas las etapas anteriores, se debe detallar la especificación de cada clase hasta llegar a métodos y atributos finales. Aunque se puede generar una especificación completa a nivel de "pseudo-código" o incluso código en un lenguaje particular, esto sería sobrespecificar el diseño ya que el diseño debe definir los aspectos más relevantes de la solución pero sin ser el código final. El diseño debe dar cierta libertad al programador siempre y cuando se mantenga dentro del marco de la arquitectura de objetos generada hasta el momento. El límite de la faceta de diseño dependerá del manejo de complejidad al que se llegue.

Por tanto se continúa hasta donde se considera que la arquitectura del SGI-ENEO ya resuelve los aspectos que más afectan a toda la arquitectura. Esto incluye de manera importante la definición precisa de las responsabilidades públicas, ya que cualquier cambio en ellas afectará a todos los respectivos clientes. Para ello se define el concepto de *protocolo*, donde un *protocolo* corresponde al conjunto de *firmas* correspondientes a las distintas responsabilidades de una clase.

El objetivo de los protocolos es refinar las responsabilidades hasta llegar a métodos precisos dando especial énfasis a las responsabilidades agrupadas en los contratos ofrecidos por las clases.

A continuación se describen las clases principales incorporando protocolos en el SGI-ENEO :

InterfaceGeneral

La clase *InterfaceGeneral* define dos responsabilidades públicas, correspondientes a dos contratos.

El método "desplegarPantalla" es llamado por los diversos manejadores como parte del contrato "DesplegarPantalla". Dado que esta responsabilidad es cliente del contrato "1", con el mismo nombre, de la clase *Pantalla* y es sobrecargada por las diversas pantallas, entonces es necesario definir como parte del protocolo un parámetro correspondiente a la pantalla a ser desplegada. Este parámetro de tipo *Pantalla* corresponde a un objeto ya instanciado por parte del manejador controlador de dicha *Pantalla*. Tenemos también que definir algún tipo de resultado, de manera sencilla podemos simplemente devolver un tipo nulo ("void").

En el caso de la otra responsabilidad, "enviarEvento", la situación es la opuesta. En este caso las diversas pantallas solicitan a las distintas interfaces enviar los eventos generados por *Usuario, Clientes, Administradores y Coordinadores*. Posteriormente, las diferentes interfaces llaman al contrato "1", correspondiente a "Manejar Evento", definido en la clase *Manejador* y sobrecargado por los diferentes manejadores. En principio, deberíamos definir dos parámetros, el evento generado y otro correspondiente a la clase *Manejador* a quien se le enviará posteriormente el evento. El evento puede corresponder a una clase tipo "Evento" que deberá corresponder a algún tipo definido posteriormente y de acuerdo al lenguaje de implementación. El tipo de devolución puede ser nuevamente un "void".

Clase: InterfaceGeneral
Descripción: superclase principal de todas las clases Interface.
Módulo: Interfaces
Estereotipo: Interface

Propiedades: Abstracta	
Superclase:	
Subclases: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Atributos:	
Contratos	
1. Desplegar Pantalla	
DesplegarPantalla(Pantalla) devuelvevoid	PantallaGeneral(1): PantallaOfrecerServicio(1), PantallaValidacion(1), PantallaValidaCredencial(1), PantallaConsultaAcervo(1), PantallaResultadoAcervo(1), PantallasAdministracionAcervo(1), PantallaObtenerRegistroAcervo(1), PantallaBuscarRegistroAcervo(1), PantallaDatosAcervo(1), PantallaObtenerRegistroCliente(1), PantallaBuscarRegistroCliente(1), PantallaDatosCliente(1), PantallaObtenerRegistroUsuario(1), PantallaBuscarRegistroUsuario(1), PantallaInscripcion(1), PantallaImprimirCredencial(1), PantallaReporteResultadoUsuarioUA(1), PantallaReporteGuardarUsuarioUA(1), PantallaReporteResultadoAcervoAdq(1), PantallaReporteGuardarAcervoAdq(1), PantallaDatosUsuarioArea(1), PantallaModificarUsuarioArea(1), PantallaReporteInicialUsuario(1), PantallaReporteResultadoUsuarioIC(1), PantallaReporteGuardarUsuarioIC(1), PantallaReporteInicialAcervo(1), PantallaReporteResultadoAcervoAE(1), PantallaReporteGuardarAcervoAE(1), PantallaReporteAdquisiciones(1)
2. Enviar evento	
EnviarEvento(Evento, Manejador) devuelve void	ManejadorPrincipal(1): SubsistemaServicio(1), SubsistemaManejadores(1)

Clase InterfaceGeneral con protocolos.

PantallaGeneral

El contrato "desplegarPantalla" es llamado por las distintas clases Interface solicitando a las diversas pantallas que sobreescriben la responsabilidad desplegar. Dado que el contenido de la pantalla es conocida por las diversas pantallas, en principio, no es necesario agregar ningún parámetro. Adicionalmente, se puede devolver un tipo "void".

En el caso de la responsabilidad privada "enviarEvento", ésta es llamada localmente por lo cual se puede dejar el parámetro sin asignar y se puede devolver un tipo "void".

Clase: PantallaGeneral
Descripción: superclase principal de todas las clases pantalla.
Módulo: Interfaces
Estereotipo: Borde.
Propiedades: Abstracta
Superclase:
Subclases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente,

PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones	
Atributos:	
Contratos	
1.Desplegar Pantalla	
DesplegarPantalla() devuelve void	
Responsabilidades Privadas	
EnviarEvento() devuelve void	InterfaceGeneral(2): InterfaceCliente(2), InterfaceUsuario(2), InterfaceAdministrador(2), InterfaceCoordinador(2)

Clase PantallaGeneral con protocolos.

ManejadorPrincipal

El contrato "Manejar Evento" es llamado por las distintas clases *Interfaz*, las cuales le envian el evento generado por las diversas pantallas. Por lo tanto debe incluirse un parámetro de tipo "Evento" y nuevamente podemos asignar un tipo "void" de devolución. Las demás responsabilidades son privadas y podemos dejar de asignar parámetros y definir un tipo "void" de devolución.

Clase: ManejadorPrincipal	
Descripción: superclase principal de todas las clases Manejadores.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Abstracta	
Superclase:	
Subclases: ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,	
Atributos:	
Contratos	
1. Manejar Evento	
ManejarEvento(String Evento) devuelve void	
Propiedades privadas	
DesplegarPantalla() devuelve void	InterfaceGeneral(1): InterfaceCliente(1), InterfaceUsuario(1), InterfaceAdministrador(1), InterfaceCoordinador(1)
OfrecerServicio() devuelve void	SubsistemaServicios(2)

Clase ManejadoPrincipal con Protocolos.

ManejadorServicio

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "Ofrecer Servicio", la responsabilidad "ofrecerServicio" es llamada por el los demás manejadores, algo que inicia un servicio controlado totalmente por esta clase, por lo tanto no hay necesidad de incluir ningún parámetro e incluso se puede devolver un tipo "void".

–

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encargara de controlar las distintas pantallas para los diferentes tipos de clientes	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase:	
Subclases:	
Atributos:	
Contrato	
1. Manejar Evento	
ManejarEvento(String Evento) devuelve void	
2. Ofrecer Servicio	
OfrecerServici() devuelve void	
Responsabilidades privadas	
ConsultarAcervo() devuelve void	ManejadorConsultaAcervo(2)
ModificarAcervo() devuelve void	ManejadorAcervo(2)
ModificarCliente() devuelve void	ManejadorCliente(2)
ModificarUsuario() devuelve void	ManejadorUsuario(2)
Inscripción() devuelve void	ManejadorUsuario(2)
Validación() devuelve void	ManejadorCliente(2)
ValidaciónUsuario() devuelve void	ManejadorCliente(2)
ValidaciónAdministrador() devuelve void	ManejadorCliente(2)
ValidaciónCoordinador() devuelve void	ManejadorCliente(2)
AdministrarUsuarioPorArea() devuelve void	ManejadorUsuarioArea(2)
GenerarReporteUsuario() devuelve void	ManejadorReporteUsuario(2)
GenerarReporteAcervo() devuelve void	ManejadorReporteAcervo(2)

ManejadorCliente

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "Administrar Cliente" se tienen dos responsabilidades, *"modificarCliente", "validación".* Todas las responsabilidades son llamadas por el ManejadorServicio. En el caso de "modificarCliente" el ManejadorServicio solicita al ManejadorCliente que inicie el proceso de administración de los registros cliente (administrador, coordinador). Este proceso será controlado en su totalidad por el ManejadorCliente, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void". En el caso de *"validación" el Manejadorservicio* solicita al ManejadorCliente que inicie el proceso de validacion de los registros de algun usuario del sistema (cliente, usuario, administrador, coordinador). Este proceso será controlado en su totalidad por el ManejadorCliente, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void". En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorCliente
Descripción: El manejador Clientes se encarga de controlar los procesos asociados con el cliente.
Módulo: ClaseControl

Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
2. Administrar cliente	
ModificarCliente() devuelve void	
validación () devuelve void	
Responsabilidades privadas	
ValidarRegistroCliente() devuelve void	InterfaceBaseDatosCliente(1)
actualiza() devuelve void	InterfaceBaseDatosCliente(1)
eliminar () devuelve void	InterfaceBaseDatosCliente(1)
BuscarCliente() devuelve void	InterfaceBaseDatosCliente(1)
obtenerPrimerRegistro() devuelve void	InterfaceBaseDatosCliente(1)
obtenerUltimoRegistro() devuelve void	InterfaceBaseDatosCliente(1)
obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosCliente(1)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosCliente(1)
crearRegistroCliente () devuelve void	InterfaceBaseDatosCliente(1)
cerrarElSistema() devuelve void	

ManejadorUsuario

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador principal*, como se hizo con los demás manejadores.

En relación al contrato "Administrar Usuario" se tienen dos responsabilidades: "*modificarUsuario*", "*Inscripcion*". Todas las responsabilidades son llamadas por el ManejadorServicio. En el caso de "modificarUsuario" el ManejadorServicio solicita al ManejadorUsuario que inicie el proceso de administración de los registros usuario. Este proceso será controlado en su totalidad por el ManejadorUsuario, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void". En el caso de "Inscripcion" el ManejadorServicio solicita al ManejadorUsuario que inicie el proceso de inscripcion de un registro usuario. Este proceso será controlado en su totalidad por el ManejadorUsuario, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorUsuario	
Descripción: El manejador Usuario se encarga de controlar los procesos asociados con el usuario.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar Evento	
ManejarEvento(String Evento) devuelve void	
2. Administrar usuario	

modificarUsuario () devuelve void	
inscripcion () devuelve void	
Responsabilidades privadas	
ValidarTiempo() devuelve void	InterfaceBaseDatosCliente(2)
Actualizar() devuelve void	InterfaceBaseDatosCliente(2)
eliminar() devuelve void	InterfaceBaseDatosCliente(2)
BuscarUsuario() devuelve void	InterfaceBaseDatosCliente(2)
obtenerPrimerRegistro() devuelve void	InterfaceBaseDatosCliente(2)
obtenerUltimoRegistro() devuelve void	InterfaceBaseDatosCliente(2)
obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosCliente(2)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosCliente(2)
resinscribir() devuelve void	InterfaceBaseDatosCliente(2)
crearNuevoRegistroUsuario () devuelve void	InterfaceBaseDatosCliente(2)
imprimirEnImpresora () devuelve void	

ManejadorConsultaAcervo

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "ConsultarAcervo" se tiene una responsabilidad: "consultarAcervo", la responsabilidad es llamada por el ManejadorServicio. El ManejadorServicio solicita al ManejadorConsultaAcervo que inicie el proceso de consulta del acervo por parte de un usuario. Este proceso será controlado en su totalidad por el ManejadorConsultaAcervo, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorConsultaAcervo	
Descripción: El manejador de consultas se encargara de controlar las consultas hechas de los usuarios en el acervo.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar evento	
ManejarEvento(String Evento) devuelve void	
2. Consultar acervo	
ConsultarAcervo() devuelve void	
Propiedades privadas	
BuscarMaterial() devuelve void	InterfaceBaseDatosAcervo(3)

ManejadorAcervo

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "ModificarAcervo" se tiene una responsabilidad, "ModificarAcervo", la responsabilidad es llamada por el ManejadorServicio. El ManejadorServicio solicita al ManejadorAcervo que inicie el proceso de administracion del acervo por parte de un

administrador o coordinador. Este proceso será controlado en su totalidad por el ManejadorAcervo, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorAcervo	
Descripción: El manejador Acervo controla los registros almacenados en el acervo.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase:ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejador evento	
ManejarEvento(String Evento) devuelve void	
2.Administrar Acervo	
ModificarAcervo () devuelve void	
Responsabilidades privadas	
Actualizar() devuelve void	InterfaceBaseDatosAcervo(1)
eliminar() devuelve void	InterfaceBaseDatosAcervo(1)
BuscarAcervo() devuelve void	InterfaceBaseDatosAcervo(1)
ObtenerPrimerRegistro() devuelve void	InterfaceBaseDatosAcervo(1)
ObtenerUltimoRegistro() devuelve void	InterfaceBaseDatosAcervo(1)
obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosAcervo(1)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosAcervo(1)
crearRegistroAcervo() devuelve void	InterfaceBaseDatosAcervo(1)
imprimirEnImpresora () devuelve void	

ManejadorUsuarioArea

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "Administrar Usuario Area" se tiene una responsabilidad, "administrarUsuarioPorArea", la responsabilidad es llamada por el ManejadorServicio. El ManejadorServicio solicita al ManejadorUsuarioArea que inicie el proceso de administracion del acervo por parte de un administrador o coordinador. Este proceso será controlado en su totalidad por el ManejadorUsuarioArea, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorUsuarioArea	
Descripción: El manejador	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar evento	

ManejarEvento(String Evento) devuelve void	
2.Administrar usuario area	
AdministrarUsuarioPorArea() devuelve void	
Responsabilidades privadas	
guardar() devuelve void	InterfaceBaseDatosCliente(3)
ObtenerDatos() devuelve void	InterfaceBaseDatosCliente(3)

ManejadorReporteUsuario

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "Reporte usuario" se tiene una responsabilidad, "generarReporteUsuario", la responsabilidad es llamada por el ManejadorServicio. El ManejadorServicio solicita al ManejadorReporteUsuario que inicie el proceso de administracion de los distintos reportes por parte de un administrador o coordinador. Este proceso será controlado en su totalidad por el ManejadorReporteUsuario, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorReporteUsuario	
Descripción: El manejador Reporte Usuario se encarga de controlar los distintos reportes asociados con los usuarios generados por el sistema.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar evento	
manejarEvento(Evento) devuelve void	
2. Reporte usuario	
generarReporteUsuario() devuelve void	
Responsabilidades privadas	
obtenerDatosInsCat() devuelve void	InterfaceBaseDatosCliente(4)
ObtenerDatosInsCatG() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosInsCat() devuelve void	InterfaceBaseDatosCliente(4)
ImprimeReporteUsuarioInsCat() devuelve void	
ObtenerDatosUsuArea() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosUsuAreaG() devuelve void	InterfaceBaseDatosCliente(4)
ObtenerDatosUsuArea() devuelve void	InterfaceBaseDatosCliente(4)
ImprimeReporteUsuarioUsuArea() devuelve void	

ManejadorReporteAcervo

En el caso del contrato "Manejar Evento" se debe definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.

En relación al contrato "Reporte Acervo" se tiene una responsabilidad, "generarReporteAcervo", la responsabilidad es llamada por el ManejadorServicio. El

ManejadorServicio solicita al ManejadorReporteAcervo que inicie el proceso de administracion de los distintos reportes por parte de un administrador o coordinador. Este proceso será controlado en su totalidad por el ManejadorReporteAcervo, por lo tanto no agrega ningún parámetro y se puede devolver un tipo "void".

En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".

Clase: ManejadorReporteAcervo	
Descripción: El manejador Reporte Acervo se encarga de controlar los distintos reportes asociados con el acervo generados por el sistema.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos:	
1. Manejar evento	
ManejarEvento(Evento) devuelve void	
2. Reporte Acervo	
GenerarReporteAcervo() devuelve void	
Responsabilidades privadas	
obtenerDatosAcerExist() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAcerExistG() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAcervoAcerExist() devuelve void	InterfaceBaseDatosAcervo(2)
ImprimeReporteAcervoAcerExist() devuelve void	
obtenerDatosAdq() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAdqG() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAdq() devuelve void	InterfaceBaseDatosAcervo(2)
ImprimeReporteUsuarioAdq () devuelve void	

InterfaceBaseDatosCliente

Se definen cuatro contratos, "Administrar cliente", "Administrar usuario", "Administrar Usuario Area" y "Administrar Reporte Usuario" los cuales son encargados de interactuar con al base de datos cliente para el almacenamiento de los registros de usuario, cliente y generación de los distintos reportes relacionados con los usuarios, respectivamente. Todas las responsabilidades deben enviar parámetros correspondientes a los registros a ser guardados en la base de datos cliente o incluso a ser devueltos, ya que podemos hacer esto último a través de los parámetros.

En el caso del contrato "Administrar Cliente" se tendrán como parámetro a la clase *RegistroCliente* para las diversas responsabilidades con excepción de validarRegistroCliente. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que buscamos. En el caso de "validarRegistroCliente", se tiene que enviar dos parámetros de tipo "String" correspondientes al tipo de usuario y a su login. En general podemos devolver un tipo "boolean" con excepción de "validarRegistroCliente" que debe devolver un "boolean".

En el caso del contrato "Administrar Usuario" se tendría como parámetro a la clase *RegistroUsuario* para las diversas responsabilidades con excepción de validarTiempo. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que buscamos. En el caso de "validarTiempo", se tiene que enviar un parámetro de tipo "String" correspondiente a su password.

En general se puede devolver un tipo "boolean" con excepción de "validarTiempo" que debe devolver un "boolean".

En el caso del contrato "Administrar Usuario Area" se tendría como parámetro a la clase *UsuarioPorArea* para las diversas responsabilidades. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que se busca.

En general se puede devolver un tipo "boolean" .

En el caso del contrato "Administrar Reporte Usuario" se tendría como parámetro a la clase RegistroUsuario para las diversas responsabilidades. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que buscamos.

En general se puede devolver un tipo "boolean .

Clase: InterfaceBaseDatosCliente	
Descripción: La información de cada usuario se almacena en la base de datos cliente la cual se accesa mediante la interface de la base de datos cliente. Esto permite validar a los distintos usuarios además de guardar información sobre estos.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades: Concreta	
Superclase:	
Subclases:	
Atributos:	
1. Administrar cliente	
ValidarRegistroCliente(String, String) devuelve boolean	BaseDatosCliente
ActualizarCliente(RegistroCliente) devuelve boolean	BaseDatosCliente
eliminarCliente(RegistroCliente) devuelve boolean	BaseDatosCliente
BuscarCliente(RegistroCliente) devuelve boolean	BaseDatosCliente
obtenerPrimerRegistroCliente(RegistroCliente) devuelve String	BaseDatosCliente
obtenerUltimoRegistroCliente(RegistroCliente) devuelve String	BaseDatosCliente
obtenerRegistroSiguienteCliente(RegistroCliente) devuelve String	BaseDatosCliente
obtenerRegistroAnteriorCliente(RegistroCliente) devuelve String	BaseDatosCliente
*CrearRegistro*Cliente*(RegistroCliente)* devuelve boolean	BaseDatosCliente
2. Administrar usuario	
ValidarTiempo (String) devuelve boolean	BaseDatosCliente
ActualizarUsuario(RegistroUsuario) devuelve boolean	BaseDatosCliente
eliminarusuario(RegistroUsuario) devuelve boolean	BaseDatosCliente
buscarUsuario (RegistroUsuario) devuelve boolean	BaseDatosCliente
obtenerPrimerRegistroUsuario(RegistroUsuario) devuelve String	BaseDatosCliente
ObtenerUltimoRegistroUsuario(RegistroUsuario) devuelve String	BaseDatosCliente
obtenerRegistroSiguienteUsuario(RegistroUsuario) devuelve String	BaseDatosCliente
obtenerRegistroAnteriorUsuario(RegistroUsuario) devuelve String	BaseDatosCliente
resinscribir(RegistroUsuario) devuelve boolean	BaseDatosCliente
CrearNuevoRegistroUsuario(RegistroUsuario) devuelve boolean	BaseDatosCliente
3. Administrar usuario area	
Guardar(UsuarioPorArea) devuelve boolean	BaseDatosCliente
ObtenerDatos(UsuarioPorArea) devuelve String	BaseDatosCliente

4. Administrar reporte usuario	
ObtenerDatosInsCat(ReporteUsuario) devuelve String[]	BaseDatosCliente
ObtenerDatosInsCatG(ReporteUsuario) devuelve String[]	BaseDatosCliente
obtenerDatosInsCat(ReporteUsuario) devuelve String[]	BaseDatosCliente
ObtenerDatosUsuArea(ReporteUsuario) devuelve String[]	BaseDatosCliente
obtenerDatosUsuAreaG(ReporteUsuario) devuelve String[]	BaseDatosCliente
obtenerDatos(ReporteUsuario) devuelve String[]	BaseDatosCliente

InterfaceBaseDatosAcervo

Se definen tres contratos, "Administrar acervo", "Administrar Reporte Acervo" y "Buscar Material". los cuales son encargados de interactuar con al base de datos acervo para el almacenamiento de los registros de acervo y generación de los distintos reportes relacionados con el acervo, respectivamente. Todas las responsabilidades deben enviar parámetros correspondientes a los registros a ser guardados en la base de datos acervo o incluso a ser devueltos, ya que se puede hacer esto último a través de los parámetros.

En el caso del contrato "Administrar acervo" se tendría como parámetro a la clase *RegistroAcervo* para las diversas responsabilidades. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que se busca. En general se puede devolver un tipo "boolean".

En el caso del contrato "Administrar Reporte Acervo" se tendría como parámetro a la clase RegistroAcervo para las diversas responsabilidades. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que se busca. En general se puede devolver un tipo "boolean" .

En el caso del contrato "Buscar Material" tendríamos como parámetro a la clase *RegistroAcervo* para las diversas responsabilidades. En el caso de las distintas responsabilidades para obtener los registros es necesario también enviar algún identificador que especifique el registro particular que se busca. En general se puede devolver un tipo "boolean".

