UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES

ARAGÓN

INGENIERÍA EN COMPUTACIÓN

"SISTEMA DE CONTROL DE TITULACIÓN CoTER"

DESARROLLO DE UN CASO PRÁCTICO

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

RAMOS MARQUEZ JUAN CARLOS

ASESORA:

MTRA. SILVIA VEGA MUYTOY

MÉXICO, 2011

Agradecimientos

A Dios:

Gracias padre celestial por haberme dado el don de la vida y darme salud además por haberme dado una familia maravillosa que me cuido y apoyo para estudiar una carrera. Gracias por guiarme en sendas rectas y cuidarme, gracias Padre celestial.

A mi papá:

Por ser un gran señor, de gran respeto, un ejemplo a seguir, por haberme inculcado la honradez, el trabajo, elser un hombre de provecho y haberme guiado en mis estudios y jalarme las orejas cuando fue necesario para no moverme de la meta, has sido un éxito rotundo de padre, gracias.

A mi mamá:

Gracias mi Santa madre, cuanto amor has tenido conmigo, nunca te lo podre pagar ni con todas las riquezas de este mundo, gracias por todo tu amor, apoyo, y compresión, tu amor me ha ayudado a salir adelante, gracias por todos tus desvelos y madrugadas por siempre estar al pendiente de mí, yo sé porque has pasado y que hermoso he vivido contigo eres una Santa, gracias.

A mis hermanos:

Gracias Beto porque me he divertido bastante junto a tu lado y a ti Wences por estar ahí.

A Brenda:

Por darme el ultimo empujoncito y aguantarme este tiempo.

ÍNDICE

INTRODUCCIÓN

En la actualidad los sistemas de información han crecido en gran número y dan soporte a cualquier área, ahora no se puede pensar por ejemplo en hacer trámites gubernamentales, compras en el supermercado, transacciones bancarias sin el uso de una computadora. Además las cantidades de información han crecido a pasos agigantados, por lo que es más difícil administrarlos, distribuirlos, respaldarlos y explotarlos. Pero gracias a toda la tecnología subyacente de estos sistemas de información nos han ayudado a hacer el trabajo menos complejo y más eficiente.

El sistema de información CoTER[1] fue desarrollado en base a las necesidades actuales en el área de titulación, ya que ayudara a tener un mejor control sobre los egresados o alumnos que estén en dicho proceso.

El presente trabajo cubre las áreas de desarrollo de software es decir su ciclo de vida, abarcando la notación gráfica UML, el uso de patrones de diseño así como los estándares para creación de páginas Web todo esto con ayuda de herramientas que facilitaran su análisis, desarrollo e implantación y pruebas como lo son IDE's y frameworks de desarrollo.

En la etapa de análisis y diseño se verán todas las necesidades del sistema así como presentar un bosquejo de solución, con ayuda de UML se presentara este bosquejo y se verá si el sistema satisface estas necesidades además si es amigable al usuario.

Durante la etapa de desarrollo e implementación se procederá a crear el sistema y si lo requiere hacer cambios en el análisis, además de descartar lo que no sea posible computacionalmente. El lenguaje de programación utilizado es Java que junto con el entorno de desarrollo integrado Netbeans ayuda para agilizar la construcción de los distintos componentes. El sistema gestor de BD usado es MySQL, además de que el sistema exige el uso de estándares apegados a la Web como XHTML, CSS, JavaScript, Ajax, y el uso de MVC para crear sitios flexibles y bien estructurados. También se va a hacer uso de la herramienta Ireport para generar los correspondientes reportes en formato PDF.

La etapa de pruebas y mantenimiento sirven para ver si el sistema cumple los requisitos especificados y si lo hace de la mejor manera, también se deben hacer varias pruebas para ver si el sistema no emite algún error con un tipo de información no común. Si hubiera algún desperfecto se tendría que hacer el mantenimiento necesario del sistema.