Clase: InterfaceBaseDatosAcervo	
Descripción: La información de cada registro acervo se almacena en la base de datos acervo la cual se accesa mediante la interface de la base de datos acervo. Permite hacer consultas, crear, actualizar registros del acervo.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades: Concreta	
Superclase:	
Subclases:	
Atributos:	
1. Administrar acervo	
Actualizar(RegistroMaterialAcervo) devuelve bolean	BaseDatosAcervo
Eliminar(RegistroMaterialAcervo) devuelve boolean	BaseDatosAcervo
BuscarAcervo(RegistroMaterialAcervo) devuelve boolean	BaseDatosAcervo
obtenerPrimerRegistro(RegistroMaterialAcervo) devuelve String	BaseDatosAcervo
obtenerUltimoRegistro(RegistroMaterialAcervo) devuelve String	BaseDatosAcervo
obtenerRegistroSiguiente(RegistroMaterialAcervo) devuelve String	BaseDatosAcervo
ObtenerRegistroAnterior(RegistroMaterialAcervo) devuelve String	BaseDatosAcervo
crearRegistroAcervo(RegistroMaterialAcervo) devuelve bolean	BaseDatosAcervo
2.Administrar reporte acervo	
ObtenerDatosAcerExist(ReporteAcervo) devuelve String[]	BaseDatosAcervo
ObtenerDatosAcerExistG(ReporteAcervo) devuelve String[]	BaseDatosAcervo

obtenerDatosAcerExist(ReporteAcervo) devuelve String[]	BaseDatosAcervo
obtenerDatosAdqReporte(ReporteAcervo) devuelve String[]	BaseDatosAcervo
obtenerDatosAdqG(ReporteAcervo) devuelve String[]	BaseDatosAcervo
ObtenerDatosAdqImprimir(ReporteAcervo) devuelve String[]	BaseDatosAcervo
3. Buscar material	
buscarMaterial(RegistroMaterialAcervo) devuelve boolean	BaseDatosAcervo

La descripción de las distintas clases *Pantalla* se mantiene de manera similar a la clase PantallaGeneral.

5.1.1.9 ATRIBUTOS

En las secciones anteriores se ha diseñado las clases hasta llegar a los protocolos de cada una de ellas. Estos protocolos describen las firmas para las responsabilidades, en otras palabras, las interfaces externas de la clase. Es difícil decidir en que momento termina el diseño y comienza la implementación. El objetivo general es lograr un diseño robusto que le de cierta flexibilidad al programador pero sin modificar de manera importante la arquitectura de diseño. Lo que se hará a continuación, es detallar un poco más nuestro diseño especificando *atributos* que definan la manera de implementar las diversas responsabilidades correspondientes a los múltiples contratos de la aplicación.

En general, los *atributos* corresponden a los aspectos estructurales de las clases, como son los valores (números y textos), junto con las referencias a otros objetos. Estos conceptos varían de gran manera entre lenguajes. Los atributos correspondientes a referencias deben agregarse de acuerdo a las colaboraciones establecidas durante el diseño. Los atributos que guardan valores son lo último que se especifica en el diseño de objetos.

Se muestran las clases mas representativas con sus atributos.

InterfaceGeneral

Como se pudo observar anteriormente, la clase *InterfaceGeneral* se relaciona primordialmente y a través de polimorfismo con las diversas pantallas y manejadores. Dado que sólo se desplegará una pantalla a la vez y se comunicará con un solo manejador en un momento específico, es suficiente definir dos tipos de referencias, *ManejadorPrincipal* para poder comunicarse con los diversos manejadores y *PantallaGeneral* para comunicarse con las diversas pantallas. Por lo tanto, modificamos la tarjeta de clase correspondiente a la *InterfaceGeneral*.

Clase: InterfaceGeneral	
Descripción: superclase principal de todas las clases Interface.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades: Abstracta	
Superclase:	
Subclases: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Atributos: ManejadorPrincipal, PantallaGeneral	
Contratos	
1. Desplegar Pantalla	
DesplegarPantalla(Pantalla) devuelvevoid	PantallaGeneral(1): PantallaOfrecerServicio(1), PantallaValidacion(1), PantallaValidaCredencial(1), PantallaConsultaAcervo(1), PantallaResultadoAcervo(1), PantallasAdministracionAcervo(1), PantallaObtenerRegistroAcervo(1),

	PantallaBuscarRegistroAcervo(1), PantallaDatosAcervo(1), PantallaObtenerRegistroCliente(1), PantallaBuscarRegistroCliente(1), PantallaDatosCliente(1), PantallaObtenerRegistroUsuario(1), PantallaBuscarRegistroUsuario(1), PantallaInscripcion(1), PantallaImprimirCredencial(1), PantallaReporteResultadoUsuarioUA(1), PantallaReporteGuardarUsuarioUA(1), PantallaReporteResultadoAcervoAdq(1), PantallaReporteGuardarAcervoAdq(1), PantallaDatosUsuarioArea(1), PantallaModificarUsuarioArea(1), PantallaReporteInicialUsuario(1), PantallaReporteResultadoUsuarioIC(1), PantallaReporteGuardarUsuarioIC(1), PantallaReporteInicialAcervo(1), PantallaReporteResultadoAcervoAE(1), PantallaReporteGuardarAcervoAE(1), PantallaReporteAdquisiciones(1)
2. Enviar evento	
EnviarEvento(Evento, Manejador) devuelve void	ManejadorPrincipal(1): SubsistemaServicio(1), SubsistemaManejadores(1)

PanatallaGeneral

De manera similar a la *InterfaceGeneral*, las diversas pantallas deben tener conocimiento de la *Interfaces* y de los manejadores que las controlan. Aunque se pudiera especificar el tipo particular de manejador que administra una pantalla en particular, se puede aprovechar y definir una referencia de tipo genérica a *ManejadorPrincipal* ya que cada pantalla es administrada por un solo manejador. Este conocimiento se implementa mediante atributos correspondientes a las clases anteriores.

Clase: PantallaGeneral	
Descripción: superclase principal de todas las clases pantalla.	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades: Abstracta	
Superclase:	
Subclases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones	
Atributos: InterfaceGeneral, ManejadorPrincipal	
Contratos	
1.Desplegar Pantalla	
DesplegarPantalla() devuelve	

void	
Responsabilidades Privadas	
EnviarEvento() devuelve void	InterfaceGeneral(2): InterfaceCliente(2), InterfaceUsuario(2), InterfaceAdministrador(2), InterfaceCoordinador(2)

ManejadorPrincipal

De manera similar a las Interfaces y las pantallas, los manejadores deben poder acceder a las Interfaces y también a sus diferentes pantallas. Por tal motivo se definen dos atributos, uno de tipo *InterfaceGeneral* y otro de tipo *PantallaGeneral* correspondiente a la pantalla actualmente siendo desplegada. Si revisamos un poco más, podemos apreciar que la gran mayoría de los menajadores requieren comuincarse con el *SubsistemaServicios* para ofrecer algún servicio. Por lo tanto, también se requiere un atributo de referencia de tipo *ManejadorServicio* el cual puede ser definido a este nivel en la superclase *Manejador*. Adicionalmente, vamos a agregar una referencia genérica para el manejador *padre* el cual reside a un nivel administrativo superior, en otras palabras, el manejador encargado de instanciar al siguiente nivel de manejadores y así sucesivamente. Dado que el manejador *padre* puede tener cualquier tipo dentro de la jerarquía de manejadores, lo agregaremos bajo el tipo genérico *ManejadoPrincipal*. Esta referencia es siempre buena tener, ya que un manejador particular puede tener acceso a cualquier otro manejador siempre y cuando no pierda la referencia a su superior.

Clase: ManejadorPrincipal	
Descripción: superclase principal de todas las clases Manejadores.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Abstracta	
Superclase:	
Subclases: ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,	
Atributos: PantallaGeneral, InterfaceGeneral, ManejadorPrincipal, ManejadorServicio	
Contratos	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
Propiedades privadas	
DesplegarPantalla() devuelve void	InterfaceGeneral(1): InterfaceCliente(1), InterfaceUsuario(1), InterfaceAdministrador(1), InterfaceCoordinador(1)
OfrecerServicio() devuelve void	SubsistemaServicios(2)

ManejadorServicio

En el caso de la clase *ManejadorServicio* es necesario agregar una referencia particular a cada pantalla y manejador que pueda ser accesado desde este clase, tomando en cuenta los ya definidos a nivel de la superclase *Manejador*.

Clase: ManejadorServicio
Descripción: El manejador de servicios se encargara de controlar las distintas pantallas para los diferentes tipos de clientes
Módulo: ClaseControl
Estereotipo: Control
Propiedades: Concreta
Superclase:
Subclases:

Atributos: ManejadorCliente, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea, PantallaValidaCredencial, PantallaOfrecerServicio, InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Contrato	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
2. Ofrecer Servicio	
OfrecerServici() devuelve void	
Responsabilidades privadas	
ConsultarAcervo() devuelve void	ManejadorConsultaAcervo(2)
ModificarAcervo() devuelve void	ManejadorAcervo(2)
ModificarCliente() devuelve void	ManejadorCliente(2)
ModificarUsuario() devuelve void	ManejadorUsuario(2)
Inscripción() devuelve void	ManejadorUsuario(2)
Validación() devuelve void	ManejadorCliente(2)
ValidaciónUsuario() devuelve void	ManejadorCliente(2)
ValidaciónAdministrador() devuelve void	ManejadorCliente(2)
ValidaciónCoordinador() devuelve void	ManejadorCliente(2)
AdministrarUsuarioPorArea() devuelve void	ManejadorUsuarioArea(2)
GenerarReporteUsuario() devuelve void	ManejadorReporteUsuario(2)
GenerarReporteAcervo() devuelve void	ManejadorReporteAcervo(2)

ManejadorCliente

Clase: ManejadorCliente	
Descripción: El manejador Clientes se encarga de controlar los procesos asociados con el cliente.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos: PantallaValidacion, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, InterfaceCliente, InterfaceCoordinador, InterfaceAdministrador. InterfaceBaseDatosCliente	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
2. Administrar cliente	
ModificarCliente() devuelve void	
validación () devuelve void	
Responsabilidades privadas	
ValidarRegistroCliente() devuelve void	InterfaceBaseDatosCliente(1)
actualiza() devuelve void	InterfaceBaseDatosCliente(1)
eliminar () devuelve void	InterfaceBaseDatosCliente(1)
BuscarCliente() devuelve void	InterfaceBaseDatosCliente(1)
obtenerPrimerRegistro() devuelve void	InterfaceBaseDatosCliente(1)
obtenerUltimoRegistro() devuelve void	InterfaceBaseDatosCliente(1)

obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosCliente(1)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosCliente(1)
*crearRegistro*Cliente *() devuelve void*	InterfaceBaseDatosCliente(1)
cerrarElSistema() devuelve void	

ManejadorUsuario

Clase: ManejadorUsuario	
Descripción: El manejador Usuario se encarga de controlar los procesos asociados con el usuario.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos: PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaImprimirCredencial, PantallaInscripcion, InterfaceCoordinador, InterfaceAdministrador. InterfaceBaseDatosCliente	
1. Manejar Evento	
ManejarEvento	
2. Administrar usuario	
modificarUsuario () devuelve void	
inscripcion () devuelve void	
Responsabilidades privadas	
ValidarTiempo() devuelve void	InterfaceBaseDatosCliente(2)
Actualizar() devuelve void	InterfaceBaseDatosCliente(2)
eliminar() devuelve void	InterfaceBaseDatosCliente(2)
BuscarUsuario() devuelve void	InterfaceBaseDatosCliente(2)
obtenerPrimerRegistro() devuelve void	InterfaceBaseDatosCliente(2)
obtenerUltimoRegistro() devuelve void	InterfaceBaseDatosCliente(2)
obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosCliente(2)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosCliente(2)
resinscribir() devuelve void	InterfaceBaseDatosCliente(2)
crearNuevoRegistroUsuario () devuelve void	InterfaceBaseDatosCliente(2)
imprimirEnImpresora () devuelve void	

ManejadorConsultaAcervo

Clase: ManejadorConsultaAcervo	
Descripción: El manejador de consultas se encargara de controlar las consultas hechas de los usuarios en el acervo.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributo: PantallaConsultaAcervo, PantallaResultadoAcervo, InterfaceUsuario, InterfaceBaseDatosAcervo	
1. Manejar evento	

ManejarEvento(Evento) devuelve void	
2. Consultar acervo	
ConsultarAcervo() devuelve void	
Propiedades privadas	
BuscarMaterial() devuelve void	InterfaceBaseDatosAcervo(3)

ManejadorAcervo

Clase: ManejadorAcervo	
Descripción: El manejador Acervo controla los registros almacenados en el acervo.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase:ManejadorPrincipal	
Subclases:	
Atributos: PantallaObtenerRegistroUsuario, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaImprimirEtiquetas, InterfaceAdministrador, InterfaceCoordinador, InterfaceBaseDatosAcervo	
1. Manejador evento	
ManejarEvento(Evento) devuelve void	
2.Administrar Acervo	
ModificarAcervo () devuelve void	
Responsabilidades privadas	
Actualizar() devuelve void	InterfaceBaseDatosAcervo(1)
eliminar() devuelve void	InterfaceBaseDatosAcervo(1)
BuscarAcervo() devuelve void	InterfaceBaseDatosAcervo(1)
ObtenerPrimerRegistro() devuelve void	InterfaceBaseDatosAcervo(1)
ObtenerUltimoRegistro() devuelve void	InterfaceBaseDatosAcervo(1)
obtenerRegistroSiguiente() devuelve void	InterfaceBaseDatosAcervo(1)
obtenerRegistroAnterior() devuelve void	InterfaceBaseDatosAcervo(1)
crearRegistroAcervo() devuelve void	InterfaceBaseDatosAcervo(1)
imprimirEnImpresora () devuelve void	

ManejadorUsuarioArea

Clase: ManejadorUsuarioArea	
Descripción: El manejador	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos: PantallaUsuarioArea, PantallaModificarUsuarioArea, InterfaceAdministrador, InterfaceCoordinador, InterfaceBaseDatosCliente	
1. Manejar evento	
ManejarEvento(Evento) devuelve void	
2.Administrar usuario area	
AdministrarUsuarioPorArea() devuelve void	
Responsabilidades privadas	

guardar() devuelve void	InterfaceBaseDatosCliente(3)
ObtenerDatos() devuelve void	InterfaceBaseDatosCliente(3)

ManejadorReporteUsuario

Clase: ManejadorReporteUsuario	
Descripción: El manejador Reporte Usuario se encarga de controlar los distintos reportes asociados con los usuarios generados por el sistema.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos: PantallaReporteInicialUsuario, PantallaDatosInsCat, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, InterfaceAdministrador, InterfaceCoordinador, InterfaceBaseDatosCliente	
1. Manejar evento	
manejarEvento(Evento) devuelve void	
2. Reporte usuario	
generarReporteUsuario() devuelve void	
Responsabilidades privadas	
obtenerDatosInsCat() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosInsCatG() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosInsCat() devuelve void	InterfaceBaseDatosCliente(4)
ImprimeReporteUsuarioInsCat() devuelve void	
obtenerDatosUsuArea() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosUsuAreaG() devuelve void	InterfaceBaseDatosCliente(4)
obtenerDatosUsuArea() devuelve void	InterfaceBaseDatosCliente(4)
ImprimeReporteUsuarioUsuArea() devuelve void	

ManejadorReporteAcervo

Clase: ManejadorReporteAcervo	
Descripción: El manejador Reporte Acervo se encarga de controlar los distintos reportes asociados con el acervo generados por el sistema.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase: ManejadorPrincipal	
Subclases:	
Atributos: PantallaReporteInicialAcervo, PantallaDatosAcervExist, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, InterfaceAdministrador, InterfaceCoordinador, InterfaceBaseDatosAcervo	
1. Manejar evento	
ManejarEvento(Evento) devuelve void	
2. Reporte Acervo	

GenerarReporteAcervo() devuelve void	
Responsabilidades privadas	
obtenerDatosAcerExist() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAcerExistG() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAcervoAcerExist() devuelve void	InterfaceBaseDatosAcervo(2)
ImprimeReporteAcervoAcerExist() devuelve void	
obtenerDatosAdq() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAdqG() devuelve void	InterfaceBaseDatosAcervo(2)
obtenerDatosAdq() devuelve void	InterfaceBaseDatosAcervo(2)
ImprimeReporteUsuarioAdq () devuelve void	

CAPÍTULO 6
MODELO DE IMPLEMENTACIÓN

6.1 IMPLEMENTACIÓN

El modelo de implementación toma el resultado del *Modelo de Diseño* para generar el código final. Esta traducción debe ser relativamente sencilla y directa, ya que las decisiones mayores han sido tomadas durante las etapas previas.

Durante el *Modelo de Implementación* se hace una adaptación al lenguaje de programación y/o la Base de Datos de acuerdo a la especificación del diseño y según las propiedades del lenguaje de implementación y Base de Datos.

Aunque el diseño de objetos es bastante independiente del lenguaje actual, todos los lenguajes tendrán sus particularidades, las cuales deberán adecuarse durante la implementación final. La elección del lenguaje influye en el diseño, pero el diseño no debe depender de los detalles del lenguaje. Si se cambia de lenguaje de programación no debe requerirse el re-diseño del sistema.

El desarrollo del SGI-ENEO, se ha basado en los capítulos anteriores, abarcando desde el capítulo de *Modelo de Requisitos* hasta el capítulo del *Modelo de Diseño*. Se han atendido todas y cada uno de los requisitos del Cliente para la implementación del sistema.

La implementación del Sistema de Gestión de Información, se ha llevado a cabo en lenguaje JAVA, desarrollando toda la capa de presentación (borde) con las bibliotecas del paquete JAVA SWING que vienen incluidas en sdk proporcionado por SUN.

Para implementar el SGI-ENEO se ha empleado el software JBuilder 6.0.

Para el diseño de la Base de Datos se contó con la herramienta PowerDesigner 9.1, en esta se pudo desarrollar algunos atributos de las clases que se originaban de los casos de uso y las clases del SGI-ENEO.

Para la implementación de *casos de uso* se tomo como referencia los capítulos referentes a los Modelos de Requisitos, Análisis y Diseño de esta Tesis, en el cuál se obtienen detalles específicos de cada *caso de uso*, por lo cual en este capítulo solo se implementa su programación y su relación con la Base de Datos

En las páginas siguientes se hace mención de la implementación de algunos *casos de uso*, sin embargo, es importante mencionar que se siguió la misma métodología para todos, apoyándose también en la tecnología UML.

6.1.1 IMPLEMENTACIÓN DEL SISTEMA

En el capítulo tres (Requisitos) se detectaron tres actores primarios: Coordinadores, Administradores y Usuarios, y son ellos quienes interactuan con el sistema a través de pantallas las cuales fueron definidas en los capítulos de Análisis e Implementación. Revisando los casos de uso (diagramas) se observa que todos incluyen el caso de uso Ofrecer Servicios (Figura 1). El propósito de este caso es el proporcionar una pantalla en la que se presentará los servicios dependiendo del tipo de Cliente (Usuario, Administrador, Coordinador).

Caso de uso:	**Ofrecer Servicios**
Actores:	Cliente, Usuario, Administrador, Coordinador
Tipo:	Inclusión
Propósito:	Proporciona una pantalla en la que se presentará los servicios dependiendo del tipo de cliente (Usuario, Administrador, Coordinador)
Resumen:	Este caso es iniciado por un cliente. Muestra las opciones para utilizar el sistema SGI – ENEO.
Precondiciones:	Debe de haberse validado el cliente en todos los subflujos excepto el S – 1.
Flujo principal:	Se ejecuta el caso de uso Ofrecer Servicios y dependiendo del tipo de cliente, se continuara con los diversos subflujos de este caso de uso.
Subflujos:	S – *1 Sin Servicio.* Se presenta al cliente la pantalla P – 2. El cliente puede seleccionar entre las siguientes opciones de un menú desplegable llamado Inicio, dejando inhabilitados el resto de los menús. El cliente puede seleccionar de este menú las siguientes opciones: "Validación" y "Salir". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. Si la actividad seleccionada es "Salir" sale del sistema.
	S – 2 *Ofrecer Servicios a Usuarios.* Se presenta al usuario la pantalla P–2. El usuario puede seleccionar los siguientes menús desplegables: Inicio, Usuarios, dejando inhabilitados el resto de los menús. El usuario puede seleccionar del menú Inicio la opción: "Validación". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. El usuario puede seleccionar del menú Usuario las siguientes opciones: "Validar Usuario", "Consultar Acervo". Si la actividad seleccionada es "Validar Usuario" se continúa con el caso *Validar Usuario*. Si la actividad seleccionada es "Consultar Acervo" se continúa con el caso de uso *Consultar Acervo*.
	S – 3 *Ofrecer Servicios Administradores* Se presenta al Administrador la pantalla P–2. El Administrador puede seleccionar los siguientes menús desplegables: Inicio, Administrador, dejando inhabilitados el resto de los menús. El Administrador puede seleccionar del menú Inicio las siguientes opciones: "Validación" y "Salir". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. Si la actividad seleccionada es "Salir" sale del sistema. El administrador puede seleccionar del menú Administradores las siguientes opciones: "Administrar Acervo(AA)", "Administrar Usuario(AU)", "Administrar Horarios de Asesorías(AH)", "Reporte de Acervo(RA)", "Reporte Usuarios (RU) " y "Administrar Usuarios por Área (AU)". Si la actividad seleccionada es "AA – Administrar acervo" se continúa con el caso de uso *Administrar Acervo*. Si la actividad seleccionada es "AU – Inscripción" se continúa con el caso de uso *Inscripción*. Si la actividad seleccionada es "AU – Administrar usuario" se continúa con el caso de uso *Administrar Usuario*. Si la actividad seleccionada es "R - Usuario" se continúa con el caso de uso *Generar Reporte Usuario*. Si la actividad seleccionada es "R - Acervo" se continúa con el caso de uso *Generar Reporte Acervo*. Si la actividad seleccionada es "AU – Administrar Usuarios por área" se continúa con el caso de uso *Capturar Usuarios por Área*.
	S – 4 *Ofrecer servicios Coordinadores* Se presenta al Coordinador la pantalla P–2. El coordinador puede seleccionar los siguientes menús desplegables: Inicio, Coordinador, dejando inhabilitados el resto de los menús. El Coordinador puede seleccionar las mismas opciones que el administrador pero se agrega un nuevo item "Administrar cliente".
Excepciones:	Ninguno

Figura 1.

La tarjeta de clase de la pantalla ofrecer servicios es la siguiente:

Clase: PantallaOfrecerServicio.	
Descripción Pantalla en la que se presentará los servicios dependiendo del tipo de cliente (Usuario, Administrador, Coordinador).	
Módulo: Interfaces	
Estereotipo: Borde.	
Propiedades: Concreta	
Superclase: PantallaGeneral	
Subclases:	
Atributos:	
EnviarEvento	InterfaceUsuario

Figura 2.

La PantallaOfrecerServicios tiene como superclase a PantallaGeneral (Figura 3). Esta clase es la superclase de todas las pantallas del sistema y es quien proporciona los mecanismos basicos de configuración, despliegue y cierre de pantalla. PantallaGeneral tiene como atributos a las clases InterfaceGeneral y ManejadorPrincipal.