El sistema beneficiara a los profesores ya que tendrán un control central sobre sus asesorados o revisores, además de mostrarles información específica sobre cuáles son las tendencias de los alumnos sobre la opción de titulación.

[1] CoTER. Acrónimo de Control de Titulados, Egresados y Revisores.

CAPÍTULO 1

ANÁLISIS Y DISEÑO.

En esta etapa se especifican las necesidades del sistema a desarrollarse. La especificación de requisitos puede servir como base para la negociación entre los desarrolladores y clientes del sistema, es esencial que los clientes que no tengan un conocimiento de la computación puedan comprender el modelo para facilitar la interacción con ellos. Se busca comprender los requisitos del sistema logrando con esto la estructuración de una solución correspondiente a la arquitectura general además se debe especificar el alcance del mismo.

1.1 El Proceso para el Desarrollo de Software

Un proceso está definido como una serie de acciones u operaciones que conducen a un fin. En general, una empresa u organización requiere de uno o más procesos para lograr sus objetivos, los cuales por lo general involucran la utilización de sistemas de software. En el caso de una empresa que se dedica al 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. ¡De lo contrario todos haríamos siempre software perfecto! Un aspecto básico para manejar la complejidad inherente en los sistemas de software es contar con un modelo de proceso a seguir.

1.1.1 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.

Los modelos de proceso varían mucho entre sí y dependen de las diversas opiniones o máximas generales en el desarrollo de software. Por ejemplo algunas creencias en el desarrollo de software son:
· Es mejor comprender el problema antes de desarrollar una solución.
· El proceso para resolver un problema debe dar un resultado predecible, sin importar del individuo que hace el trabajo.
· Debe ser posible planear y calcular el proceso con gran precisión.
· Evaluar y administrar el riesgo es importante para el éxito del proceso.
· Etapas bien definidas con entregas intermedias aumentan la confianza que se tiene en el resultado final.

En general, todas las creencias luego actúan como base para definir las estrategias, actividades, métodos, y tareas del modelo de proceso.
Estos conceptos se describen a continuación.

· Una estrategia es un plan para llevar a cabo un objetivo, en este 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, por ejemplo, utilizando elementos sencillos como bloques y componentes o como elementos prefabricados de más alto nivel. Esta arquitectura puede incluso integrar diversos niveles de sofisticación en los elementos. Las estrategias básicas escogidas afectan directamente el tipo de programación y los lenguajes que se utilizarán. Para este proyecto la estrategia básica de desarrollo de software es el uso de tecnología orientada a objetos, en particular usando el lenguaje Java.

Sin embargo, esta estrategia de orientación a objetos puede refinarse aún más. (Obviamente, se puede utilizar una estrategia distinta, incluso que no sea orientada a objetos.) La estrategia no sólo afecta la arquitectura del sistema sino también cómo se llevarán a cabo las actividades del proceso. Mientras no

se tengan conflictos, es posible combinar múltiples estrategias, donde las distintas actividades del proceso de software pueden hacerse bajo estrategias diferentes, definiendo implícitamente la estrategia global del modelo de proceso.

· Una actividad es una unidad o paso organizacional para llevar a cabo cierto aspecto de un proceso. En este caso las actividades definen 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 simples de aprender y usar; deben simplificar la comprensión del sistema, deben ser suficientemente poderosas para expresar la información requerida para modelar el sistema, deben ser lo suficientemente descriptivas para poder discutir el sistema sin ambigüedades y deben proveer un modelo evolucionable del sistema. Las actividades básicas necesarias para el proceso de desarrollo de software son las siguientes: (1) análisis para capturar los aspectos funcionales correspondientes, cómo un usuario interactuaría con el sistema; además para dar al sistema una estructura robusta y extensible bajo un ambiente de implementación ideal; (2) diseño para adoptar y refinar las estructuras al ambiente de implementación particular; (3) implementación para codificar el sistema; (4) pruebas para validar y verificar el sistema; (5) mantenimiento para extender la funcionalidad del sistema y (6) documentación para describir los distintos aspectos el sistema.

· Un método es un procedimiento que define las tareas que deben llevarse a cabo para satisfacer la actividad. Existen métodos, por ejemplo, para asegurar la calidad del software, seguir el progreso del proyecto y probar el software. Durante el análisis, el método debe ayudar en la identificación de los objetos necesarios para la arquitectura del sistema. Análisis estructurado y análisis orientado a objetos son ejemplos de diferentes métodos para hacer análisis, cada uno con sus propias tareas. Una metodología se refiere al estudio de los métodos, existiendo un gran número de metodologías para el desarrollo de software. En general, distintas metodologías llevan a cabo las actividades del desarrollo de software de diferente manera. En este proyecto se aplica las Metodologías más evolucionadas utilizando tecnología orientada a objetos.

· Una tarea es un grupo relacionado de acciones contribuyendo a una acción mayor. Cada método define un conjunto de tareas a llevarse a cabo para lograr los objetivos deseados. La tarea puede incluir condiciones de entrada y de salida que serán satisfechas antes y después de su realización.

1.1.2 Modelo Cascada

El modelo de cascada clásico data de la década de los 60s y 70s. El modelo 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 (“milestones” o “checkpoints”). En su época de esplendor, este modelo tuvo gran aceptación en la comunidad de contratistas gubernamentales estadounidenses, ya que éstos recibían sus pagos del gobierno en base a entregas basadas en horarios (“schedule”) predefinidos. El desarrollo de software implicaba una secuencia de actividades

a realizarse y cuyo seguimiento era verificar que cada actividad haya sido completada. La ejecución del modelo era muy lineal, por lo cual el modelo fue sencillo y atractivo; donde se especificaba las actividades para luego hacerlas de principio a fin. Se consideraba que una vez terminada una actividad se continuaba con la siguiente. La Figura 1.1 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.

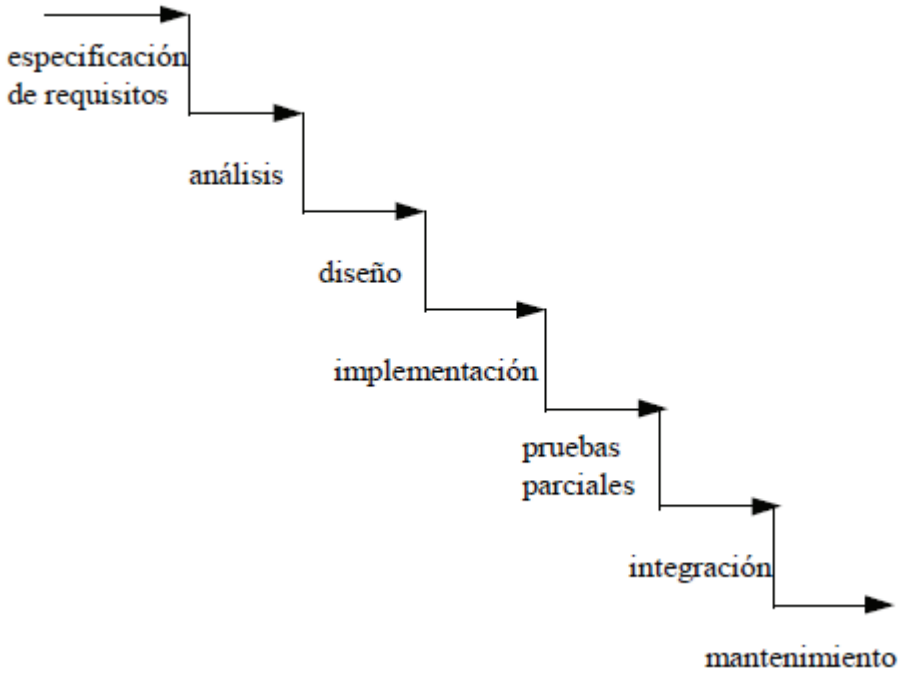

Figura 1.1 Diagrama del modelo de cascada

Las siguientes máximas sirven de base para el Modelo de Cascada (Goldberg y Rubin 1995):

· Las metas se logran de mejor manera teniendo como fin puntos de revisión bien definidas y documentadas, dividiendo el desarrollo en etapas secuenciales bien definidas.