Clase: PantallaGeneral
Descripción: superclase principal de todas las clases pantalla.
Módulo: Interfaces
Estereotipo: Borde.
Propiedades: Abstracta

Superclase:	
Subclases: PantallaOfrecerServicio, PantallaValidacion, PantallaValidaCredencial, PantallaConsultaAcervo, PantallaResultadoAcervo, PantallasAdministracionAcervo, PantallaObtenerRegistroAcervo, PantallaBuscarRegistroAcervo, PantallaDatosAcervo, PantallaObtenerRegistroCliente, PantallaBuscarRegistroCliente, PantallaDatosCliente, PantallaObtenerRegistroUsuario, PantallaBuscarRegistroUsuario, PantallaInscripcion, PantallaImprimirCredencial, PantallaReporteResultadoUsuarioUA, PantallaReporteGuardarUsuarioUA, PantallaReporteResultadoAcervoAdq, PantallaReporteGuardarAcervoAdq, PantallaDatosUsuarioArea, PantallaModificarUsuarioArea, PantallaReporteInicialUsuario, PantallaReporteResultadoUsuarioIC, PantallaReporteGuardarUsuarioIC, PantallaReporteInicialAcervo, PantallaReporteResultadoAcervoAE, PantallaReporteGuardarAcervoAE, PantallaReporteAdquisiciones	
Atributos: InterfaceGeneral, ManejadorPrincipal	
Contratos	
1.Desplegar Pantalla	
DesplegarPantalla() devuelve void	
Responsabilidades Privadas	
EnviarEvento() devuelve void	InterfaceGeneral(2): InterfaceCliente(2), InterfaceUsuario(2), InterfaceAdministrador(2), InterfaceCoordinador(2)

Figura 3.

Se analizará la implementación final de la clase PantallaGeneral.

Se indica el paquete a que pertenecera la clase.

```
package tesis;
```

Se importan las bibliotecas necesarias para la construcción de la clase.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
```

Se inicia el cuerpo de la clase definiéndola de tipo abstracto, es decir, no se pueden crear instancias de esta clase. Un aspecto importante es que esta clase no hereda de ninguna clase tipo gráfico contenedor (JFrame, JDialog, etc.), pues el contenedor será la interface que solicite a las pantallas como se verá posteriormente.

```
public abstract class PantallaGeneral {
```

Variables de instancia y de clase.

```
  protected InterfaceGeneral InterfaceGeneral;
  protected ManejadorGeneral ManejadorGeneral;
  public static boolean banderaservicio = false;
  public static boolean sinDatos = true;
  String consulta;
```

El constructor recibe como parámetros dos objetos de tipo InterfaceGeneral y ManejadorGeneral estos son asignados a las respectivas variables de instancia.

```
  public PantallaGeneral(InterfaceGeneral ig, ManejadorGeneral mg) {
        InterfaceGeneral =ig;
        ManejadorGeneral = mg;
  }
```

El siguiente método es la implementación para desplegar la pantalla, que corresponde al contrato 1 con las interfaces.

```
  public void desplegarPantalla(){
```

Se verifica si se trata de una pantalla normal o de la pantalla ofrecer servicios. Si es una pantalla normal se agrega al panel de contenido de la interface (que es de tipo JFrame) solicitante.

```
        if (banderaservicio && sinDatos){
          InterfaceGeneral.setContentPane(crearPantalla());
        }
```

Si se trata de la pantalla ofrecer servicios se agrega un menú a la interface solicitante.

```
        if(banderaservicio ==false){
          InterfaceGeneral.setJMenuBar((JMenuBar)crearPantalla());
        }
  }
```

El siguiente método no se encuentra definido en la tarjeta de clase, pero fue agregado debido a que se detecto que todas las clases borde lo utilizan. Por medio de este método posicionamos nuestras ventanas en el centro de la Pantalla con excepcion de la PantallaOfrecerServicio.

```
  public void tamañoVentana(int ancho, int alto){
        InterfaceGeneral.setSize(ancho, alto);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = InterfaceGeneral.getSize();
        if (frameSize.height > screenSize.height) {
          frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
          frameSize.width = screenSize.width;
        }
        InterfaceGeneral.setLocation((screenSize.width - frameSize.width) / 2,
                                     (screenSize.height - frameSize.height) / 2);
  }
```

Los siguientes métodos son abstractos y por lo tanto todas las clases que hereden de la clase PantallaOfrecerServicios deben implementarlos de acuerdo a las necesidades de la clase heredada.

Este método es llamado por el método desplegarPantalla. Su funcion es crear los componentes gráficos de la pantalla a desplegar.

```
  public abstract Container crearPantalla();
```

El método obtenerAtributosPantalla enviara una cadena (String) a la clase InterfaceBaseDatosCliente ó InterfaceBaseDatosAcervo, esta clase hará las operaciones necesarias para obtener la información que necesite la pantalla (llenado de Combo box, campos de texto, etc) la información ó las herramientas con las cuales estas clases reponderan a la petición son enviadas a traves del método asignarAtributosPantalla

```
  public abstract String obtenerAtributosPantalla();

  public abstract void asignarAtributosPantalla(ResultSet rs, Statement stmt);
```

Los últimos dos métodos son empleados para el cierre de las pantallas.

```
  public abstract void cerrarPantalla();

  protected abstract void cerradoPantallaTecla(KeyEvent e, String s);
```

Se cierra el cuerpo de la clase.

```
}
```

Como se mencionó anteriormente, las subclases de PantallaGeneral, no pueden desplegar los componentes gráficos de las pantallas (creados en el método crearPantalla()) por si mismas, dependen de las interfaces. Las interfaces son parte de la dimensión de presentación o borde (ver capítulo 4) y su función es implementar la presentación o vista correspondiente a las bordes del sistema hacia el mundo externo, para todo tipo de actores, no sólo usuarios humanos. Para el caso de una pantalla se creo la clase InterfaceGeneral (Figura 4) de la cual heredan todas las demás interfaces excepto InterfaceBaseDatosCliente e InterfaceBaseDatosAcervo cuya superclase es InterfaceBaseGeneral y se encarga de interactuar con la Base de Datos.

Clase: InterfaceGeneral	
Descripción: superclase principal de todas las clases Interface.	
Módulo: Interfaces	
Estereotipo: Interface	
Propiedades: Abstracta	
Superclase:	
Subclases: InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Atributos: ManejadorPricipal, PantallaGeneral	
Contratos	
1. Desplegar Pantalla	
DesplegarPantalla(Pantalla) devuelvevoid	PantallaGeneral(1): PantallaOfrecerServicio(1), PantallaValidacion(1), PantallaValidaCredencial(1), PantallaConsultaAcervo(1), PantallaResultadoAcervo(1), PantallasAdministracionAcervo(1), PantallaObtenerRegistroAcervo(1), PantallaBuscarRegistroAcervo(1), PantallaDatosAcervo(1), PantallaObtenerRegistroCliente(1), PantallaBuscarRegistroCliente(1), PantallaDatosCliente(1), PantallaObtenerRegistroUsuario(1), PantallaBuscarRegistroUsuario(1), PantallaInscripcion(1), PantallaImprimirCredencial(1), PantallaReporteResultadoUsuarioUA(1), PantallaReporteGuardarUsuarioUA(1), PantallaReporteResultadoAcervoAdq(1), PantallaReporteGuardarAcervoAdq(1), PantallaDatosUsuarioArea(1), PantallaModificarUsuarioArea(1), PantallaReporteInicialUsuario(1), PantallaReporteResultadoUsuarioIC(1), PantallaReporteGuardarUsuarioIC(1), PantallaReporteInicialAcervo(1), PantallaReporteResultadoAcervoAE(1), PantallaReporteGuardarAcervoAE(1), PantallaReporteAdquisiciones(1)
2. Enviar evento	
EnviarEvento(Evento, Manejador) devuelve void	ManejadorPrincipal(1): SubsistemaServicio(1), SubsistemaManejadores(1)

Figura 4.

El código final de la clase InterfaceGeneral se muestra a continuación:

Se indica el paquete a que pertenecera la clase.

```
package tesis;
```

Se importan las bibliotecas necesarias para la construcción de la clase.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import com.borland.jbcl.layout.*;
```

Se inicia el cuerpo de la clase definiéndola de tipo abstracto, es decir, no se pueden crear instancias de esta clase, además se hereda de la clase JFrame (*Contenedor Swing de alto nivel que proporciona ventanas para applets y aplicaciones. Un frame tiene decoraciones como un borde, un título, y botones para cerrar y minimizar la ventana*) para poder desplegar las pantallas y se implementan las interfaces (*en este caso se maneja el término interface para el lenguaje java y son una colección de definiciones de métodos (sin implementaciones) y de valores constantes*) ActionListener y KeyListener. ActionListener escucha los eventos de mouse para enviarlos al manejador correspondiente; KeyListener escucha eventos de teclado y los envia a la pantalla desplegada.

```
public abstract class InterfaceGeneral extends JFrame implements  ActionListener,KeyListener{
```

Variables de instancia y clase.

```
  private ManejadorGeneral manejador;
  private PantallaGeneral pantalla;
  ImageIcon logo = new ImageIcon("Imagenes/UNAM1.gif");
  Image logo1 = logo.getImage();
  private XYLayout xYLayout1 = new XYLayout();
```

El constructor recibe como parámetro la clase ManejadorGeneral. Este manejador es el que solicita el despliegue de una determinada pantalla y es asignado a la respectiva variable de instancia. Además se agrega el oyente para eventos de teclado.

```
  public InterfaceGeneral(ManejadorGeneral m) {
    this.addKeyListener(this);
```

```
    try {
      jbInit();
    }
    catch(Exception e) {
    }
}
```

Este método es la implementación para el contrato 1 con la clase PantallaGeneral. Recibe como argumento la pantalla a desplegar. Verifica que el objeto pantalla no sea nulo. En caso de existir el objeto, la interfaz despliega la pantalla.

```
public void desplegarPantalla(PantallaGeneral p) {
    if (p != null){
      return;
    }
    pantalla = p;
    if (pantalla != null){
      pantalla.desplegarPantalla();
    }
    setVisible(true);
}
```

Método que asigna el manejador que controla la interfaz.

```
public void setManejador(ManejadorGeneral m){
    manejador= m;
}
```

El método actionPerformed corresponde al contrato dos de la InterfaceGeneral, contrato que esta ligado con el Manejador General y subclases de este y cuya funcion es enviar el evento generado en las pantallas al manejador correspondiente para su procesamiento. Esto lo hace a traves de la interface ActionListener.

```
public void actionPerformed(ActionEvent event) {
    if (manejador != null){
      manejador.manejarEvento(event.getActionCommand());
    }
    else
}
```

Metédo con el cual se le da una configuración inicial al frame que contendrá la pantalla a desplegar.

```
private void jbInit() throws Exception {
    this.getContentPane().setBackground(new Color(246, 246, 244));
    this.setIconImage(logo1);
    this.setResizable(false);
    this.setTitle("SGI - ENEO");
    this.getContentPane().setLayout(xYLayout1);
}
```

Los siguientes métodos corresponden a la implementación de la interface KeyListener, aunque sólo se ocupa el método keyReleased para enviarle a la pantalla el juego de teclas ocupado.

```
public void keyTyped(KeyEvent e) {

}

public void keyPressed(KeyEvent e) {

}

public void keyReleased(KeyEvent e) {
```

```
        pantalla.cerradoPantallaTecla(e, "KEY RELEASED: ");
    }
```

Se cierra el cuerpo de la clase.

```
}
```

Las clases InterfaceGeneral, PantallaGeneral y las subclases de ambas conforman la presentaciòn ó borde del sistema pero no controlan la lógica de este. Las clases encargadas de la lógica son el ManejadorGeneral (en los capítulo cuatro y cinco se denominaba ManejadorPrincipal) y las subclases de este. Estas clases responden a las peticiones de los clientes del sistema a partir de los distintos eventos generados por las pantallas y que son enviados a traves de las interfaces; ademas deciden la creación y destrucción de las pantallas e interfaces de acuerdo a la lógica programada en ellos. Son Intermediarios entre la dimensión de presentación y la del dominio (información). La tarjeta del ManejadorGeneral (Figura 5) y el código final de la clase se muestran a continuación.

Clase: ManejadorPrincipal	
Descripción: superclase principal de todas las clases Manejadores.	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Abstracta	
Superclase:	
Subclases: ManejadorPrincipal: ManejadorCliente, ManejadorServicio, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea,	
Atributos: PantallaGeneral, InterfaceGeneral, ManejadorPrincipal, ManejadorServicio	
Contratos	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
Propiedades privadas	
DesplegarPantalla() devuelve void	InterfaceGeneral(1): InterfaceCliente(1), InterfaceUsuario(1), InterfaceAdministrador(1), InterfaceCoordinador(1)
OfrecerServicio() devuelve void	SubsistemaServicios(2)

Figura 5.

Se indica el paquete al que pertenecera la clase.

```
package tesis;
```

Se inicia el cuerpo de la clase definiéndola de tipo abstracto es decir no se pueden crear instancias de esta clase

```
public abstract class ManejadorGeneral  {
```

Variables de instancia.

```
  protected InterfaceGeneral interfac;
  protected PantallaGeneral pantalla;
  protected ManejadorGeneral mg;
  protected ManejadorServicio ms;
```

Contructor de la clase.

```
  public ManejadorGeneral() {

  }
```

Este método corresponde genera el contrato uno de las interfaces del sistema, su función consiste en enviar a las distintas interfaces que pantalla deben desplegar. Recibe dos objetos como parámetros de tipo PantallaGeneral e InterfaceGeneral .

```
  public void desplegarPantalla(PantallaGeneral p, InterfaceGeneral interfa){
```

Los parámetros son asignados a variables de instancia para su uso local.

```
        pantalla = p;
        interfac = interfa;
```

Verificamos que el objeto pantalla no sea nulo, se le indica a la interface que manejador pide el despliegue de pantalla y la pantalla a desplegar.

```
        if (pantalla != null){
           interfac.setManejador(this);
           interfac.desplegarPantalla(p);
        }
  }
```

El último método de esta clase es abstracto por lo tanto todas las clases que hereden de la clase ManejadorGeneral deben implementarlo de acuerdo a las necesidades de la clase heredada. Su función es controlar los eventos enviados por las pantallas y da respuesta al contrato dos hecho con las interfaces.

```
  public abstract void manejarEvento (String str);
```

Se cierra el cuerpo de la clase.

```
}
```

Se explicó la implementación de estas tres clases (PantallaGeneral,InterfaceGeneral y ManejadorGeneral) antes de continuar con la PantallaOfrecerServicio ya que son ellas las que implementan los mecanismo básicos para el funcionamiento del sistema y de ellas se desprenden todas las clases presentación y control. Ahora veamos como interaccionan estas clases en un caso especifico, retomando el caso de uso ofrecer servicios.

Se mencionó que el propósito de este caso de uso es el proporcionar una pantalla en la que se presentará los servicios dependiendo del tipo de cliente (Usuario, Administrador, Coordinador). Esta pantalla esta implementada por la clase PantallaOfrecerServicio, pero por si sola no puede desplegar la pantalla necesita de un objeto de tipo interface y un manejador que controle la lógica del caso de uso. En los capítulos anteriores se identificó la clase ManejadorServicio (Figura 6) para controlar la funcionalidad del caso de uso ofrecer sevicios.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encargara de controlar las distintas pantallas para los diferentes tipos de clientes	
Módulo: ClaseControl	
Estereotipo: Control	
Propiedades: Concreta	
Superclase:	
Subclases:	
Atributos: ManejadorCliente, ManejadorConsultaAcervo, ManejadorAcervo, ManejadorUsuario, ManejadorReporteUsuario, ManejadorReporteAcervo, ManejadorUsuarioArea, PantallaValidaCredencial, PantallaOfrecerServicio, InterfaceCliente, InterfaceUsuario, InterfaceAdministrador, InterfaceCoordinador	
Contrato	
1. Manejar Evento	
ManejarEvento(Evento) devuelve void	
2. Ofrecer Servicio	
OfrecerServici() devuelve void	
Responsabilidades privadas	
ConsultarAcervo() devuelve void	ManejadorConsultaAcervo(2)
ModificarAcervo() devuelve void	ManejadorAcervo(2)
ModificarCliente() devuelve void	ManejadorCliente(2)
ModificarUsuario() devuelve void	ManejadorUsuario(2)
Inscripción() devuelve void	ManejadorUsuario(2)
Validación() devuelve void	ManejadorCliente(2)
ValidaciónUsuario() devuelve void	ManejadorCliente(2)
ValidaciónAdministrador() devuelve void	ManejadorCliente(2)
ValidaciónCoordinador() devuelve void	ManejadorCliente(2)
AdministrarUsuarioPorArea() devuelve void	ManejadorUsuarioArea(2)
GenerarReporteUsuario() devuelve void	ManejadorReporteUsuario(2)
GenerarReporteAcervo() devuelve void	ManejadorReporteAcervo(2)

Figura 6

Se presentará sólo parte del código de la clase ManejadorServicio debido a su extenso tamaño.

Código de la clase ManejadorServicio.

```
package tesis;
```

La clase ManejadorServicio hereda de la clase ManejadorGeneral.

```
public class ManejadorServicio extends ManejadorGeneral {
```

Variables de intancia y clase.

En esta variable de clase de tipo interfeceCliente se despliega la pantalla ofrecer servicios. Se definio de tipo estática porque la pantalla estará presente mientras este ejecutándose el sistema.

```
public static InterfaceCliente InterfaceServicio
```

También se encuentra dentro de las variables de instancia (que corresponden a los atributos en la tarjeta de clase del ManejadorServicio) una variable tipo PantallaOfrecerServicio

```
protected InterfaceCliente InterfaceCliente;
protected PantallaOfrecerServicio PantallaServicio;
protected ManejadorCliente mc;
protected ManejadorUsuario mu;
protected ManejadorAcervo ma;
protected ManejadorUsuarioArea mua;
protected ManejadorConsultaAcervo mcau;
protected ManejadorReporteUsuario mru;
protected ManejadorReporteAcervo mra;
protected ManejadorBusquedaGeneralUsuario mbgu;
protected ManejadorBusquedaGeneralAcervo mbga;
private static int contador =1;
```

Constructor de la clase.

```
public ManejadorServicio() {
```

En el constructor se crea un objeto estático de tipo InterfaceCliente (InterfaceCliente hereda de la clase InterfaceGeneral)

```
        InterfaceServicio = new InterfaceCliente(this);
```

Se verifica si la pantalla que se tiene que desplegar es la pantalla ofrecer servicio, si es así se llama al método *ofrecerServico* de esta clase pasándole como parámetro una cadena ("Sin Servicio")

```
        if(PantallaGeneral.banderaservicio == false && contador == 1){
          ofrecerServicio("Sin Servicio");
          contador++;
        }
    }
```

A través del método *ofrecerServicio* definimos la configuración de la pantalla ofrecer servicios dependiendo del tipo de cliente que sea, se desplegarán los menús a los que puede acceder.

```
public  void ofrecerServicio(String persona){
        if (persona.equals("Sin Servicio")){
          desplegarPantallaOfrecerServicio(1);
        }

        if (persona.equals("Coordinador")){
          desplegarPantallaOfrecerServicio(2);
        }
```

```
        if (persona.equals("Administrador")){
           desplegarPantallaOfrecerServicio(3);
        }

        if (persona.equals("Usuario")){
           desplegarPantallaOfrecerServicio(4);
        }

    }
```

Observando nuevamente la tarjeta de caso de uso (figura 1) se comprenderá mejor este método, la cadena que se le envio al método *ofrecerServicio* indica que estamos en el subflujo 1 (Figura 7).

Caso de uso:	**Ofrecer Servicios**
Actores:	Cliente, Usuario, Administrador, Coordinador
Tipo:	Inclusión
Propósito:	Proporciona una pantalla en la que se presentará los servicios dependiendo del tipo de cliente (Usuario, Administrador, Coordinador)
Resumen:	Este caso es iniciado por un cliente. Muestra las opciones para utilizar el sistema SGI – ENEO.
Precondiciones:	Debe de haberse validado el cliente en todos los subflujos excepto el S – 1.
Flujo principal:	Se ejecuta el caso de uso Ofrecer Servicios y dependiendo del tipo de cliente, se continuara con los diversos subflujos de este caso de uso.
Subflujos:	S – *1 Sin Servicio.* Se presenta al cliente la pantalla P – 2. El cliente puede seleccionar entre las siguientes opciones de un menú desplegable llamado Inicio, dejando inhabilitados el resto de los menús. El cliente puede seleccionar de este menú las siguientes opciones: "Validación" y "Salir". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. Si la actividad seleccionada es "Salir" sale del sistema.

Figura 7.

El Método desplegarPantallaOfrecerServicio crea un objeto de la clase PantallaOfrecerServicio, y es utilizado como argumento junto con el objeto InterfaceServicio para el método desplegarPantalla definido en la clase ManejadorGeneral a través del cual se le dice a la interface que pantalla debe desplegar.

```
public  void desplegarPantallaOfrecerServicio(int actor){
            PantallaServicio = new PantallaOfrecerServicio(InterfaceServicio, this, actor);
            desplegarPantalla(PantallaServicio, InterfaceServicio);
    }
```

Se puede ver mejor a través de un diagrama de secuencia.

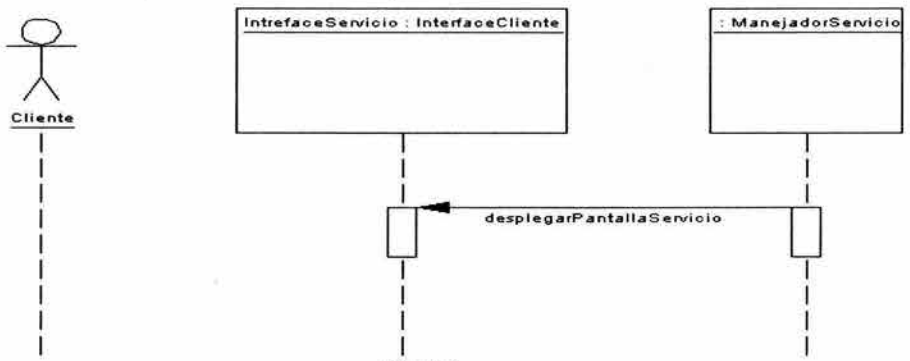


Figura 8.

Y la pantalla tendría la configuración descrita en el subflujo uno del caso de uso

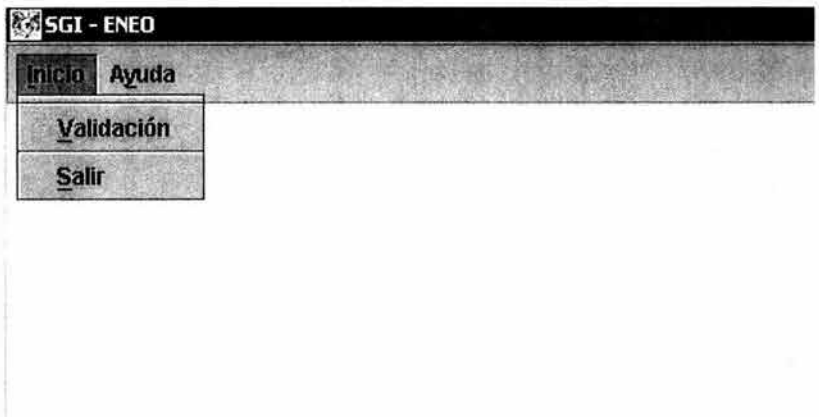


Figura 9.

La única pantalla disponible es la PantallaOfrecerServicio que está en estatus "Sin Servicio", es decir, no se ha validado ningún tipo de cliente (Usuario, Administrador, Coordinador).

Hasta el momento el sistema no ha interactuado con algún actor. Como se observa en el diagrama de secuencia (Figura 8). El momento en que un actor empieza a interactuar con el sistema es cuando envía eventos a través de las pantallas (Figura 11). Se tomará el caso de uso Validar Cliente (analizado en los capítulos anteriores) para dar un ejemplo de la interaccion entre el sistema y los clientes.

Revisando el caso de uso Validar Cliente (Figura 10) su función es Validar a un Cliente (Usuario, Administrador, Coordinador) ya registrado para el uso del sistema.

Caso de Uso:	**Validar Cliente**
Actores:	Usuario, Administrador, Coordinador, Base de Datos
Tipo:	Inclusión
Propósito:	Validar a un Cliente (Usuario, Administrador, Coordinador) ya registrado para el uso del SGI-ENEO
Resumen:	Este caso es iniciado por un Cliente, validándolo mediante un *login* y *password*, así como, con la selección de una de las tres opciones presentadas (Coordinador, Administrador, Usuario) comparando estos datos con los almacenados en los registros de la Tabla Registro Cliente de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente los Casos de Uso: Inscripción (Usuario), Administrar Cliente (Administrador, Coordinador)
Flujo principal:	Se presenta al Cliente la pantalla P–1. El Cliente puede seleccionar entre las siguientes opciones: "Aceptar" y "Cancelar". Si la actividad seleccionada es "Aceptar" se valida el registro del Cliente mediante los datos proporcionados (una de las tres opciones y su *login* y *password*). Una vez validado el Cliente (Ex–1), se continua con el caso Ofrecer Servicios y dependiendo del tipo de Cliente, el subflujo será S–2 (Usuario), S–3 (Administrador), S–4(Coordinador). Si la actividad seleccionada es "Cancelar" se continua con el caso *Ofrecer Servicios* subflujo S–1.
Subflujos:	Ninguno
Excepciones:	EX –1 no hubo validación, la contraseña (*login* y *password*) no corresponde, se cancelan los datos proporcionados y se manda un mensaje de error, y en seguida se solicita al Cliente validarse de nuevo.

Figura 10.

Para validar un cliente se selecciona el item Validación(Figura 9), esta operación genera un evento el cual es capturado por la InterfaceCliente y lo envia al manejador correspondiente que en este caso es el ManejadorServicio (ver diagrama de secuencia, Figura 11).

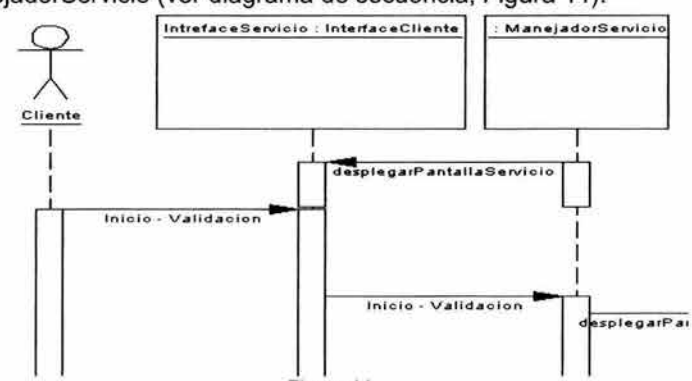


Figura 11.

El método que procesa el evento es manejarEvento() heredado por la clase ManejadorGeneral pero que es abstracto por lo tanto se tiene que implementar. El código final para el método es el siguiente:

```
public void manejarEvento(String Evento) {
        if (Evento.equals("Salir")){

          cerrarSistema();
        }

        else if (Evento.equals("Validación")){    ------- Aqui se verifica que se trata del evento validación
          validacion();
        }
        .
        .
        .
        .
}
```

Se observa que el método recibe como argumento un valor tipo cadena (String), el cual contiene el nombre del evento generado, en nuestro caso es el evento "Validación". Se llama al método Validación(),el cual crea una intancia de un objeto de la clase ManejadorCliente; manejador encargado de la lógica del caso de uso Validar cliente.

```
private void validacion(){
        mc = new ManejadorCliente();
        mc.desplegarPantallaValidacion();
  }
```

El manejador llama al método desplegarPantallaValidacion el cual crea un objeto de la clase InterfaceCliente, deshabilita la PantallaOfrecerServicios, crea un objeto de la clase PantallaValidación, le agrega la información que necesite y pide su despliegue a la interfaceCliente a través del método desplegarPantalla heredado de la Clase ManejadorGeneral. En el diagrama de secuencia (Figura 12) se comprende mejor la lógica.

```
public void desplegarPantallaValidacion(){
        InterfaceCliente = new InterfaceCliente(this);
        ManejadorServicio.InterfaceServicio.setEnabled(false);
        pantallaValidacion = new PantallaValidacion(InterfaceCliente, this);
        interfaceBaseDatosCliente.obtenerDatosPantallas(pantallaValidacion);
        desplegarPantalla(pantallaValidacion, InterfaceCliente);
  }
```

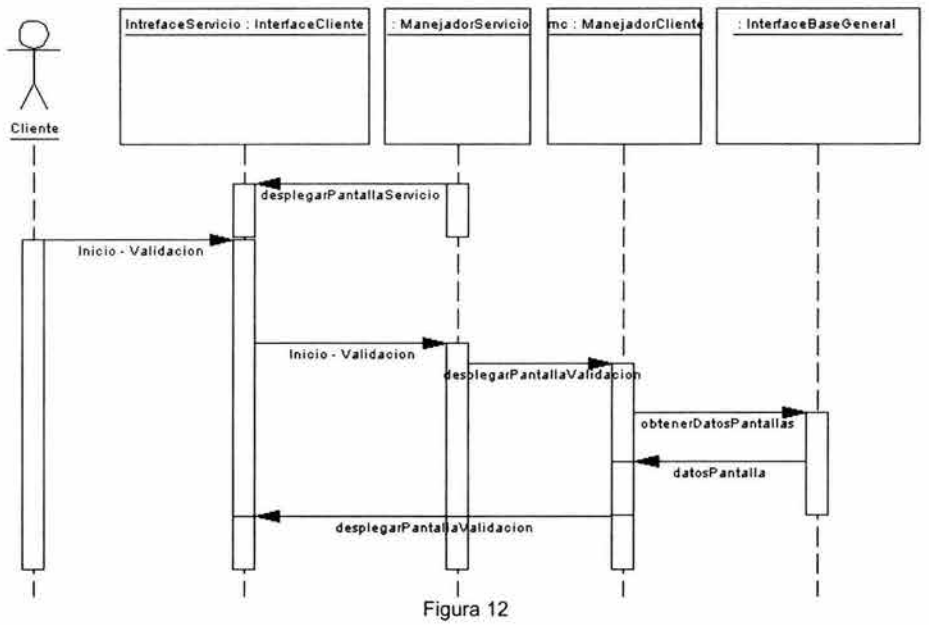


Figura 12

Y la pantalla quedaria asi:

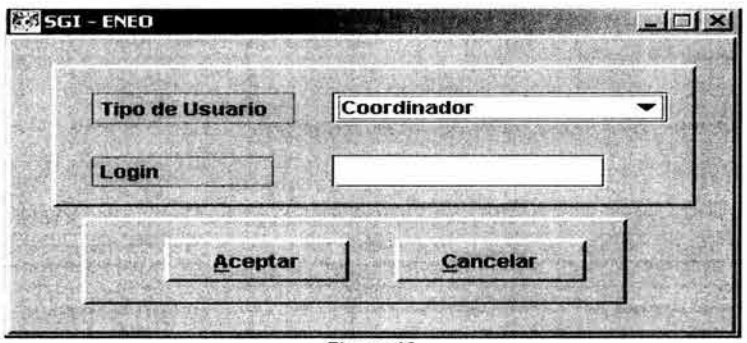


Figura 13

La Clase PantallaValidación hereda de la clase PantallaGeneral, esta pantalla ofrece a los distintos clientes el poder validarse como Usuario, Administrador ó Coordinador. Se presenta solo el código representantivo de la clase.

El contructor de la clase, utiliza el contructor de su superclase (PantallaGeneral), define las dimensiones de la pantalla a través del método tamañoVentana().

```
public PantallaValidacion(InterfaceGeneral ig, ManejadorCliente mg) {
        super(ig,mg);
        tamañoVentana(390,235);
        mc = mg;
  }
```

A través de este método se le pasa una consulta (en SQL) a la interfaceBaseDatosCliente para indicarle que información necesita para desplegar su información necesaria.

```
public  String obtenerAtributosPantalla(){
        consulta = "Select Tipo_cliente from TipoCliente";
        return consulta;
  }
```

La InterfaceBaseDatosCliente devuelve la información ó las herramientas para la construcción de la pantalla.

```
public  void asignarAtributosPantalla(ResultSet rs,Statement stmt){
        int sum =0;
        int tamaño;
        try{
          rs.last();
          tamaño=rs.getRow();
          String [] usuarios = new String[tamaño];
          rs.beforeFirst();
          while(rs.next()){
                usuarios[sum] = rs.getString(1);
                ++ sum;
          }
          listaUsuarios = new JComboBox(usuarios);
        }
        catch (SQLException ex){
        }
}
```

El cliente necesita escribir su password y seleccionar su perfil (Usuario, Administrador y Coordinador) en la PantallaValidacion (Figura 13) y dar clic en el botón Aceptar. La PantallaValidación generará un evento que la InterfaceCliente pasará al ManejadorCliente.

Se presentará sólo parte del código de la clase ManejadorCliente debido a su extenso tamaño.

El método constructor de la clase ManejadorCliente a diferencia de ManejadorServicio crea un objeto de la Clase InterfaceBaseDatosCliente, clase cuya función primordial es recibir y enviar información de la Base de Datos.

```
public ManejadorCliente() {
        interfaceBaseDatosCliente = new InterfaceBaseDatosCliente();
  }
```

Al igual que la clase ManejadorServicio, ManejadorCliente hereda de la clase ManejadorGeneral el método abstracto ManejarEvento() y tiene que implementarlo dándole la funcionalidad para procesar los eventos generados por las pantallas que dependen del manejador.

```
public void manejarEvento(String str) {
        if (str.equals("Salir validacion")){
          if(interfaceBaseDatosCliente != null){
                interfaceBaseDatosCliente.cerrarConexion();
          }
          ManejadorServicio.interfaceServicio.setEnabled(true);
          InterfaceCliente.dispose();
        }

        if (str.equals("Aceptar")){                    ------- Aquí se verifica que se trata del evento Aceptar
          validarRegistroCliente();
          if(interfaceBaseDatosCliente == null){
                interfaceBaseDatosCliente.cerrarConexion();
        }
        .
        .
        .
        .
        }
```

Si el evento es "Aceptar", se ejecuta el método validarRegistroCliente(), del cual se tratará a detalle su implementación.

```
private void validarRegistroCliente(){

        String login = pantallaValidacion.getLogin();
```

```
        String usuario = pantallaValidacion.getListaUsuario();

        if(interfaceBaseDatosCliente.validarRegistroCliente(usuario, login) == true){ -----Validación de datos.
          InterfaceCliente.dispose();
          PantallaGeneral.banderaservicio = false;
          if(interfaceBaseDatosCliente != null){                    ----------Cierre de comunicaciones
                interfaceBaseDatosCliente.cerrarConexion();                    la Base de Datos.
          }

          if (ManejadorServicio.InterfaceServicio != null){         ----------Cierre de PantallaServicio
                ManejadorServicio.InterfaceServicio.hide();                    anterior.
          }
          ManejadorServicio mc = new ManejadorServicio();           ----------Creación de Manejador
          mc.ofrecerServicio(usuario);                                         Servicio.
        }
        else{                                                       ----------Datos no válidos.
          JOptionPane mensaje = new JOptionPane();
          mensaje.showMessageDialog(InterfaceCliente,
                                    "Tus datos de tipo de usuario y login son incorrectos, "+
         "Por favor verificalos",
"Mensaje de error", 1);
          pantallaValidacion.setLogin("");
        }
}
```

Se obtiene el tipo de Cliente y el password con el cual el cliente desea validarse, mediante los métodos getLogin() y getListaUsuario() de la pantalla validación y los datos son almacenados en las variables locales tipo String login y usuario, Estas variables se pasan como parámetros al método validarRegistroCliente() de la clase InterfaceBaseDatosCliente. El código de este método se visualizó renglones atrás. El método devuelve un valor tipo boolean que es verdadero cuando los datos proporcionados por el Cliente se encuentran en la Base de Datos y falso en caso contrario.

En base a estos valores se envía un mensaje de error (Figura 14) en caso de ser falso y en caso contrario se despliega la PantallaOfrecerServicio dependiendo del tipo de Cliente (ver caso de uso Ofrecer Servicio Figura1).

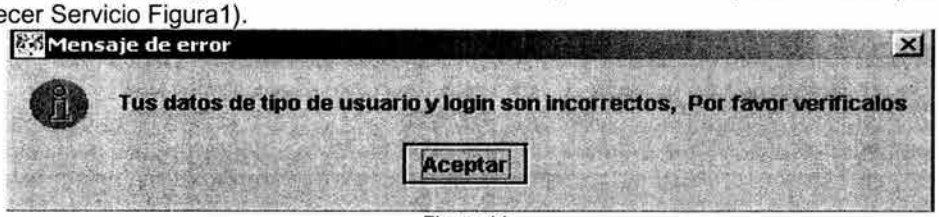


Figura 14.

Y el diagrama de secuencia para la operación anterior desde el caso de uso ofrecerServicio hasta la validación del Cliente es:

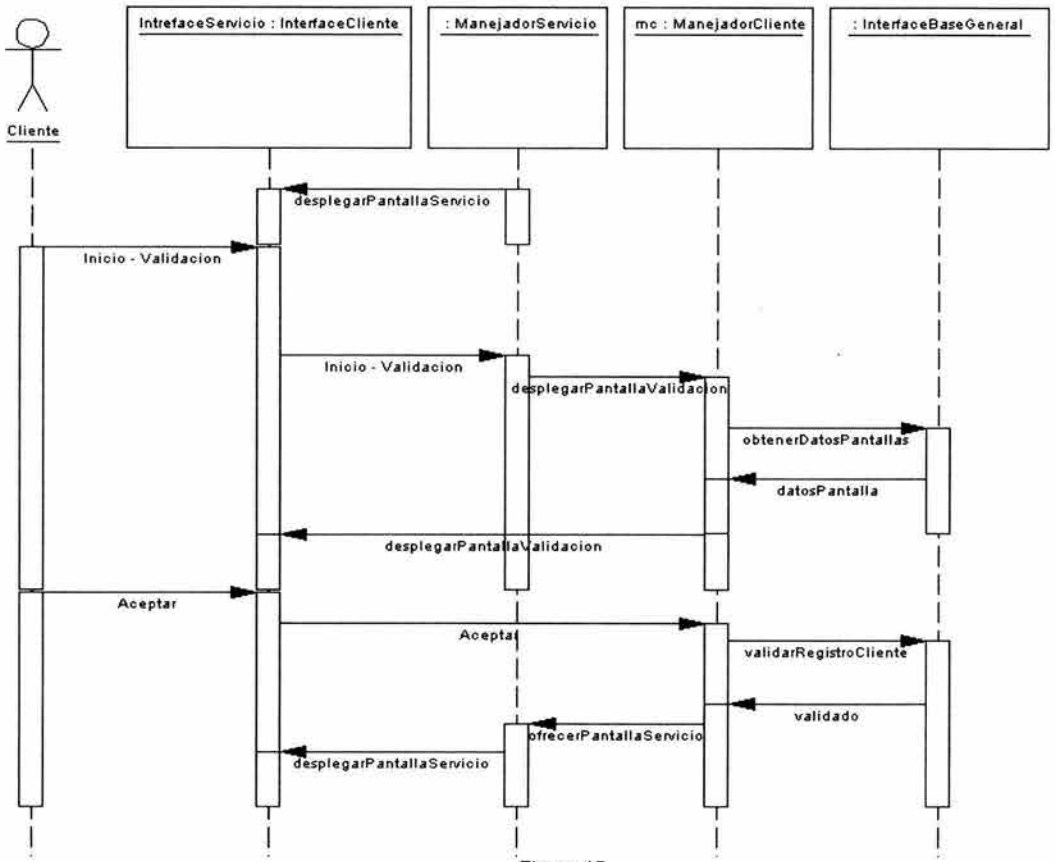


Figura 15.

Se ha visto el funcionamiento de la dimensión borde interactuando con actores humanos, pero el sistema también se comunica con una Base de Datos que es considerada como un actor secundario (Figura 16) y para poder establecer comunicación entre el sistema y la base se creó la clase InterfaceBaseGeneral y sus subclases InterfaceBaseDatosCliente (para los casos de uso relacionados con Usuarios, Administradores y Coordinadores) e InterfaceBaseDatosAcervo (para los casos de uso relacionados con el Acervo de la UAII).

Actor:	**Base de Datos**
Casos de Uso:	Validar Cliente, Validar Usuario, Inscripción, Administrar Usuario, Imprimir Credencial, Generar Reportes Usuario, Administrar Cliente, Administrar Usuarios por Área, Consultar Acervo, Administrar Acervo, Generar Reportes Acervo.
Tipo:	Secundario
Descripción:	Es un actor secundario y representa las Base de Datos donde se guarda toda la información relacionada con los Clientes (Usuario, Administrador y Coordinador) y el Acervo.

Figura 16. Actor Base de Datos.

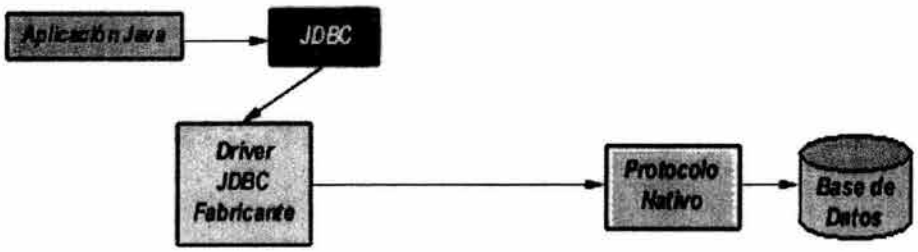
La clase interfaceBaseGeneral no fue detectada durante los capítulos de Análisis y Diseño, fue diseñada y creada durante la implementación de este sistema, surgió debido a que las clases InterfaceBaseDatosCliente e InterfaceBaseDatosAcervo en su diseño final no se definieron métodos para realizar la conexión entre el sistema y la Base de Datos.

La conexión a la Base de Datos (cuyo DBMS es Sybase) será a través del API JDBC.

JDBC (*Java DataBase Connectivity*) es un API de Java que permite al programador ejecutar instrucciones en lenguaje estándar de acceso a Bases de Datos, **SQL** (*Structured Query Language*, lenguaje estructurado de consultas), que es un lenguaje de muy alto nivel que permite crear, examinar, manipular y gestionar Bases de Datos relacionales.

Para que una aplicación pueda hacer operaciones en una Base de Datos, ha de tener una conexión con ella, que se establece a través de un *driver*, que convierte el lenguaje de alto nivel a sentencias de Base de Datos. Es decir, las tres acciones principales que realizará JDBC son las de establecer la conexión a una Base de Datos, ya sea remota o no; enviar sentencias SQL a esa Base de Datos y, en tercer lugar, procesar los resultados obtenidos de la Base de Datos.

Un driver suele ser un archivo .jar (dentro del cual se almacenan clases) que contienen una implementación de todos los interfaces del API de JDBC. Se usará un driver realizado completamente en Java (proporcionado por Sybase gratuitamente) que se comunica con el servidor DBMS utilizando el protocolo de red nativo del servidor. De esta forma, el driver no necesita intermediarios para hablar con el servidor y convierte todas las peticiones JDBC en peticiones de red contra el servidor. La ventaja de este tipo de driver es que es una solución *100% Java* y, por lo tanto, independiente de la máquina en la que se va a ejecutar el programa.



La clase que se encarga de cargar inicialmente todos los drivers JDBC disponibles es **DriverManager**. Una aplicación puede utilizar **DriverManager** para obtener un objeto de tipo conexión, **Connection**, con una Base de Datos. La conexión se especifica siguiendo una sintaxis basada en la especificación más amplia de los URL, de la forma

jdbc:subprotocolo//servidor:puerto/Base de Datos

Una vez que se tiene un objeto de tipo **Connection**, se pueden crear sentencias, **statements**, ejecutables. Cada una de estas sentencias puede devolver cero o más resultados, que se devuelven como objetos de tipo **ResultSet**.

Y la tabla siguiente muestra la misma lista de clases e interfaces a utilizar junto con una breve descripción.

Clase/Interface	***Descripción***
Driver	Permite conectarse a una Base de Datos: cada gestor de Base de Datos requiere un driver distinto
DriverManager	Permite gestionar todos los drivers instalados en el sistema
Connection	Representa una conexión con una Base de Datos. Una aplicación puede tener más de una conexión a más de una Base de Datos
DatabaseMetadata	Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc.
Statement	Permite ejecutar sentencias SQL sin parámetros
PreparedStatement	Permite ejecutar sentencias SQL con parámetros de entrada
ResultSet	Contiene las filas o registros obtenidos al ejecutar un SELECT
ResultSetMetadata	Permite obtener información sobre un **ResultSet**, como el número de columnas, sus nombres, etc.