· Documentos técnicos son comprensibles para usuarios y administradores no-técnicos, y estos participantes no técnicos pueden comunicarse de forma efectiva durante las diversas actividades.

· Cada detalle sobre los requisitos puede conocerse de antemano antes de desarrollarse el software, y estos detalles son estables a través del desarrollo.

· Pruebas y evaluaciones pueden llevarse a cabo eficientemente al final del desarrollo.

El modelo de cascada fue inicialmente bien recibido ya que identificaba etapas razonables y lógicas para las diversas actividades. Lamentablemente, el modelo no explicaba entre otras cosas cómo modificar un resultado. No existía una guía del por qué y cuándo se debía revisar un resultado previo para sus posibles cambios, en especial considerando que es extremadamente difícil definir todos los requisitos de un sistema al inicio y que éstos se mantengan

estables y sin cambios a lo largo del desarrollo. Esta rigidez trajo dudas sobre la utilidad del modelo. La ironía en la mayoría de los proyectos de desarrollo que usan este modelo es que los administradores no están de acuerdo con las máximas básicas, aunque eligen modelos de proceso basados en ellas. A menudo, el requisito de producir entregas intermedias (mayormente documentos) para ser seguidos por financiamiento obliga a seguir este enfoque secuencial, separando drásticamente las actividades, aun cuando los administradores crean que otro enfoque sería mejor. Por lo tanto, el modelo de cascada dejó de ser utilizado de acuerdo a su definición original, llevando a los usuarios a utilizar variantes del modelo básico.

Ed Yourdon, en su libro “Decline and Fall of the American Programmer” (Yourdon 1992), 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 rehúso de software.
· No promueve el uso de prototipos.
· No se practica de manera formal.

1.1.3 Modelo Espiral

El modelo de espiral es una modificación al modelo de cascada desarrollado durante la década de los 80s. El modelo de espiral se basa en una estrategia para reducir riesgo, al contrario del modelo de cascada que es dirigido por documentos. Como parte del manejo de riesgo el modelo incorpora una estrategia de uso de prototipos algo muy aceptado en la actualidad. 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.

Para utilizar este modelo se debe ser particularmente bueno en identificar y manejar riesgos. La Figura1.2 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.