Regresando al código de la clase

Se indica el paquete al que pertenece la clase.

```
package tesis;
```

Se importan las bibliotecas necesarias para la construcción de la clase.

```
import java.sql.*;      ------------------------ se importa toda la funcionalidad de JDBC
import java.util.*;
import entidad.*;
import java.text.*;
import javax.swing.*;
```

Se inicia el cuerpo de la clase.

```
public class InterfaceBaseGeneral {
```

Se crea un objeto de la clase ControladorTesis, por medio del cual obtendremos del archivo "ArchivoC.SGI" los datos para realizar la conexión con la base. Estos son:

- ClassforName: carga el nombre del driver (com.sybase.jdbc2.jdbc.SybDriver)
- url: jdbc:sybase:Tds(direccion IP):2638/Tesis.db
- login:dba
- password:sql

```
  private ControladorTesis controladorTesis = new ControladorTesis();
  protected Connection con;
  protected Statement stmt;
  protected ResultSet rs;
  protected String classForName;
  protected String url;
  protected String login;
  protected String password;
```

Inicia el constructor de clase. Se obtienen los datos necesarios para realizar la conexión y se manda llamar al método revisarDriver(); este método cargará el Driver, si este se carga satisfactoriamente se abrirá una conexión a la Base de Datos y se crea un Objeto Statement por medio del método abrirConexion().

```
  public InterfaceBaseGeneral() {
        classForName = controladorTesis.configuracion("ClassforName");
        url = controladorTesis.configuracion("url");
        login = controladorTesis.configuracion("login");
        password = controladorTesis.configuracion("password");
        int existenciaDriver = revisarDriver();
        if( existenciaDriver == 0)
   abrirConexion(url, login, password);
  }
```

Método que carga el Driver para realizar la conexión con la Base de Datos.

```
  private int revisarDriver()
  {
        int fg = 0;
        try {
          Class.forName (classForName);
        }
        catch (ClassNotFoundException ex) {
          fg = 1;
        }
        return fg;
  }
```

El método abrirConexion() utiliza los parámetros obtenidos por el objeto de la clase ControladorTesis (URL, Login, Password), verifica los posibles errores de conexión y crea un objeto de la clase Statement (Permite ejecutar sentencias SQL).

Los parámetros pasados al metodo createStatement (perterneciente a los objetos de la clase Connection) son:

- ResultSet.TYPE_SCROLL_SENSITIVE el cual permite movimiento bidireccional entre registros y que los cambios efectuados en la Base de Datos se vean reflejeados en el ResulSet.
- ResultSet.CONCUR_UPDATABLE permite que la Base de Datos sea actualizable mediante el objeto ResultSet.

```
private void abrirConexion(String url,String log,String pass)
{
        try {
           con = DriverManager.getConnection (url, log, pass);
           checkForWarning (con.getWarnings ());
           stmt = con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
        }
        catch (SQLException ex) {
           while (ex != null) {
           }
        }
        catch (Exception ex) {
        }
}
```

Este método verifica los posibles errores de conexión.

```
private static boolean checkForWarning (SQLWarning warn)
        throws SQLException  {
        boolean rc = false;
        if (warn != null) {
           rc = true;
           while (warn != null) {
                warn = warn.getNextWarning ();
           }
        }
        return rc;
}
```

El método obtenerDatosPantallas fue creado debido a la necesidad que tenian los objetos de tipo pantalla para mostrar información en sus componentes (Combo Box, Text field, etc) verifica los posibles errores de conexión.

```
public void obtenerDatosPantallas(PantallaGeneral p){
        try{
           String consulta = p.obtenerAtributosPantalla();
           rs = stmt.executeQuery(consulta);
           p.asignarAtributosPantalla(rs, stmt);
        }
        catch (SQLException ex){
        }
}
```

Mediante este método se cierra la conexión a la Base de Datos.

```
public void cerrarConexion(){
        try{
           con.close();
        }
        catch (SQLException ex){
        }
}

}
```

La clase InterfaceBaseGeneral hereda a las clases InterfaceBaseDatosCliente e InterfaceBaseDatosAcervo. Objetos de la clase InterfaceBaseDatosCliente fueron instanciados durante el caso de uso ValidarCliente y se emplearon los métodos:

- obtenerDatosPantalla() el cual esta definido en la clase InterfaceBaseGeneral
- validarRegistroCliente() cuya función es validar el tipo de Cliente (Usuario, Administrador ó Coordinador).

```
public boolean validarRegistroCliente(String usuario, String login){
        REGISTROCLIENTE rc =new REGISTROCLIENTE();
        try{
```

Se crea una variable tipo String en la cual se almacena la consulta quese envía a la Base de Datos para determinar si el Registro del Cliente existe con el tipo y password proporcionados.

```
        String busquedaSQL = "SELECT * FROM "+ obtenerNombreClase(rc) +" RC, TIPOCLIENTE TC WHERE
        RC.PASSWORD = '" + login + "' AND TC.tipo_cliente = '" + usuario + "' and rc.id_tipo_cliente =
        tc.id_tipo_cliente";
```

Se ejecuta la consulta a través del objeto stmt (instancia de la clase Statement) almacenando el resultado en el objeto rs (instancia de la clase ResultSet) Si la consulta obtuvo un registro, regresa un valor booleano igual a true, en caso contrario regresa un false.

```
          rs = stmt.executeQuery(busquedaSQL);
          if(rs.next()){
                return true;
          }
          else{
                return false;
          }
        }
        catch (SQLException ex){
          return false;
        }
}
```

6.2 BASE DE DATOS

6.2.1 Definición de Bases de Datos

Desde que aparecen las computadoras, el problema que se presenta es cómo almacenar información y cómo se extrae la información para resolver problemas.

En un inicio los datos se organizan en registros de la misma estructura y un conjunto de registros formal lo que se denomina Archivo. Inicialmente, el registro se organiza en forma secuencial es decir, en orden de aparición. Sin embargo, esta forma presenta varias limitaciones, para encontrar información de un registro era necesario recorrer todo el archivo hasta encontrarlo. Por esto comienzan a aparecer nuevas formas de organización de la memoria, obteniendo el concepto de Base de Datos.

Una Base de Datos se define como:

Un conjunto estructurado de datos, almacenados en soportes, periféricos, accesibles por la computadora, para múltiples usuarios, al mismo tiempo y de manera eficiente.

6.2.1.1 Sistema de Gestión de Bases de Datos (DBMS)

Es un conjunto de software cuya finalidad es lograr que una Base de Datos funcione correctamente, de acuerdo a los siguientes objetivos:

* *DESCRIPCIÓN*: Es necesario contar con una lista de todos los elementos que componen la Base de Datos. Esto se puede implementar mediante la herramienta conocida como Lenguaje de Definición de Datos (LDD). Esta definición, impide que la información se repita en la Base de Datos.

La Definición puede ser:

- *Lógica:* en esta forma se especifica la manera en que el usuario ve los datos.
- *Física:* en esta forma se especifica la manera en cómo se almacena la información.

* *UTILIZACIÓN:* En general se dice que una Base de Datos tiene dos tipos de Usuarios, especificándose para cada grupo un tipo de 'lenguaje'.

Tipos de Usuarios:

- No especializados: los usuarios que utilizan la Base de Datos mediante comandos.
- Especializados: emplean lenguajes de manipulación de datos o lenguajes de programación.

**INTEGRIDAD:* Se refiere a la existencia de reglas que asegura a coherencia de la Base de Datos, tales reglas se denominan *Reglas de Integridad.*

* *CONFIDENCIALIDAD* Es necesario impedir que personas No autorizadas accedan a la Base de Datos por medio de claves.

* *CONCURRENCIA:* El hecho de que dos ó más personas pueden solicitar el mismo dato al mismo tiempo; el sistema debe ser capaz de resolver sin error lo deseado simultáneamente.

**SEGURIDAD DE FUNCIONAMIENTO:* Se refiere al control contra ciertas fallas que pueden ocurrir: Accidentales ó Provocadas. Por lo cual es sistema debe tener un seguro para mantener la Base de Datos en un estado seguro y coherente.

**NO REDUNDANCIA DE DATOS:* Se desea que un dato este únicamente una vez en la Base de Datos.

6.2.1.2 Diferentes niveles de representación de BD

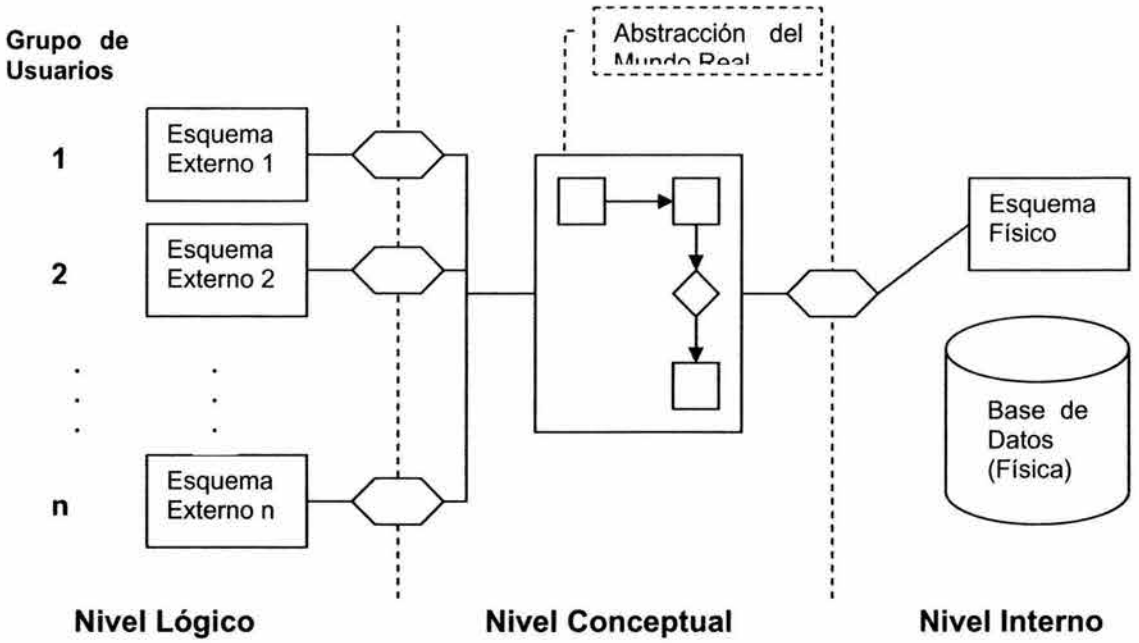
En la representación de las Bases de Datos, existen diferentes formas de representación.

* *Externo (Lógico)*: Se refiere a la manera en que los Usuarios ven a la Base de Datos.

* *Conceptual:* se refiere a la vista global de la Base de Datos. Se refiere al proceso de modelización que siguen los datos desde que están en el mundo real hasta que están disponibles a todos los grupos de Usuarios.

* *Interno (Físico):* Se refiere a la forma en que están almacenados los datos en el interior de la computadora (los bits de almacenamiento).

En la siguiente figura se muestra el detalle de los niveles de representación de una Base de Datos:



Interface

Dentro del modelado de las Bases de Datos, refriedose al Nivel Conceptual de representación, se tiene que el hay tres modelos fundamentales para desarrollar una Base de Datos:

* *JERÁRQUICO:* Representa las asociaciones entre los datos usando arboresencia.
* *REDES:* La noción de arboresencia se extiende a *n* posibilidades.
* *RELACIONAL:* Representa los datos utilizando el concepto de relación matemática que se traduce como tablas.

6.2.2 PRINCIPALES MODELOS DE DISEÑO DE BASES DE DATOS

6.2.2.1 MODELO ESQUEMA CONCEPTUAL

Algunas definiciones para este modelo son:

ENTIDAD: Objeto que se distingue de los demás de una manera única. (▭)

ASOCIACIÓN (entre Entidades): Enumera las relaciones entre las entidades, del mundo que se va a modelar. (◊)

TIPOS DE ASOCIACIÓN:

* ***1:1*** Cuando los objetos de distintas clases están relacionados uno a uno.
* ***1:N*** Cuando a un objeto de una clase le corresponden *N* objetos de otra clase.
* ***N:M*** Cuando a *N* objetos de una clase le corresponden *M* objetos de otra clase.

6.2.2.2 MODELO ENTIDAD RELACIÓN

De manera sistemática, en este modelo se enumeran los atributos del modelo:

- Conjunto de Entidades
- Conjunto de Asociaciones

Para la implementación de este modelo, se recomiendan los siguientes pasos:

* Enumerar los tipos de Entidades

* Enumerar Tipos de Asociaciones
* Dibujar el modelo Entidad-Relación paras las entidades y asociaciones enumeradas.
* Identificar el tipo de Variables y Atributos.
* Traducir el diagrama Entidad-Relación al diseño lógico siguiendo alguno de los modelos posibles: jerárquico, de redes ó relacional.
* Diseñar los formatos de Registro.

6.2.3 BASE DE DATOS DEL SGI-ENEO

La Base de Datos del SGI-ENEO, se diseñó mediante la herramienta Power Designer 9.1, esta herramienta, permite elaborar asociaciones, relaciones e identificación de entidades involucradas en el Sistema.

Así, durante el desarrollo del modelo Entidad-Relación dela Base de Datos del SGI-ENEO se obtuvieron la siguiente información:

TABLAS DE LA BASE

NOMBRE	*CÓDIGO*	*DESCRIPCIÓN*
RegistroCliente	REGISTROCLIENTE	Tabla de almacenamiento para datos de los distintos Clientes del sistema.
TipoCliente	TIPOCLIENTE	Tabla para almacenar a los tipos de Cliente en el SGI-ENEO.
RegistroUsuario	REGISTROUSUARIO	Tabla para almacenar los datos de los Usuarios de la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.
Categorías	CATEGORIAS	Tabla para almacenar las distintas categorías de Usuario.
Idiomas	IDIOMAS	Catálogo de Idiomas.
Carreras	CARRERAS	Catálogo de Carreras de los alumnos de la ENEO, usuarios del SGI-ENEO.
RegistroAcervo	REGISTROACERVO	Tabla para almacenar datos del acervo de la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.
Acentos	ACENTOS	Catálogo de acentos del acervo.
CatDificultad	CATDIFICULTAD	Catálogo de dificultad del acervo.
Estados	ESTADOS	Catálogo del estado en que se encuentra el acervo.
Soportes	SOPORTES	Catálogo de tipo de soporte del acervo.
Secciones	SECCIONES	Catálogo de secciones del acervo.
Subsecciones	SUBSECCIONES	Catálogo de subsecciones del acervo.
DivisionTematica	DIVISIONTEMATICA	Catálogo de las divisiones temáticas del acervo.
SubdivisionTematica	SUBDIVISIONTEMATICA	Catálogo de las subdivisiones temáticas del acervo.
UsuariosPorAreas	USUARIOSPORAREAS	Tabla en la que se almacenarán los de las consultas de Usuarios por áreas.
Ubicaciones	UBICACIONES	Catálogo de ubicaciones del acervo.
Catalogadores	CATALOGADORES	Catálogo de Clientes del SGI-ENEO, clasificados como Catalogadores del acervo.

ENTIDADES:

NOMBRE	*CÓDIGO*	*DESCRIPCIÓN*	*TABLA*
Id_cliente	ID_CLIENTE	Número consecutivo de los clientes con acceso al SGI - ENEO	RegistroCliente
nombre	NOMBRE	Nombre + Apellido paterno + Apellido materno del cliente.	RegistroCliente
cargo	CARGO	Cargo que ocupa el cliente dentro de la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.	RegistroCliente
Id_tipo_cliente	ID_TIPO_CLIENTE	Identificador del tipo de Cliente (Usuario, Administrador o Coordinador)	RegistroCliente
telefono	TELEFONO	Número de teléfono particular del Cliente del SGI-ENEO.	RegistroCliente
password	PASSWORD	Palabra clave para validar el tipo de	RegistroCliente

		Cliente del SGI-ENEO.	
notas	NOTAS	Observaciones relacionadas con el Cliente del SGI-ENEO.	RegistroCliente
id_tipo_cliente	ID_TIPO_CLIENTE	Identificador del tipo de Cliente (Usuario, Administrador y Coordinador)	TipoCliente
tipo_cliente	TIPO_CLIENTE	Describe el tipo de Cliente (Coordinador, Administrador o Usuario).	TipoCliente
id_usuario	ID_USUARIO	Número consecutivo único o Login del usuario.	RegistroUsuario
nombre	NOMBRE	Nombre del usuario.	RegistroUsuario
ape_pat	APE_PAT	Apellido paterno del Usuario.	RegistroUsuario
ape_mat	APE_MAT	Apellido materno del Usuario.	RegistroUsuario
fecha_reg	FECHA_REG	En un principio contiene la fecha de inscripción del Usuario, pero en seguida almacena las fechas de reinscripciones.	RegistroUsuario
fecha_insc	FECHA_INSC	Fecha de inscripción del Usuario a la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.	RegistroUsuario
cta_RFC	CTA_RFC	Número de cuenta UNAM (en caso de ser estudiante) o RFC (en caso de ser trabajador) del usuario.	RegistroUsuario
id_categoria	ID_CATEGORIA	Número consecutivo que se utiliza como identificador de categoría del Usuario.	RegistroUsuario
id_idioma	ID_IDIOMA	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.	RegistroUsuario
f_nac	F_NAC	Fecha de nacimiento del Usuario.	RegistroUsuario
calle	CALLE	nombre de la calle + número + edificio + número interior del domicilio del Usuario.	RegistroUsuario
colonia	COLONIA	Nombre de la colonia del domicilio del usuario.	RegistroUsuario
del	DEL	Delegación o Municipio del domicilio del usuario.	RegistroUsuario
C.P.	C_P_	Código Postal del domicilio del Usuario.	RegistroUsuario
tel	TEL	Número de teléfono particular del usuario.	RegistroUsuario
no_insc	NO_INSC	Número de veces que se ha inscrito el Usuario a la Unidad de Aprendizaje Autónomo -Idiomas de la ENEO.	RegistroUsuario
no_acc	NO_ACC	Número de veces que un Usuario ha tenido acceso a la Unidad de Aprendizaje Autónomo - Idiomas de la ENE0.	RegistroUsuario
email	EMAIL	Correo electrónico del Usuario.	RegistroUsuario
Id_carrera	ID_CARRERA	Número consecutivo que se utiliza como identificador de carrera del Usuario.	RegistroUsuario
password	PASSWORD	Palabra clave del Usuario para validar su acceso al SGI-ENEO.	RegistroUsuario
sexo	SEXO	Indica el Sexo del usuario.	RegistroUsuario
id_categoria	ID_CATEGORIA	Número consecutivo que se utiliza como identificador de categoría de Usuario.	Categorías
categoria	CATEGORIA	Categoría a la que pertenecen los Usuarios (Estudiante, Académico, Administrativo, Otro)	Categorías
id_idioma	ID_IDIOMA	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.	Idiomas
idioma	IDIOMA	Idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de	Idiomas

		la ENEO.	
id_carrera	ID_CARRERA	Número consecutivo que se utiliza como identificador de carrera del Usuario.	Carreras
carrera	CARRERA	Carrera cursada por el Usuario.	Carreras
id_acervo	ID_ACERVO	Número consecutivo que identifica al material de acervo, también se conoce como número de adquisición.	RegistroAcervo
clasificacion	CLASIFICACION	Descripción de la Clasificación a la que pertenece el Acervo: código de letras + código de números.	RegistroAcervo
titulo	TITULO	Descripción del Título del material de Acervo: título + subtítulo.	RegistroAcervo
aut_per	AUT_PER	Detalles del Autor del Acervo: apellido (s) + nombre(s) + fecha de nacimiento / muerte.	RegistroAcervo
aut_cor	AUT_COR	Descripción del Autor Corporativo del material de Acervo, que contiene: Nombre de la institución + unidad subordinada + lugar + fecha.	RegistroAcervo
pie_imp	PIE_IMP	Descripción del Pie de Imprenta del Acervo: lugar(es) + editorial + fecha	RegistroAcervo
serie	SERIE	Descripción de la Serie a la que pertenece el material del Acervo: título de la serie + (número | parte | sección) + volumen + issn de la serie	RegistroAcervo
edicion	EDICION	Define el número de edición del material de Acervo.	RegistroAcervo
fondo	FONDO	Describe el Fondo del que proviene el material del Acervo: nombre de la institución + unidad subordinada + lugar.	RegistroAcervo
ISBN	ISBN	Define el International Standar Book Number, asignado a cada material bibliográfico del acervo.	RegistroAcervo
ISSN	ISSN	Define el International Standart Serial Number, asignado al material hemerográfico del acervo.	RegistroAcervo
id_idioma	ID_IDIOMA	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.	RegistroAcervo
id_acento	ID_ACENTO	Número consecutivo que se utiliza como identificador del tipo de acento del acervo.	RegistroAcervo
id_dificultad	ID_DIFICULTAD	Número consecutivo que se utiliza como identificador de la dificultad del acervo.	RegistroAcervo
fecha_ing	FECHA_ING	Fecha en que se registra el material del acervo en el SGI-ENEO.	RegistroAcervo
id_estado	ID_ESTADO	Número consecutivo que se utiliza como identificador del estado del acervo.	RegistroAcervo
catalogador	CATALOGADOR	Iniciales de la persona que realizó la catalogación.	RegistroAcervo
master	MASTER	Campo booleano, que indica si un material es original o copia.	RegistroAcervo
no_copias	NO_COPIAS	En caso de ser master un material, se indica el número de copias que se tiene.	RegistroAcervo
no_paginas	NO_PAGINAS	Número de páginas del material bibliográfico o hemerográfico del acervo.	RegistroAcervo
material_comp	MATERIAL_COMP	Descripción del Material complementario al acervo	RegistroAcervo
duracion	DURACION	Duración del material de audio y/o video, en ***Formato:***o horas, minutos y segundos.	RegistroAcervo