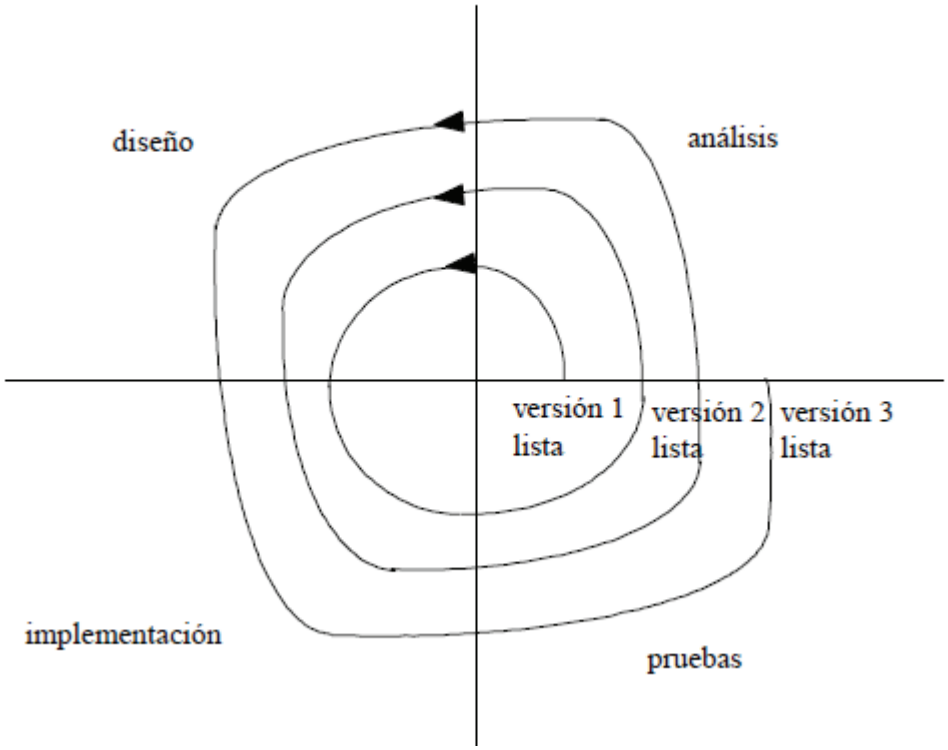

Figura 1.2 Diagrama del modelo en espiral

El modelo de espiral contempla que el desarrollo de sistemas es un proceso de cambios progresivos. Un sistema normalmente se desarrolla mediante cambios en la especificación de la versión anterior del sistema que son incorporados a nuevas versiones, donde un cambio se conoce como un delta en la especificación de requisitos o versión. En la Figura 1.3 se ilustra este concepto.

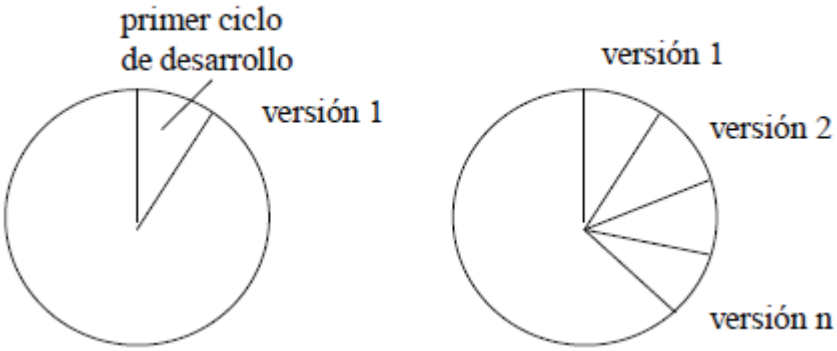

Figura 1.3 Secuencia de versiones en el modelo de espiral.

Aunque modificaciones a sistemas existentes constituyen la mayor parte del costo total durante el ciclo de vida del software, la mayoría de los métodos de desarrollo de software se concentran en nuevos desarrollos, tratando revisiones como algo menor. El modelo de proceso debiera enfocarse en los cambios del sistema.

Las máximas del modelo de espiral son:
· Una actividad comienza con un entendimiento de los objetivos y riesgos involucrados.

· Basado en la evaluación de soluciones alternas, se usa las herramientas que mejor reduzcan los riesgos.
· Todo el personal relacionado debe involucrarse en una revisión que determina cada actividad, planeando y comprometiéndose a las siguientes actividades.
· El desarrollo puede proceder en incrementos en cada etapa, permitiendo prototipos sucesivos del producto.
Con algunas variantes, este es el modelo de proceso más importante en la actualidad.

1.2 Descripción del problema y requerimientos de usuario.

Actualmente en el proceso de titulación se cuenta con el sistema tradicional esto es: el universitario egresado que quiere empezar el proceso de titulación lo primero que tiene que hacer es escoger a su asesor quien lo guiará a través de este proceso, una vez que ambos se ponen de acuerdo en la modalidad de titulación y el bosquejo del desarrollo de esa modalidad empiezan los respectivos trámites con los formatos establecidos , ante esto el usuario tiene que descargar de la página de la facultad dichos formatos en los cuales ingresa su información personal como nombre, domicilio, etc., así como información referente a su titulación que puede ser por ejemplo el tipo de modalidad, el nombre del tema, el asesor, etc., todos estos datos se van propagando a lo largo de los distintos formatos dentro de los cuales se va repitiendo la información, unos llevando menos datos que otros pero finalmente todos hacen uso de esa misma información, finalmente el asesor va revisando y firmando los distintos formatos en el transcurso del desarrollo de la titulación, pudiendo haber errores u omisiones de un documento a otro durante el proceso.