id_soporte	ID_SOPORTE	Número consecutivo que se utiliza como identificador del soporte del acervo.	RegistroAcervo
id_subdiv_tema	ID_SUBDIV_TEMA	Número consecutivo que se utiliza como identificador de la subdivisión temática del acervo.	RegistroAcervo
id_subseccion	ID_SUBSECCION	Número consecutivo que se utiliza como identificador de la subsección del acervo.	RegistroAcervo
id_ubicacion	ID_UBICACION	Número consecutivo que se utiliza como identificador para la ubicación física del material.	RegistroAcervo
id_produccion	ID_PRODUCCION	Número consecutivo que se utiliza como identificador para el tipo de producción del material de acervo.	RegistroAcervo
id_catalogador	ID_CATALOGADOR	Número consecutivo que se utiliza como identificador del catalogador que registra el acervo.	RegistroAcervo
objetivo	OBJETIVO	Descripción del objetivo del acervo.	RegistroAcervo
contenido	CONTENIDO	Descripción del contenido del material de acervo.	RegistroAcervo
estructura	ESTRUCTURA	Estructura del acervo.	RegistroAcervo
sugerencias	SUGERENCIAS	Sugerencias sobre el acervo.	RegistroAcervo
actividades	ACTIVIDADES	Descripción de las actividades que contiene el material de acervo.	RegistroAcervo
respuestas	RESPUESTAS	Campo booleano, verdadero en caso de que el acervo contenga clave de respuestas de ejercicios, falso en caso contrario.	RegistroAcervo
transc	TRANSC	Campo booleano, verdadero en caso de que contenga transcripciones.	RegistroAcervo
descriptores	DESCRIPTORES	Expresa el contenido significativo del material.	RegistroAcervo
asesor	ASESOR	Indica el nombre del Asesor que elaboró la ficha descriptiva del material de Acervo.	RegistroAcervo
Id_master	ID_MASTER	Número consecutivo que se utiliza como identificador del Master al que pertenece la copia del acervo.	RegistroAcervo
id_acento	ID_ACENTO	Número consecutivo que se utiliza como identificador del tipo de acento del acervo.	Acentos
acento	ACENTO	Nombre del acento contenido en el material.	Acentos
id_dificultad	ID_DIFICULTAD	Número consecutivo que se utiliza como identificador de la dificultad del acervo.	CatDificultad
dificultad	DIFICULTAD	Nombre de la dificultad a la que pertenece el material.	CatDificultad
id_estado	ID_ESTADO	Número consecutivo que se utiliza como identificador del estado del acervo.	Estados
estado	ESTADO	Nombre del estado en que se encuentra el material.	Estados
id_soporte	ID_SOPORTE	Número consecutivo que se utiliza como identificador del soporte del acervo.	Soportes
soporte	SOPORTE	Nombre del soporte al que pertenece el material de Acervo	Soportes
id_seccion	ID_SECCION	Número consecutivo que se utiliza como identificador de la sección del acervo.	Secciones
seccion	SECCION	Nombre de la sección a la que pertenece el material de Acervo.	Secciones
id_subseccion	ID_SUBSECCION	Número consecutivo que se utiliza como identificador de la subsección del acervo.	Subsecciones
id_seccion	ID_SECCION	Número consecutivo que se utiliza	Subsecciones

		como identificador de la sección del acervo.	
subseccion	SUBSECCION	Nombre de la subsección a la que pertenece el material.	Subsecciones
id_div_tema	ID_DIV_TEMA	Número consecutivo que se utiliza como identificador de la división temática del acervo.	DivisionTematica
div_tema	DIV_TEMA	Nombre de la división temática contenida en el material.	DivisionTematica
id_subdiv_tema	ID_SUBDIV_TEMA	Número consecutivo que se utiliza como identificador de la subdivisión temática del acervo.	SubdivisionTematica
id_div_tema	ID_DIV_TEMA	Número consecutivo que se utiliza como identificador de la división temática del acervo.	SubdivisionTematica
sub_tema	SUB_TEMA	Nombre de la Subdivisión temática contenida en el material.	SubdivisionTematica
fecha	FECHA	Fecha actual en que se realiza la captura de datos.	UsuariosPorAreas
consulta	CONSULTA	Indica el número de Usuarios registrados en esta fecha en el área de Consulta.	UsuariosPorAreas
audio	AUDIO	Indica el número de Usuarios registrados en esta fecha en el área de Audio.	UsuariosPorAreas
video	VIDEO	Indica el número de Usuarios registrados en esta fecha en el área de Video.	UsuariosPorAreas
multimedia	MULTIMEDIA	Indica el número de Usuarios registrados en esta fecha en el área de Multimedia.	UsuariosPorAreas
id_ubicacion	ID_UBICACION	Número consecutivo que se utiliza como identificador para la ubicación física del material.	Ubicaciones
ubicacion	UBICACION	Nombre del área en que se ubica físicamente al material.	Ubicaciones
id_catalogador	ID_CATALOGADOR	Número consecutivo que se utiliza como identificador del catalogador que registra el acervo.	Catalogadores
catalogador	CATALOGADOR	Indica el nombre del Cliente clasificado como Catalogador del acervo.	Catalogadores
password	PASSWORD	Clave de acceso del Catalogador para registrar un acervo.	Catalogadores

ASOCIACIONES:

NOMBRE	*CÓDIGO*	*TABLA PADRE*	*TABLA HIJA*
Reference_1	REFERENCE_1	TipoCliente	RegistroCliente
Reference_2	REFERENCE_2	Categorias	RegistroUsuario
Reference_3	REFERENCE_3	Idiomas	RegistroUsuario
Reference_4	REFERENCE_4	Carreras	RegistroUsuario
Reference_5	REFERENCE_5	Idiomas	RegistroAcervo
Reference_6	REFERENCE_6	Acentos	RegistroAcervo
Reference_7	REFERENCE_7	CatDificultad	RegistroAcervo
Reference_8	REFERENCE_8	Estados	RegistroAcervo
Reference_9	REFERENCE_9	Soportes	RegistroAcervo
Reference_10	REFERENCE_10	SubdivisionTematica	RegistroAcervo
Reference_11	REFERENCE_11	DivisionTematica	SubdivisionTematica
Reference_12	REFERENCE_12	Subsecciones	RegistroAcervo
Reference_13	REFERENCE_13	Secciones	Subsecciones
Reference_14	REFERENCE_14	Ubicaciones	RegistroAcervo
Reference_15	REFERENCE_15	Catalogadores	RegistroAcervo

DESCRIPCIÓN DE LAS ASOCIACIONES:

ASOCIACIÓN 1:

NOMBRE	Reference_1
CÓDIGO	REFERENCE_1
TABLA PADRE	TipoCliente
TABLA HIJA	RegistroCliente
COLUMNA DE LA TABLA PADRE	id_tipo_cliente
COLUMNA DE LA TABLA HIJA	id_tipo_cliente

ASOCIACIÓN 2:

NOMBRE	Reference_2
CÓDIGO	REFERENCE_2
TABLA PADRE	Categorías
TABLA HIJA	RegistroUsuario
COLUMNA DE LA TABLA PADRE	id_categoría
COLUMNA DE LA TABLA HIJA	id_categoria

ASOCIACIÓN 3:

NOMBRE	Reference_3
CÓDIGO	REFERENCE_3
TABLA PADRE	Idiomas
TABLA HIJA	RegistroUsuario
COLUMNA DE LA TABLA PADRE	id_idioma
COLUMNA DE LA TABLA HIJA	id_idioma

ASOCIACIÓN 4:

NOMBRE	Reference_4
CÓDIGO	REFERENCE_4
TABLA PADRE	Carreras
TABLA HIJA	RegistroUsuario
COLUMNA DE LA TABLA PADRE	id_carrera
COLUMNA DE LA TABLA HIJA	id_carrera

ASOCIACIÓN 5:

NOMBRE	Reference_5
CÓDIGO	REFERENCE_5
TABLA PADRE	Idiomas
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_idioma
COLUMNA DE LA TABLA HIJA	id_idioma

ASOCIACIÓN 6:

NOMBRE	Reference_6
CÓDIGO	REFERENCE_6
TABLA PADRE	Acentos
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_acento
COLUMNA DE LA TABLA HIJA	id_acento

ASOCIACIÓN 7:

NOMBRE	Reference_7
CÓDIGO	REFERENCE_7
TABLA PADRE	CatDificultad
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_dificultad
COLUMNA DE LA TABLA HIJA	id_dificultad

ASOCIACIÓN 8:

NOMBRE	Reference_8
CÓDIGO	REFERENCE_8
TABLA PADRE	Estados
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_estado
COLUMNA DE LA TABLA HIJA	id_estado

ASOCIACIÓN 9:

NOMBRE	Reference_9
CÓDIGO	REFERENCE_9
TABLA PADRE	Soportes
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_soporte
COLUMNA DE LA TABLA HIJA	id_soporte

ASOCIACIÓN 10:

NOMBRE	Reference_10
CÓDIGO	REFERENCE_10
TABLA PADRE	SubdivisionTematica
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_subdiv_tema
COLUMNA DE LA TABLA HIJA	id_subdiv_tema

ASOCIACIÓN 11:

NOMBRE	Reference_11
CÓDIGO	REFERENCE_11
TABLA PADRE	DivisionTematica
TABLA HIJA	SubdivisionTematica
COLUMNA DE LA TABLA PADRE	id_div_tema
COLUMNA DE LA TABLA HIJA	id_div_tema

ASOCIACIÓN 12:

NOMBRE	Reference_12
CÓDIGO	REFERENCE_12
TABLA PADRE	Subsecciones
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_subseccion
COLUMNA DE LA TABLA HIJA	id_subseccion

ASOCIACIÓN 13:

NOMBRE	Reference_13
CÓDIGO	REFERENCE_13
TABLA PADRE	Ubicaciones
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_ubicación
COLUMNA DE LA TABLA HIJA	id_ubicación

ASOCIACIÓN 14:

NOMBRE	Reference_14
CÓDIGO	REFERENCE_14
TABLA PADRE	Secciones
TABLA HIJA	Subsecciones
COLUMNA DE LA TABLA PADRE	id_seccion
COLUMNA DE LA TABLA HIJA	id_seccion

ASOCIACIÓN 15:

NOMBRE	Reference_15
CÓDIGO	REFERENCE_15
TABLA PADRE	Catalogadores
TABLA HIJA	RegistroAcervo
COLUMNA DE LA TABLA PADRE	id_catalogador
COLUMNA DE LA TABLA HIJA	id_catalogador

DICCIONARIO DE DATOS:

TABLA ACENTOS:

Nombre:	id_acento
Código:	ID_ACENTO
Tipo de Dato:	Integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del tipo de acento del acervo.

Nombre:	Acento
Código:	ACENTO
Tipo de Dato:	varchar(15)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción	Nombre del acento contenido en el material.

TABLA CARRERAS:

Nombre:	id_carrera
Código:	ID_CARRERA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción	Número consecutivo que se utiliza como identificador de carrera del Usuario.

Nombre:	Carrera
Código:	CARRERA

Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Carrera cursada por el Usuario.

TABLA CATDIFICULTAD:

Nombre:	id_dificultad
Código:	ID_DIFICULTAD
Tipo de Dato:	Integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la dificultad del acervo.

Nombre:	Dificultad
Código:	DIFICULTAD
Tipo de Dato:	varchar(15)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre de la dificultad a la que pertenece el material.

TABLA CATALOGADORES:

Nombre:	id_catalogador
Código:	ID_CATALOGADOR
Tipo de Dato:	unsigned smallint
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del catalogador que registra el acervo.

Nombre:	Catalogador
Código:	CATALOGADOR
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Indica el nombre del Cliente clasificado como Catalogador del acervo.

Nombre:	Password
Código:	PASSWORD
Tipo de Dato:	varchar(10)
Formato:	
No Permite Modificación:	No
Descripción:	Clave de acceso del Catalogador para registrar un acervo.

TABLA CATEGORÍAS:

Nombre:	id_categoria
Código:	ID_CATEGORIA
Tipo de Dato:	Integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de categoría de Usuario.

Nombre:	Categoria
Código:	CATEGORIA
Tipo de Dato:	Varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Categoría a la que pertenecen los Usuarios (Estudiante, Académico, Administrativo, Otro).

TABLA DIVISIONTEMATICA:

Nombre:	id_div_tema
Código:	ID_DIV_TEMA
Tipo de Dato:	Integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la división temática del acervo.

Nombre:	Div_tema
Código:	DIV_TEMA
Tipo de Dato:	varchar(50)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre de la división temática contenida en el material.

TABLA ESTADOS:

Nombre:	id_estado
Código:	ID_ESTADO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del estado del acervo.

Nombre:	estado
Código:	ESTADO
Tipo de Dato:	varchar(15)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre del estado en que se encuentra el material.

TABLA IDIOMAS:

Nombre:	id_idioma
Código:	ID_IDIOMA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.

Nombre:	idioma
Código:	IDIOMA
Tipo de Dato:	varchar(15)

Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO

TABLA REGISTROACERVO:

Nombre:	id_acervo
Código:	ID_ACERVO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que identifica al material de acervo, también se conoce como número de adquisición.

Nombre:	clasificacion
Código:	CLASIFICACION
Tipo de Dato:	varchar(30)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Descripción de la Clasificación a la que pertenece el Acervo: código de letras + código de números.

Nombre:	titulo
Código:	TITULO
Tipo de Dato:	varchar(200)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Descripción del Título del material de Acervo: título + subtítulo.

Nombre:	aut_per
Código:	AUT_PER
Tipo de Dato:	varchar(100)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Detalles del Autor del Acervo: apellido (s) + nombre(s) + fecha de nacimiento / muerte.

Nombre:	aut_cor
Código:	AUT_COR
Tipo de Dato:	varchar(100)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción del Autor Corporativo del material de Acervo, que contiene: Nombre de la institución + unidad subordinada + lugar + fecha.

Nombre:	pie_imp
Código:	PIE_IMP
Tipo de Dato:	varchar(200)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción del Pie de Imprenta del Acervo: lugar(es) + editorial + fecha.

Nombre:	serie

Código:	SERIE
Tipo de Dato:	varchar(80)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción de la Serie a la que pertenece el material del Acervo: título de la serie + (número | parte | sección) + volumen + issn de la serie

Nombre:	edicion
Código:	EDICION
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Define el número de edición del material de Acervo.

Nombre:	fondo
Código:	FONDO
Tipo de Dato:	varchar(40)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Describe el Fondo del que proviene el material del Acervo: nombre de la institución + unidad subordinada + lugar.

Nombre:	ISBN
Código:	ISBN
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Define el International Standar Book Number, asignado a cada material bibliográfico del acervo.

Nombre:	ISSN
Código:	ISSN
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Define el International Standart Serial Number, asignado al material hemerográfico del acervo.

Nombre:	id_idioma
Código:	ID_IDIOMA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.

Nombre:	id_acento
Código:	ID_ACENTO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del tipo de acento del acervo.

Nombre:	id_dificultad
Código:	ID_DIFICULTAD

Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la dificultad del acervo.

Nombre:	fecha_ing
Código:	FECHA_ING
Tipo de Dato:	datetime
Obligatorio:	No
Formato:	0000/00/00
No Permite Modificación:	No
Descripción:	Fecha en que se registra el material del acervo en el SGI-ENEO.

Nombre:	id_estado
Código:	ID_ESTADO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del estado del acervo.

Nombre:	catalogador
Código:	CATALOGADOR
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Iniciales de la persona que realizó la catalogación.

Nombre:	master
Código:	MASTER
Tipo de Dato:	bit
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Campo booleano, que indica si un material es original o copia.

Nombre:	no_copias
Código:	NO_COPIAS
Tipo de Dato:	integer
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	En caso de ser master un material, se indica el número de copias que se tiene.

Nombre:	no_paginas
Código:	NO_PAGINAS
Tipo de Dato:	integer
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número de páginas del material bibliográfico o hemerográfico del acervo.

Nombre:	material_comp
Código:	MATERIAL_COMP
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	

No Permite Modificación:	No
Descripción:	Descripción del Material complementario al acervo

Nombre:	duracion
Código:	DURACION
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Duración del material de audio y/o video, en formato horas, minutos y segundos.

Nombre:	id_soporte
Código:	ID_SOPORTE
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del soporte del acervo.

Nombre:	id_subdiv_tema
Código:	ID_SUBDIV_TEMA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la subdivisión temática del acervo.

Nombre:	id_subseccion
Código:	ID_SUBSECCION
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la subsección del acervo.

Nombre:	id_ubicacion
Código:	ID_UBICACION
Tipo de Dato:	smallint
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador para la ubicación física del material.

Nombre:	id_produccion
Código:	ID_PRODUCCION
Tipo de Dato:	unsigned smallint
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador para el tipo de producción del material de acervo.

Nombre:	id_catalogador
Código:	ID_CATALOGADOR
Tipo de Dato:	unsigned smallint
Obligatorio:	No
Formato:	
No Permite Modificación:	No

Descripción:	Número consecutivo que se utiliza como identificador del catalogador que registra el acervo.

Nombre:	objetivo
Código:	OBJETIVO
Tipo de Dato:	varchar
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción del objetivo del acervo.

Nombre:	contenido
Código:	CONTENIDO
Tipo de Dato:	varchar
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción del contenido del material de acervo.

Nombre:	estructura
Código:	ESTRUCTURA
Tipo de Dato:	varchar
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Estructura del acervo.

Nombre:	sugerencias
Código:	SUGERENCIAS
Tipo de Dato:	varchar
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Sugerencias sobre el acervo.

Nombre:	actividades
Código:	ACTIVIDADES
Tipo de Dato:	varchar(100)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Descripción de las actividades que contiene el material de acervo.

Nombre:	respuestas
Código:	RESPUESTAS
Tipo de Dato:	bit
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Campo booleano, verdadero en caso de que el acervo contenga clave de respuestas de ejercicios, falso en caso contrario.

Nombre:	transc
Código:	TRANSC
Tipo de Dato:	bit
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Campo booleano, verdadero en caso de que contenga transcripciones.

Nombre:	descriptores
Código:	DESCRIPTORES

Tipo de Dato:	varchar(300)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Expresa el contenido significativo del material.

Nombre:	asesor
Código:	ASESOR
Tipo de Dato:	varchar(40)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Indica el nombre del Asesor que elaboró la ficha descriptiva del material de Acervo.

Nombre:	id_master
Código:	ID_MASTER
Tipo de Dato:	unsigned smallint
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del Master al que pertenece la copia del acervo.

TABLA REGISTROCLIENTE:

Nombre:	id_cliente
Código:	ID_CLIENTE
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo de los clientes con acceso al SGI - ENEO

Nombre:	nombre
Código:	NOMBRE
Tipo de Dato:	varchar(40)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre + Apellido paterno + Apellido materno del cliente.

Nombre:	cargo
Código:	CARGO
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Cargo que ocupa el cliente dentro de la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.

Nombre:	id_tipo_cliente
Código:	ID_TIPO_CLIENTE
Tipo de Dato:	integer
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Identificador del tipo de Cliente (Usuario, Administrador o Coordinador)

Nombre:	telefono

Código:	TELEFONO
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número de teléfono particular del Cliente del SGI-ENEO.

Nombre:	password
Código:	PASSWORD
Tipo de Dato:	varchar(8)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Palabra clave para validar el tipo de Cliente del SGI-ENEO.

Nombre:	notas
Código:	NOTAS
Tipo de Dato:	varchar(100)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Observaciones relacionadas con el Cliente del SGI-ENEO.

TABLA REGISTROUSUARIO:

Nombre:	id_usuario
Código:	ID_USUARIO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo único o Login del usuario.

Nombre:	nombre
Código:	NOMBRE
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre del usuario.

Nombre:	ape_mat
Código:	APE_MAT
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Apellido paterno del Usuario.

Nombre:	ape_pat
Código:	APE_PAT
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Apellido materno del Usuario.

Nombre:	fecha_reg
Código:	FECHA_REG
Tipo de Dato:	datetime
Obligatorio:	Yes
Formato:	

No Permite Modificación:	No
Descripción:	En un principio contiene la fecha de inscripción del Usuario, pero en seguida almacena las fechas de reinscripciones.

Nombre:	fecha_insc
Código:	FECHA_INSC
Tipo de Dato:	datetime
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Fecha de inscripción del Usuario a la Unidad de Aprendizaje Autónomo - Idiomas de la ENEO.

Nombre:	cta_RFC
Código:	CTA_RFC
Tipo de Dato:	varchar(15)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número de cuenta UNAM (en caso de ser estudiante) o RFC (en caso de ser trabajador) del usuario.

Nombre:	id_categoria
Código:	ID_CATEGORIA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de categoría del Usuario.

Nombre:	id_idioma
Código:	ID_IDIOMA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del idioma de estudio en la Unidad Autónoma de Aprendizaje - Idiomas de la ENEO.

Nombre:	f_nac
Código:	F_NAC
Tipo de Dato:	datetime
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Fecha de nacimiento del Usuario.

Nombre:	calle
Código:	CALLE
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	nombre de la calle + número + edificio + número interior del domicilio del Usuario.

Nombre:	colonia
Código:	COLONIA
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	

No Permite Modificación:	No
Descripción:	Nombre de la colonia del domicilio del usuario.

Nombre:	del
Código:	DEL
Tipo de Dato:	varchar(30)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Delegación o Municipio del domicilio del usuario.

Nombre:	C.P.
Código:	C_P_
Tipo de Dato:	varchar(10)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Código Postal del domicilio del Usuario.

Nombre:	tel
Código:	TEL
Tipo de Dato:	varchar(20)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número de teléfono particular del usuario.

Nombre:	no_insc
Código:	NO_INSC
Tipo de Dato:	integer
Unidad:	
Formato:	
No Permite Modificación:	No
Descripción:	Número de veces que se ha inscrito el Usuario a la Unidad de Aprendizaje Autónomo -Idiomas de la ENEO.

Nombre:	no_acc
Código:	NO_ACC
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número de veces que un Usuario ha tenido acceso a la Unidad de Aprendizaje Autónomo - Idiomas de la ENE0.

Nombre:	email
Código:	EMAIL
Tipo de Dato:	varchar(50)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Correo electrónico del Usuario.

Nombre:	password
Código:	PASSWORD
Tipo de Dato:	varchar(5)
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Palabra clave del Usuario para validar su acceso al SGI-ENEO.

Nombre:	id_carrera

Código:	ID_CARRERA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de carrera del Usuario.

Nombre:	sexo
Código:	SEXO
Tipo de Dato:	bit
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Indica el Sexo del usuario.

TABLA SECCIONES:

Nombre:	id_seccion
Código:	ID_SECCION
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la sección del acervo.

Nombre:	seccion
Código:	SECCION
Tipo de Dato:	varchar(30)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre de la sección a la que pertenece el material de Acervo.

TABLA SOPORTES:

Nombre:	id_soporte
Código:	ID_SOPORTE
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador del soporte del acervo.

Nombre:	soporte
Código:	SOPORTE
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre del soporte al que pertenece el material de Acervo.

TABLA SUBDIVISIONTEMATICA:

Nombre:	id_subdiv_tema
Código:	ID_SUBDIV_TEMA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No

Descripción:	Número consecutivo que se utiliza como identificador de la subdivisión temática del acervo.

Nombre:	id_div_tema
Código:	ID_DIV_TEMA
Tipo de Dato:	integer
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la división temática del acervo.

Nombre:	sub_tema
Código:	SUB_TEMA
Tipo de Dato:	varchar(50)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre de la Subdivisión temática contenida en el material.

TABLA SUBSECCIONES:

Nombre:	id_subseccion
Código:	ID_SUBSECCION
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la subsección del acervo.

Nombre:	id_seccion
Código:	ID_SECCION
Tipo de Dato:	integer
Obligatorio:	No
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador de la sección del acervo.

Nombre:	subseccion
Código:	SUBSECCION
Tipo de Dato:	varchar(50)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre de la subsección a la que pertenece el material.

TABLA TIPOCLIENTE:

Nombre:	id_tipo_cliente
Código:	ID_TIPO_CLIENTE
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Identificador del tipo de Cliente (Usuario, Administrador y Coordinador).

Nombre:	tipo_cliente
Código:	TIPO_CLIENTE
Tipo de Dato:	varchar(20)

Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Describe el tipo de Cliente (Coordinador, Administrador o Usuario).

TABLA UBICACIONES:

Nombre:	id_ubicacion
Código:	ID_UBICACION
Tipo de Dato:	smallint
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Número consecutivo que se utiliza como identificador para la ubicación física del material.

Nombre:	ubicacion
Código:	UBICACION
Tipo de Dato:	varchar(20)
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Nombre del área en que se ubica físicamente al material.

TABLA USUARIOSPORAREAS:

Nombre:	fecha
Código:	FECHA
Tipo de Dato:	datetime
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Fecha actual en que se realiza la captura de datos.

Nombre:	consulta
Código:	CONSULTA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Indica el número de Usuarios registrados en esta fecha en el área de Consulta.

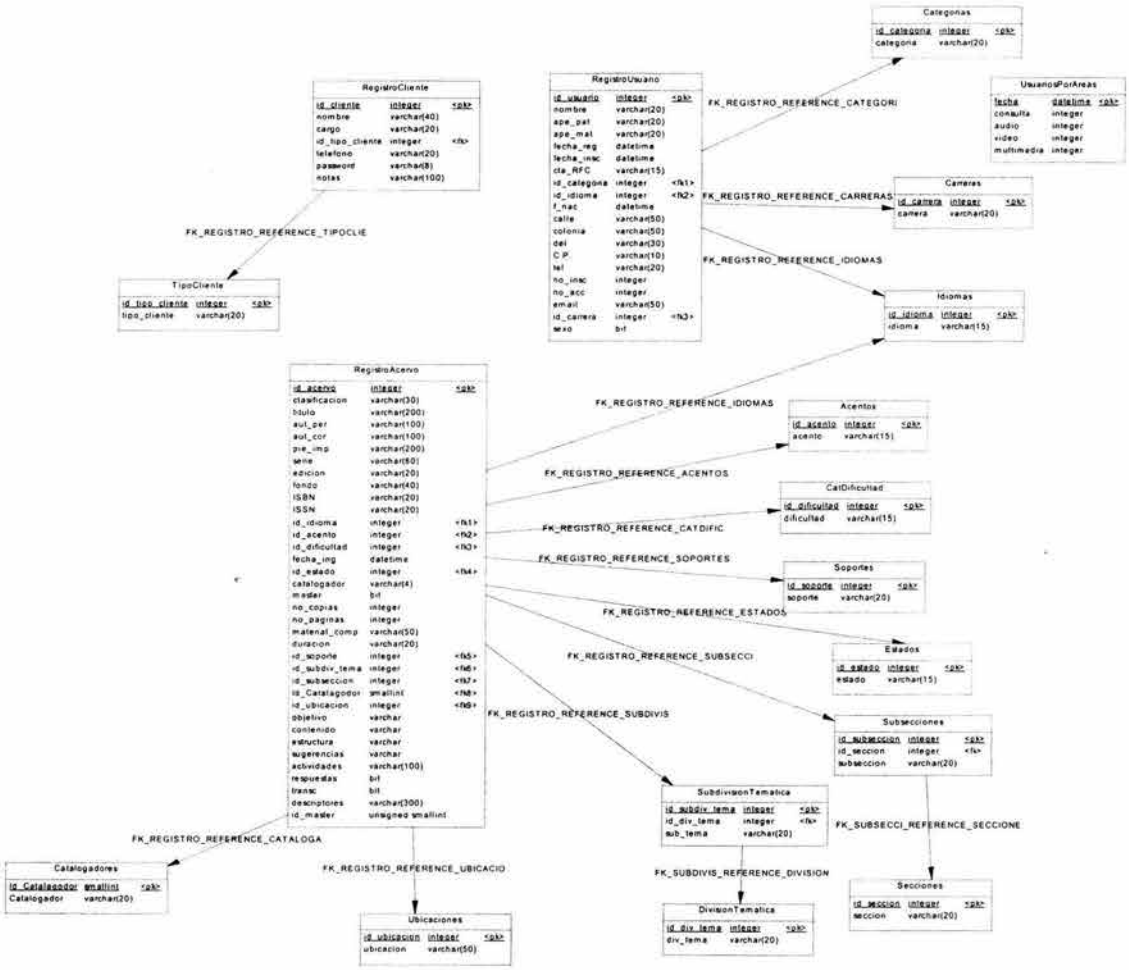
Nombre:	audio
Código:	AUDIO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Indica el número de Usuarios registrados en esta fecha en el área de Audio.

Nombre:	video
Código:	VIDEO
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Indica el número de Usuarios registrados en esta fecha en el área de Video.

Nombre:	multimedia

Código:	MULTIMEDIA
Tipo de Dato:	integer
Obligatorio:	Yes
Formato:	
No Permite Modificación:	No
Descripción:	Indica el número de Usuarios registrados en esta fecha en el área de Multimedia.

DIAGRAMA DE LA BASE DE DATOS DEL SGI-ENEO



CONCLUSIONES

Con el análisis orientado a objetos, se modela el mundo en términos de tipos de objetos y lo que le ocurre o hacen éstos. Los modelos que se construyen reflejan la realidad de modo más natural que los del análisis estructurado. Mediante las técnicas orientadas a objetos, construimos sistemas que modelan más fielmente el mundo real. Cuando el mundo real cambia, nuestro sistema es más fácil de cambiar.

Por tanto el uso de una metodología orientada a objetos que proporcione una continuidad entre sus distintas fases y que asegure algo tan importante como la rastreabilidad de los resultados, es fundamental para el diseño de sistemas de gestión.

El punto más fuerte del sistema es la modularidad que posee. Esta modularidad es la proporcionada por la metodología "Objectory" el cual divide a un sistema en tres modelos ó dimensiones que deben ser lo más independiente entre sí. En el sistema SGI – ENEO las dimensiones se identificaron e implementaron claramente, la dimensión de presentación quedó implementada en las clases pantalla e interface, la dimensión de control en las clases manejador y la dimensión de información en las clases entidad. La metodología "Objectory" es muy apropiada para el análisis de requisitos de sistemas porque es fácil de aprender y su aplicación resulta relativamente sencilla.

Uno de los principales problemas que se encuentra al desarrollar un sistema de software que administre información es definir su función y presentación ante el usuario. Esto se debe primordialmente a que las personas que solicitan la implementación de un sistema de este tipo, en la mayoría de los casos, no proporcionan la información necesaria para la construcción del sistema de software. En el caso del sistema SGI – ENEO hubo muchos cambios en los requisitos iniciales, la rastreabilidad entre las distintas fases que tiene la metodología permitió hacer los cambios de una manera sencilla y sin modificar otras partes.

El sistema representado a través de los casos de uso es la forma más simple de verlo; son esenciales para entender el flujo de información en el sistema, así como la interacción de los usuarios en él. De ellos depende el resto del sistema, por ejemplo los diagramas de secuencia hechos durante el análisis serían obsoletos si el caso de uso que están implementando es erróneo y a su vez repercutiría en los análisis para obtener responsabilidades basados en los diagramas de secuencia y en los casos de uso. Por eso se les dio mucho énfasis.

Al concluir el análisis del sistema e iniciar el diseño la complejidad aumentó drásticamente ya que se tenía que detallar aún más los componentes anteriores (diagramas de secuencia, casos de uso complementados). La obtención de responsabilidades para el llenado de las tarjetas de clase fue la parte más laboriosa del sistema ya que definimos prácticamente todo lo que el sistema tenía que hacer a través de responsabilidades. Una herramienta que fue básica para el desarrollo del sistema fue la herencia de clases ya que nos evitó escribir el mismo código en clases semejantes.

Para la ENEO la contribución que tendrá este Sistema, radica principalmente en la facilidad y agilidad con que se llevarán a cabo los procesos administrativos de la UAAI y los Administradores de la UAAI tendrán un control preciso sobre la información.

La experiencia que nos dio el desarrollo de este Sistema, fue el hecho de tratar un problema real, crear una solución, implementarla y lo más importante, el contacto y la comunicación con un cliente real.

BIBLIOGRAFÍA:

Deitel H. M., Deitel P. J., *Cómo Programar en Java*, 1ª. ed., México, Ed. Prentice Hall, 1998.

Lemay L., Cadenhead R., *Aprendiendo Java 2 en 21 días*, 1ª. ed., México, Ed. Prentice Hall, 1999.

Licenciatura en Enfermería y Obstetricia Plan de Estudios, México, ENEO – UNAM, 2000.

Stephens R. K., Plew R., Morgan B., *Teach Yourself SQL In 21 Days*, 2ª. ed., Estados Unidos de América, Ed. Macmillan Computer Publishing, 2000.

Programmer's Reference JConnect for JDBC 4.5 and 5.5, Estados Unidos de América, Sybase Inc., 2001.

Eckstein R., Loy M., Wood D., *JAVA Swing*, 1ª. ed., Estados Unidos de América, Ed. O'Reilly & Associates, Inc., 1998.

Pratik P., *Java Database Programming with JDBC*, 1ª. ed., Estados Unidos de América, Ed. The Coriolis Group, 1996.

Freinet, Célestin. *Técnica de la Escuela Moderna.* Siglo XXI. México, 1969.

Villalobos, Oscar. *Los Centros de Recursos para el Aprendizaje.* UNED. Costa Rica, 1993.

Neill, A. S. Summerhill. *Un punto de vista radical sobre la educación de los niños.* Fondo de Cultura Económica. México, 1963.

Lapassade, Georges. *Autogestión Pedagógica.* Gedisa. España, 1977.

Carvalho, Dorothea. *Self-access: Appropiate Material.* The British Council. Inglaterra, 1993.

REFERENCIAS DE WEB:

Suarez L., *El paquete java.io. Maneja de las I/O*, 1ª. ed., URL: http://www.javahispano.org, Julio 2001.

Otero A., *El ABC de JDBC*, 1ª. ed., URL: http://www.javahispano.org, 2003.

Execute an external program – Real's Java How-To, URL: http://www.rgagnon.com/javadetails/java-0014.html.

Sybase Inc., URL: http://www.sybase.com

Sun Microsystems Inc. Java Technology, URL: http:// java.sun.com

API Specifications Java 2 Plataform SE. v1.3.1, URL : http://java.sun.com/j2se/1.3/docs/api/index.html

CURSO JAVA El sonido en jdk 1.3: javax.swing.sampled, URL: http://www.htmlpoint.com/guidajava/java_36.htm

Metodología OBJECTORY:
http://www.cs.ualberta.ca/~pfiguero/soo/metod/objectory.html
http://www.fciencias.unam.mx/~gig/temario/EjemplosSVE/Metodo-1.html
http://nodo.uagrm.edu.bo/si/Grupo-10/LOS%20CASOS%20DE%20USO.htm

http://www.um.es/~giisw/docs/pfc99-00.doc
http://www.mcc.unam.mx/~cursos/Objetos/Temario/temario.html
http://www.cs.ualberta.ca/~pfiguero/soo/metod/objectory.html
http://www.cs.ualberta.ca/~pfiguero/soo/metod/omt.html
www.mcc.unam.mx/~cursos/Objetos/Temario/temario.html
CENTROS DE AUTOACCESO:

http://www.ub.es/filhis/culturele/montalto.html
http://www.dgbiblio.unam.mx/servicios/dgb/publicdgb/bole/fulltext/voll1/cele.html
http://www.latarea.com.mx/articu/articu11/azata111.htm#not6
http://caguenche.tripod.com/filosofia.htm
http://www.britishcouncil.org/mexico/spanish/english/sachist.htm
http://www.britishcouncil.org/mexico/spanish/english/sacport.htm
http://sic.uji.es/publ/edicions/jfi2/giapel.pdf
http://lenguas.uam.mx/descargas/guias/guia01.pdf

CELE:
http://cele.unam.mx
http://www.estadistica.unam.mx/memorias/2000/2000/cele.htm

ENEO:
http://dgedi.estadistica.unam.mx/memo96/eneo.htm
http://www.imbiomed.com.mx/Enfer/Env6n1/espanol/Wen81-01.html
http://www.estadistica.unam.mx/memorias/2000/2000/eneo.htm
http://www.universia.net.mx/contenidos/bibliotecas/Bibliotecas.htm

DESCRIPCIÓN DE CASOS DE USO DEL SGI-ENEO CREADOS

Caso de uso:	**Validar Usuario.**
Actores:	Usuario, Base de Datos Cliente.
Tipo:	Inclusión – Generalización.
Propósito:	Validará si el registro de un Usuario es vigente para el uso del SGI-ENEO
Resumen:	Este caso es iniciado por un Usuario. Valida al Usuario mediante un *login* y *password* los cuales le fueron proporcionados con su Credencial, esta validación la realiza comparando los datos obtenidos del *login* y *password* con dos campos homónimos de la Tabla Registro Usuario de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: validar Cliente.
Flujo principal:	Se presenta al Usuario la pantalla P–3. El Usuario deberá introducir su *login* y *password* para verificar si puede entrar a *la Unidad Para el Aprendizaje Autónomo – Idiomas de la Escuela Nacional de Enfermería y Obstetricia* si se ha vencido el tiempo permitido o no se encuentra registrado en la Base de Datos se enviara un mensaje a pantalla de su estado; si su registro es valido igualmente se mostrará un mensaje de permiso de entrada a la unidad. Posteriormente se continua con el caso de uso *Validar Usuario*. Este caso de uso es finalizado por un Administrador o Coordinador, mediante una combinación de teclas.
Subflujos:	Ninguno.
Excepciones:	Ex–1 no hubo validación, su registro no existe, se manda un mensaje de error, y enseguida se solicita al Usuario volver a validarse

Caso de uso:	**Consultar Acervo.**
Actores:	Usuario, Base de Datos.
Tipo:	Básico.
Propósito:	Poder consultar la información contenida (Libro, Publicación periódica, Audio casete, videocasete, CD, Ficha de trabajo, TV) en *la Unidad Para el Aprendizaje Autónomo – Idiomas de la Escuela Nacional de Enfermería y Obstetricia*
Resumen:	Este caso es iniciado por un Usuario. El Usuario puede consultar la información del Acervo de *la Unidad Para el Aprendizaje Autónomo – Idiomas de la Escuela Nacional de Enfermería y Obstetricia*
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: validar Usuario y validar Cliente.
Flujo principal:	Después de ejecutar *Ofrecer servicios*, si se escogió consultar Acervo en el menú Usuarios se ejecutará la pantalla P–4(a).
Subflujos:	*Consultar Acervo* El Usuario puede seleccionar entre las siguientes actividades:"Buscar", "Salir". Si la actividad seleccionada es "Buscar" se ejecutará la búsqueda (E–1) dependiendo de los campos seleccionados por el Usuario, la búsqueda se definirá con menús desplegables para los campos: tema, idioma, nivel de dificultad y tipo de material, el Usuario podrá introducir dos cadenas para los campos título, autor, ISBN/ISSN, para estos dos campos que se desee buscar se tendrán que colocar opciones O e Y. Se presentara una pantalla P–4 (b) (E–2) (la pantalla presentará la opción regresar con la cual volverá a la pantalla P–4(a)), con los resultados de la consulta. Si la actividad seleccionada es "Salir" se ejecutará Caso de Uso *Ofrecer Servicios*.
Excepciones:	E – 1 Si no se ha llenado el primer campo de búsqueda no se podrá realizar la consulta. E – 2 Si no se encontraron objetos que cumplieran con las especificaciones de la búsqueda no se desplegará la pantalla de resultados

Caso de uso:	**Inscripción.**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Permite a un Administrador o Coordinador registrar a un Usuario para otorgarle una Credencial, su *login* y su *password*, para que este acceda a *la Unidad Para el Aprendizaje Autónomo – Idiomas de la Escuela Nacional de Enfermería y Obstetricia*
Resumen:	Este caso es iniciado por un Administrador o un Coordinador. Ellos son los que crean el registro del Usuario en la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Validar Cliente*.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió la opción Inscripción del menú Administradores se ejecutará la pantalla P–5, el Administrador puede seleccionar entre las siguientes actividades: "Registrar", "Salir". Si la actividad seleccionada es "Registrar" el sistema genera un nuevo registro de Usuario(E–1) se continua con el caso de uso *Administrar Usuario subflujo* (S-1) Si la actividad seleccionada es "Salir" se ejecuta el caso de uso *Ofrecer Servicios*.
Subflujos:	Ninguno.
Excepciones:	E–1 *Información Incompleta*. Falta llenar información en el registro de Usuario. Se vuelve a solicitar al Usuario que complete el registro.

Caso de uso:	**Imprimir Credencial.**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Inclusión.
Propósito:	Permite a un Coordinador ó Administrador imprimir la Credencial de un Usuario para que este pueda validarse en *la Unidad Para el Aprendizaje Autónomo – Idiomas de la Escuela Nacional de Enfermería y Obstetricia* y así poder realizar sus actividades en este lugar.
Resumen:	Este caso es iniciado por un Administrador ó un Coordinador. Ellos imprimen la Credencial del Usuario.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Administrar Usuario*.
Flujo principal:	Después de ejecutar *Administrar Usuario* se muestra la pantalla P–7, el Administrador ó Coordinador, puede seleccionar entre las siguientes actividades: "Imprimir", "Cancelar". Si la actividad seleccionada es "Imprimir" el sistema genera una Credencial a partir de los datos contenidos en los campos de la pantalla P–7(a) se continua con el caso de uso *Administrar Usuario subflujo* (S-1) Si la actividad seleccionada es "Cancelar" se ejecuta el caso de uso *Administrar Usuario subflujo* (S-1).
Subflujos:	Ninguno.
Excepciones:	Ninguno.

Caso de uso:	**Ofrecer Servicios**
Actores:	Cliente, Usuario, Administrador, Coordinador, Base de Datos.
Tipo:	Inclusión
Propósito:	Proporciona una pantalla en la que se presentará los servicios dependiendo del tipo de Cliente (Usuario, Administrador, Coordinador)
Resumen:	Este caso es iniciado por un Cliente. Muestra las opciones para utilizar el sistema SGI – ENEO.
Precondiciones:	Debe de haberse validado el Cliente en todos los subflujos excepto el S – 1.
Flujo principal:	Se ejecuta el caso de uso Ofrecer Servicios y dependiendo del tipo de Cliente, se continuara con los diversos subflujos de este caso de uso.
Subflujos:	S – *1 Sin Servicio.* Se presenta al Cliente la pantalla P – 2. El Cliente puede seleccionar entre las siguientes opciones de un menú desplegable llamado Inicio, dejando inhabilitados el resto de los menús. El Cliente puede seleccionar de este menú las siguientes opciones: "Validación" y "Salir". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. Si la actividad seleccionada es "Salir" sale del sistema.
	S – 2 *Ofrecer Servicios a Usuarios.* Se presenta al Usuario la pantalla P–2. El Usuario puede seleccionar los siguientes menús desplegables: Inicio, Usuarios, dejando inhabilitados el resto de los menús. El Usuario puede seleccionar del menú Inicio la opción: "Validación". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. El Usuario puede seleccionar del menú Usuario las siguientes opciones: "Validar Usuario", "Consultar Acervo". Si la actividad seleccionada es "Validar Usuario" se continúa con el caso *Validar Usuario*. Si la actividad seleccionada es "Consultar Acervo" se continúa con el caso de uso *Consultar Acervo*.
	S – 3 *Ofrecer Servicios Administradores* Se presenta al Administrador la pantalla P–2. El Administrador puede seleccionar los siguientes menús desplegables: Inicio, Administrador, dejando inhabilitados el resto de los menús. El Administrador puede seleccionar del menú Inicio las siguientes opciones: "Validación" y "Salir". Si la actividad seleccionada es "Validación" se continúa con el caso de uso *Validar Cliente*. Si la actividad seleccionada es "Salir" sale del sistema. El Administrador puede seleccionar del menú Administradores las siguientes opciones: "Administrar Acervo(AA)", "Administrar Usuario(AU)", "Reporte de Acervo(RA)", "Reporte Usuarios (RU) " y "Administrar Usuarios por Área (AU)". Si la actividad seleccionada es "AA – Administrar Acervo" se continúa con el caso de uso *Administrar Acervo*. Si la actividad seleccionada es "AU – Inscripción" se continúa con el caso de uso *Inscripción*. Si la actividad seleccionada es "AU – Administrar Usuario" se continúa con el caso de uso *Administrar Usuario*. Si la actividad seleccionada es "R - Usuario" se continúa con el caso de uso *Generar Reporte Usuario*. Si la actividad seleccionada es "R - Acervo" se continúa con el caso de uso *Generar Reporte Acervo*. Si la actividad seleccionada es "AU – Administrar Usuarios por área" se continúa con el caso de uso *Capturar Usuarios por Área*.
	S – 4 *Ofrecer Servicios Coordinadores* Se presenta al Administrador la pantalla P–2. El Coordinador puede seleccionar los siguientes menús desplegables: Inicio, Coordinador, dejando inhabilitados el resto de los menús. El Coordinador puede seleccionar las mismas opciones que el Administrador pero se agrega un nuevo item "Administrar Cliente".
Excepciones:	Ninguno

Caso de uso:	**Administrar Usuario.**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Permite a un Administrador o Coordinador administrar un Registro de Usuario para actualizarlo o imprimir su Credencial.
Resumen:	Este caso es iniciado por un Administrador o un Coordinador. Ellos son los que crean el Registro del Usuario en la Base de Datos Cliente.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Validar Cliente*.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió la opción modificar Usuario, seguirá la ejecución del caso de uso Administrar Usuario subflujo (S–1).
Subflujos:	*S – 1 Administrar Registro Usuario.* Se presenta al Coordinador ó Administrador la Pantalla Obtener Registro Usuario (P-6 (a)) con los campos del Usuario. El Coordinador ó Administrador podrá seleccionar entra las siguientes actividades: "Eliminar", "Modificar", "Actualizar"(inactivo hasta que se haga clic en "Modificar"), "Búsqueda", "Primer Registro", "Último Registro", "Siguiente", "Anterior", "Imprimir" , "Reinscribir" (inactivo hasta que haya expirado el período de vigencia) y "Salir". Si el Coordinador ó Administrador presiona "Modificar" se ejecuta el subflujo *Actualizar Registro Usuario* (*S*-2). Si el Coordinador ó Administrador selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Usuario* (*S*-3). Si el Coordinador ó Administrador selecciona "Búsqueda" se ejecuta el subflujo *Búsqueda* (*S*-4). Si el Coordinador ó Administrador selecciona "Primer Registro" se ejecuta el subflujo *Recupera primer Registro* (*S*-5). Si el Coordinador ó Administrador selecciona "Último Registro" se ejecuta el subflujo *Recupera último Registro* (*S*-6). Si el Coordinador ó Administrador selecciona "Siguiente" se ejecuta el subflujo *Recupera Registro siguiente* (*S*-7). Si el Coordinador ó Administrador selecciona "Anterior" se ejecuta el subflujo *Recupera Registro anterior* (*S*-8). Si el Coordinador ó Administrador selecciona "Reinscribir" se ejecuta el subflujo *Reinscribir Usuario* (*S*-9). Si el Coordinador ó Administrador presiona "Imprimir" se continúa con el caso de uso *Imprimir Credencial*. Si la actividad seleccionada es "Salir" se ejecutará el caso de uso *Ofrecer Servicios* (si aún no se ha presionado "Actualizar", la nueva información será pérdida).
	S-2 *Actualizar Registro Usuario* Se activa el botón "Actualizar", el Coordinador ó Administrador introduce los nuevos datos, enseguida da clic en "Actualizar" el cual actualiza el Registro de Usuario con la información modificada (*E*-1), en caso de no hacer clic en el último botón mencionado, la nueva información no será almacenada. Se continúa con el subflujo *Administrar Registro Usuario* (*S*-1).
	S-3 *Eliminar Registro Usuario* Se elimina el Registro de Usuario y se continúa con el subflujo *Administrar Registro Usuario* (*S*-1).

	S – 4 Búsqueda. Se presenta la pantalla P–6(b) la cual pide el campo del Usuario con el cual se realizará la búsqueda. Se presenta las siguientes opciones: "Buscar" y "Cancelar". Si la actividad seleccionada es "Buscar" se verifica que el dato proporcionado exista en la Tabla Registro Usuario de la Base de Datos, si este no se encuentra se muestra un mensaje indicando que es incorrecto que introduzca un nuevo dato. Si el dato existe se continua con el subflujo *Modificar Usuario* (S–1)con los datos del Usuario buscado. Si la actividad seleccionada es "Cancelar", se ejecuta el caso de uso *Administrar Usuario* (S–1).
	S – 5 Recuperar Primer Registro. Se presenta la pantalla P–6(a) con los datos de los campos del primer Registro Usuario de la Tabla Registro Usuario de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Usuario* (S–1).
	S – 6 Recuperar Último Registro. Se presenta la pantalla P–6(a) con los datos de los campos del último Registro Usuario la Tabla Registro Usuario de la Base de Datos, posteriormente se continua con el subflujo *Administrar Registro Usuario* (S–1).
	S – 7 Recuperar Registro Siguiente. Se presenta la pantalla P–6(a) con los datos de los campos del siguiente Registro Usuario de la Tabla Registro Usuario de la Base de Datos, respecto al que se presentaba antes, posteriormente se continua con el subflujo *Administrar Registro Usuario* (S–1).
	S – 8 Recuperar Registro Anterior. Se presenta la pantalla P–6(a) con los datos de los campos del anterior Registro Usuario de la Tabla Registro Usuario de la Base de Datos, respecto al que se presentaba antes, posteriormente se continua con el subflujo *Administrar Registro Usuario* (S–1).
	S – 9 Reinscribir Usuario Se presenta la pantalla P–6(a) con los datos de los campos del Registro Usuario de la Tabla Registro Usuario de la Base de Datos, y con un recuadro mediante el cual el sistema avisará que el Usuario ha sido reinscrito.
Excepciones:	*E – 1 información incompleta.* Falta llenar información en el Registro de Usuario. Se vuelve a solicitar al Usuario que complete el Registro.

Caso de uso:	**Administrar Cliente.**
Actores:	Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Permite a un Coordinador modificar o crear un registro de un Cliente (Administrador, Coordinador, Usuario) para actualizarlo o crear un registro nuevo.
Resumen:	Este caso es iniciado por el Coordinador, quién crea el Registro del Administrador, Coordinador ó Usuario en la Tabla Registro Cliente de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: *Validar Cliente*.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió la opción Modificar Cliente, seguirá la ejecución del caso de uso *Administrar Registro Cliente subflujo* (S–1).
subflujos:	S – 1 *Administrar Registro Cliente.* Se presenta al Coordinador la Pantalla Obtener Registro Cliente (*P-9(a)*) con los campos vacíos El Coordinador podrá seleccionar entra las siguientes actividades: "Eliminar", "Modificar", "Actualizar"(inactivo hasta que se haga clic en "Modificar"), "Búsqueda", "Primer registro", "Último registro", "Siguiente", "Anterior", "Crear" y "Salir". Si el Coordinador presiona "Modificar" se ejecuta el subflujo *Actualizar Registro Cliente*(*S*-2). Si el Coordinador selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Cliente* (*S*-3). Si el Coordinador selecciona "Búsqueda" se ejecuta el subflujo *Búsqueda Registro Cliente* (*S*-4). Si el Coordinador selecciona "Primer Registro" se ejecuta el subflujo *Recupera Primer Registro Cliente* (*S*-5). Si el Coordinador selecciona "Último registro" se ejecuta el subflujo *Recupera Último Registro Cliente* (*S*-6). Si el Coordinador selecciona "Siguiente" se ejecuta el subflujo *Recupera Registro Siguiente Cliente* (*S*-7). Si el Coordinador selecciona "Anterior" se ejecuta el subflujo *Recupera Registro Anterior Cliente* (*S*-8). Si el Coordinador selecciona "Crear" se ejecuta el subflujo *Crear Registro Cliente* (*S*-9). Si la actividad seleccionada es "Salir" se ejecutará el caso de uso *Ofrecer Servicios* (si aún no se ha presionado "Actualizar", la nueva información será perdida).
	S-2 *Actualizar Registro Cliente* Se activa el botón "Actualizar", el Coordinador introduce los nuevos datos, enseguida da clic en "Actualizar" el cual actualiza el Registro del Cliente con la información modificada (*E*-1), en caso de no hacer clic en el último botón mencionado, la nueva información no será almacenada. Se continúa con el subflujo *Administrar Registro Cliente(S – 1)*.
	S-3 *Eliminar Registro Cliente.* Se elimina el Registro de Cliente y se continúa con el subflujo *Administrar Registro Cliente* (*S*-1).

	S – 4 Búsqueda Registro. Se presenta la pantalla P–9(b) la cual pide los datos del registro a buscar, esto puede realizarse con cualquier campo y operador. Se presenta las siguientes opciones: "Buscar" y "Cancelar". Si la actividad seleccionada es "Buscar" se verifica que los datos proporcionados existan en la Tabla Registro Cliente de la Base de Datos, si estos no se encuentran, se muestra un mensaje indicando que el registro no se encuentra o que se ha introducido mal la información para la búsqueda, por lo que se pide que, se introduzca nueva información para una nueva búsqueda. Si el registro existe se continua con el subflujo *Modificar Registro Cliente* (S–1). Si la actividad seleccionada es "Cancelar", se ejecuta el caso de uso *Administrar Cliente* (S–1).
	S – 5 *Recuperar Primer Registro.* Se presenta la pantalla P–9(a) con los campos con los datos del Primer registro de la Tabla Registro Cliente de la Base de Datos, posteriormente se continua con el subflujo *Administrar Cliente* (S–1).
	S – 6 Recuperar Último Registro. Se presenta la pantalla P–10(a) con los campos con los datos del Último registro de la Tabla Registro Cliente de la Base de Datos, posteriormente se continua con el subflujo *Administrar Cliente* (S–1).
	S – 7 Recuperar Registro Siguiente. Se presenta la pantalla P–9(a) con los campos con los datos del Siguiente registro de la Tabla Registro Cliente de la Base de Datos, respecto al que se presentaba antes, posteriormente se continua con el subflujo *Administrar Cliente* (S–1).
	S – 8 Recuperar Registro Anterior. Se presenta la pantalla P–9(a) con los campos con los datos del Anterior registro de la Tabla Registro Cliente de la Base de Datos, respecto al que se presentaba antes, posteriormente se continua con el subflujo *Administrar Cliente* (S–1).
	S – 9 Crear Registro. Se presenta la pantalla P–9(c) la cual tienen dos opciones "Crear" y "Cancelar". Si la actividad seleccionada es "Crear" se crea un nuevo registro en la Tabla Registro Cliente de la Base de Datos (E–1) con los datos proporcionados anteriormente en los distintos campos, posteriormente se continua con el subflujo *Administrar Cliente* (S–1). Si la actividad seleccionada es "Cancelar" se ejecuta el caso de uso *Administrar Cliente* (S–1).
Excepciones:	E – 1 *Información Incompleta.* Falta llenar información en el registro de Cliente. Se vuelve a solicitar al Coordinador que complete el registro.

Caso de uso:	**Generar Reporte Usuario**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Poder generar reportes de la información contenida en la Tabla Registro Usuario de la Base de Datos.
Resumen:	Este caso es iniciado por un Administrador ó un Coordinador. Estos pueden generar reportes de la información contenida en la tabla Registro Usuario de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: Validar Cliente.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió Generar Reportes Usuario, se ejecuta el subflujo *Opciones de Reporte Usuario*(S – 1) del caso de uso Generar Reporte Usuario.
Subflujos:	*S – 1 Opciones de Reporte Usuario* Se presenta la pantalla P-10(a), donde el Administrador ó Coordinador introduce un rango de fechas para el cual se realizará el reporte, está pantalla contiene tres botones los cuales son: "Inscripción por Categoría", "Usuarios por Áreas" y "Salir". Si el Administrador ó Coordinador presiona "Inscripción por Categoría" se continua con el subflujo *Inscripción por Categoría (S – 2) (E –2) (E –3).* Si el Administrador ó Coordinador presiona "Usuarios por Áreas" se continua con el subflujo *Usuarios por Áreas (S – 3) (E –2) (E –3).* Si la actividad seleccionada es "Salir" se continua con el caso de uso *Ofrecer Servicios.*
	S – 2 *Inscripción por Categoria* Se presenta la pantalla P–10(b), con los resultados de número y porcentajes de Usuarios inscritos por categoría en la cual el Administrador ó Coordinador puede seleccionar "Guardar", "Imprimir" y "Regresar". Si la actividad seleccionada es "Guardar" se muestra la pantalla P-10(c), en la cual el Administrador ó Coordinador indicará el nombre, la ruta y formato del archivo que contendrá la información del reporte, esta pantalla contiene dos botones: "Guardar" y "Cancelar" si el Coordinador ó Administrador presiona "Guardar" (E–1) el archivo se guardará y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Usuario*, si escoge "Cancelar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Usuario.* Si la actividad seleccionada es "Imprimir" se imprime el reporte mostrado y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Usuario.* Si la actividad seleccionada fue "Regresar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Usuario.*
	S-6 Usuarios por Áreas Se presenta la pantalla P–10(d), con los resultados de número y porcentajes de Usuarios por áreas utilizas en la cual el Administrador ó Coordinador puede seleccionar "Guardar", "Imprimir" y "Regresar". Si la actividad seleccionada es "Guardar" se muestra la pantalla P-11(e), en la cual el Administrador ó Coordinador indicará el nombre, la ruta y formato del archivo que contendrá la información del reporte, esta pantalla contiene dos botones "Guardar" y "Cancelar" si el Coordinador ó Administrador presiona "Guardar" (E–1) el archivo se guardará y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Usuario*, si escoge "Cancelar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Usuario.* Si la actividad seleccionada es "Imprimir" se imprime el reporte mostrado y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Usuario.* Si la actividad seleccionada fue "Regresar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Usuario.*
Excepciones:	E–1 Si no se ha puesto nombre al archivo o no se ha indicado la ruta. E – 2 Campo fecha vacía E – 3 fecha no válida

Caso de uso:	Administrar Usuario por Área.
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Administrar la información concerniente al número de Usuarios que usaron las áreas, por día.
Resumen:	Este caso es iniciado por un Administrador ó un Coordinador. Estos capturan la información de número de Usuarios por área al día, contenida en la Tabla de Usuarios Por Área de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: validar Cliente.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió Administrar Usuarios por Área se ejecuta el subflujo (S–1) del caso de uso Administrar Usuario por Área.
Subflujos:	*S – 1 Opciones de Administrar Usuario* Se presenta la pantalla P-11(a), donde el Administrador ó Coordinador introduce el número de Usuarios que utilizaron ese día las diferentes áreas: Comprensión Auditiva, Pronunciación, Cómputo, Video y Consulta. Esta pantalla ofrece tres opciones: "Guardar", "Modificar", y "Salir". Si el Administrador ó Coordinador presiona "Guardar" se continua con el subflujo *Guardar Usuarios por Área (S-2).* Si el Administrador ó Coordinador presiona "Modificar" se continua con el subflujo *Modificar Usuarios por Área (S-3)* Si la actividad seleccionada es "Salir" se continua con el caso de uso *Ofrecer Servicios.*
	S – 2 *Guardar Usuarios por Área* Se guarda la información (E –1) en la Tabla Usuarios Por Áreas de la Base de Datos y se continua con el subflujo (S – 1) del caso de uso *Administrar Usuarios por Área.*
	S – 3 *Modificar Usuarios por Área* Se presenta la pantalla P–11(b), en la cual se indica la fecha del día del cual se quiere modificar la información concerniente a Usuarios por Área, esta pantalla contiene dos botones: "Modificar" y "Regresar" Si la actividad seleccionada es "Modificar" (E–2)(E–3) se muestra la pantalla P-11(c), con los campos de las áreas conteniendo información, la cual será modificada, esta pantalla tiene dos opciones "Guardar" y "Cancelar". Si la actividad seleccionada es "Guardar" (E–1) la nueva información será almacenada y se continua con el subflujo (S–1) del caso de uso *Administrar Usuarios por Área.* Si se elige "Cancelar" se continua con el subflujo (S–1) del caso de uso *Administrar Usuarios por Área.* Si se escoge la opción "Regresar" de la pantalla P–11(b) se continua con el subflujo (S–1) del caso de uso *Administrar Usuarios por Área.*
Excepciones:	E–1 no se llenaron todos los campos. E – 2 no se indicó fecha E – 3 fecha no válida.

Caso de uso:	**Generar Reporte Acervo**
Actores:	Administrador, Coordinador, Base de Datos.
Tipo:	Básico.
Propósito:	Poder generar reportes de la información contenida en la Tabla Registro Acervo de la Base de Datos.
Resumen:	Este caso es iniciado por un Administrador ó un Coordinador. Estos pueden generar reportes de la información contenida en la Tabla Registro Acervo de la Base de Datos.
Precondiciones:	Se requiere haber ejecutado anteriormente el caso de uso: validar Cliente.
Flujo principal:	Después de ejecutar *Ofrecer Servicios*, si se escogió Generar Reportes Acervo, se continúa con el subflujo (S–1) de este caso de uso.
Subflujos:	*S – 1 Opciones de Generar Reporte Acervo* Se presenta la pantalla P-12(a), donde el Administrador ó Coordinador puede elegir el tipo de reporte los cuales son: "Adquisiciones", "Acervo Existente", y "Salir". Si el Administrador ó Coordinador presiona "Acervo Existente" se continua con el subflujo *Acervo Existente (S–2).* Si el Administrador ó Coordinador presiona "Adquisiciones" se continua con el subflujo *Nuevas Adquisiciones (S–3).* Si la actividad seleccionada es "Salir" se continua con el caso de uso *Ofrecer Servicios.*
	S – 2 *Acervo Existente* Se presenta la pantalla P–12(b), con una lista del Acervo existente, subdivida por Idioma, Tipo de Acervo y ordenada por Clasificación, en esta pantalla el Administrador ó Coordinador puede seleccionar "Guardar", "Imprimir" y "Regresar". Si la actividad seleccionada es "Guardar" se muestra la pantalla P-12 (c), en la cual el Administrador ó Coordinador indicará el nombre, la ruta y formato del archivo que contendrá la información del reporte, esta pantalla contiene dos botones "Guardar" y "Cancelar". Si el Coordinador ó Administrador presiona "Guardar" (E–1) el archivo se guardará y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si el Coordinador ó Administrador escoge "Cancelar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si la actividad seleccionada es "Imprimir" se imprime el reporte mostrado y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si la actividad seleccionada fue "Regresar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Acervo.*

	S-3 Adquisiciones Se presenta la pantalla P–13(d), en la cual el Administrador ó Coordinador introduce un período de fechas en las cuales fueron capturadas las fichas del Acervo. En esta pantalla se tienen dos opciones: "Aceptar" y "Regresar". Si la actividad elegida es "Regresar" se ejecuta el subflujo (S-1) del caso de uso Generar Reporte Acervo. Si es "Aceptar" (E –2) (E-3) Se presenta la pantalla P–13(e), con una lista del Acervo que fue capturado en el rango de fechas indicados en la pantalla anterior, subdividida por Idioma, Tipo de Acervo y ordenada por Clasificación, en esta pantalla el Administrador o Coordinador puede seleccionar "Guardar", "Imprimir" y "Regresar". Si la actividad seleccionada es "Guardar" se muestra la pantalla P-13 (f), en la cual el Administrador ó Coordinador indicará el nombre, la ruta y formato del archivo que contendrá la información del reporte, esta pantalla contiene dos botones "Guardar" y "Cancelar". Si el Coordinador ó Administrador presiona "Guardar" (E–1) el archivo se guardará y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si escoge "Cancelar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si la actividad seleccionada es "Imprimir" se imprime el reporte mostrado y se continua con el subflujo (S-1) del caso de uso *Generar Reporte Acervo.* Si la actividad seleccionada fue "Regresar" se ejecuta el subflujo (S-1) del caso de uso *Generar Reporte Acervo.*
Excepciones:	E–1 Si no se ha puesto nombre al archivo o no se ha indicado la ruta. E – 2 Información incompleta E – 3 fecha no válida

Caso de uso:	**Búsquedas Generales Acervo**
Actores:	Coordinado, Administrador, Base de Datos
Tipo:	Básico
Propósito:	Realizar Búsquedas detallasdas de Acervo por diversos parámetros.
Resumen:	Se podrán realizar búsquedas detalladas de Acervo, en base a parámetros establecidos por el Cliente.
Precondiciones:	Se requiere haber ejecutado previamente el caos de Uso *Validar Cliente*.
Flujo principal:	Después de ejecutar *Validar Cliente* se selecciona, Administradores, y se elige del menú desplegable *Búsquedas Generales*, optando por *Acervo*. Se continua con el Subflujo (S-1) de este caso de uso.
Subflujos:	*S – 1 Opciones de Búsqueda General Acervo* Aparece la pantalla P – 13(a), en donde el Administrador puede seleccionar "Ejecutar", "Guardar" ó " Salir". Si el Administrador selecciona "Ejecutar" se continua con el subflujo *Ejecutar Búsqueda General Acervo* (S – 2). Si el Administrador selecciona "Guardar" se continua con el subflujo *Guardar Búsqueda General Acervo* (S - 3). Si la actividad seleccionada es "Salir" se continua con el caso de uso *Ofrecer Servicios.*
	S – 2 Ejecutar Búsqueda General Acervo En la pantalla P – 13(a) se presentan un menú que contine los campos del Registro de un Acervo, aquí el Cliente deberá seleccioanr los cmpos de búsqueda. En la pantalla P - 13(b) se presentan menús desplegables que le permiten al Cliente definir los criterios ó parámetros de búsqueda. Aquí se establecen los filtros de búsqueda y sus condiciones. En la pantalla P – 13(c) se elegirá el orden y el campo de referencia que deberá tener el reporte de resultados de esta búsqueda. En caso de que algún parámetro de búsqueda no este bien definido, se presenta la excepción E – 1. Si en la Base de Datos, no hay elementos que cumplan con los valores establecidos como criterios de búsqueda, se presentará la excepción E – 2. En la pantalla P - 13(d) se despliegan los resultados de la búsqueda. Se presenta la opción de "Guardar" y "Salir". Si el Cliente elige "Guardar" aparece la pantalla P – 13(e), en la cula se pide al Cliente establecer una ruta y un nombre para guardar el resultado de la búsqueda.
	S – 3 Guardar Búsqueda General Acervo En la pantalla P – 13(f) se solicita una ruta y un nombre para guardar los datos que se han registrado como criterios de búsqueda. De esta manera se puede almacenar una búsqueda definida de manera particular para futuros usos.
Excepciones:	E – 1 Información incompleta, falta seleccionar algún parámetro de búsqueda. E – 2 No Existe información que cumpla con los criterios de búsqueda definidos.

Caso de uso:	**Búsquedas Generales Usuario**
Actores:	Coordinado, Administrador, Base de Datos
Tipo:	Básico
Propósito:	Realizar Búsquedas detallasdas de Usuario por diversos parámetros.
Resumen:	Se podrán realizar búsquedas detalladas de Usuario, en base a parámetros establecidos por el Cliente.
Precondiciones:	Se requiere haber ejecutado previamente el caos de Uso *Validar Cliente*.
Flujo principal:	Después de ejecutar *Validar Cliente* se selecciona, Administradores, y se elige del menú desplegable *Búsquedas Generales*, optando por *Usuario.* Se continua con el Subflujo (S-1) de este caso de uso.
Subflujos:	*S – 1 Opciones de Búsqueda General Usuario* Aparece la pantalla P – 13(a), en donde el Administrador puede seleccionar "Ejecutar", "Guardar" ó " Salir". Si el Administrador selecciona "Ejecutar" se continua con el subflujo *Ejecutar Búsqueda General Usuario* (S – 2). Si el Administrador selecciona "Guardar" se continua con el subflujo *Guardar Búsqueda General Usuario* (S - 3). Si la actividad seleccionada es "Salir" se continua con el caso de uso *Ofrecer Servicios.*
	S – 2 Ejecutar Búsqueda General Usuario En la pantalla P – 14(a) se presentan un menú que contine los campos del Registro de un Usuario, aquí el Cliente deberá seleccioanr los cmpos de búsqueda. En la pantalla P - 14(b) se presentan menús desplegables que le permiten al Cliente definir los criterios ó parámetros de búsqueda. Aquí se establecen los filtros de búsqueda y sus condiciones. En la pantalla P – 14(c) se elegirá el orden y el campo de referencia que deberá tener el reporte de resultados de esta búsqueda. En caso de que algún parámetro de búsqueda no este bien definido, se presenta la excepción E – 1. Si en la Base de Datos, no hay elementos que cumplan con los valores establecidos como criterios de búsqueda, se presentará la excepción E – 2. En la pantalla P - 14(d) se despliegan los resultados de la búsqueda. Se presenta la opción de "Guardar" y "Salir". Si el Cliente elige "Guardar" aparece la pantalla P – 14(e), en la cula se pide al Cliente establecer una ruta y un nombre para guardar el resultado de la búsquda.
	S – 3 Guardar Búsqueda General Usuario En la pantalla P – 14(f) se solicita una ruta y un nombre para guardar los datos que se han registrado como criterios de búsqueda. De esta manera se puede almacenar una búsqueda definida de manera particular para futuros usos.
Excepciones:	E – 1 Información incompleta, falta seleccionar algún parámetro de búsqueda. E – 2 No Existe información que cumpla con los criterios de búsqueda definidos.