Viendo este panorama es claro observar que el proceso de llenado de los diferentes formatos es algo tedioso por el simple hecho de sólo estar capturando la misma información en los documentos. Para ello se puede tener un sistema de información en el cual se capture una sola vez toda esta información y la guarde de manera permanente, brindando con esto grandes opciones, dentro de las cuales es llenar el contenido de los diferentes formatos de manera automática, además se puede explotar la información guardada de cada egresado, como por ejemplo saber cuántos alumnos asesoro un profesor, cuantos alumnos se titularon en un semestre, que modalidad es la más demandada, en fin se puede tener bastante información útil para poder analizar el comportamiento de los alumnos y las modalidades escogidas para de esta forma llevar a cabo acciones a favor de los alumnos egresados así como de las modalidades más complejas y más tardadas y si éstas se pueden modificar, finalmente este sistema también puede llevar a cabo un mejor control de los profesores sobre sus asesorados ya que el catedrático tiene que llevar un control de a cuantos estudiantes atiende, a qué hora ve a cada uno, en qué fase va cada estudiante, si se han presentado o por causas ajenas ya no han ido a las asesorías, además de que el profesor al entregar sus reportes de informe anual debe hacer mención de esto, y el sistema le puede ayudar en estas estadísticas.

El sistema CoTER es una herramienta que permite al profesor llevar un control sobre los alumnos asesorados o revisados, así como para que los alumnos ingresen sus datos sólo una vez y poder descargar los distintos formatos con las cartas ya llenadas además de ver como va su proceso.

El sistema proporciona en su pantalla inicial un cuadro donde el usuario (profesor, alumno y administrador) pueden ingresar su nombre de usuario y su contraseña para entrar al sistema. El administrador del sistema ya tiene asignado su usuario y contraseña. Para que un profesor haga uso del sistema se le debe asignar un nombre de usuario y una contraseña por el administrador del sistema. Para que un alumno haga uso del sistema hay una opción junto al cuadro de inicio de sesión para registrarse por primera vez, el alumno previamente ya debió haber platicado con el profesor para que éste tenga en cuenta el nuevo registro.

Una vez autenticado el usuario, se tendrán distintas pantallas o acciones dependiendo del tipo de usuario ya sea alumno, profesor o administrador.

Un Administrador tendrá las siguientes acciones:

- Dar de alta, editar y borrar a los profesores.
- Modificar los datos del sistema.
- Dar de alta un nuevo ciclo escolar.

Un profesor tendrá las siguientes acciones:

- Consultar a sus asesorados.
- Revisar estadísticas.

Un alumno tendrá las siguientes acciones:

- Editar datos.
- Descargar sus cartas llenadas.

Como se vio anteriormente para que un profesor haga uso del sistema el administrador de la aplicación lo debe dar de alta, por lo que el administrador al validarse en la pantalla inicial tendrá las opciones de *Administrar profesores* esto es registrar un profesor en el sistema ingresando sus datos y asignándole, un nombre de usuario y una contraseña. También tendrá la opción de *Editar sistema* con lo cual podrá modificar los datos de los formatos de los alumnos. Finalmente la opción *Añadir ciclo escolar* le permitirá dar de alta un nuevo semestre.

Un profesor al ingresar al sistema tendrá sus respectivas opciones. La opción *Revisar alumnos* le permite ver a todos los alumnos que está asesorando y revisar uno por uno además de checar en qué fase de su proceso van, así como ver los documentos del proceso. La opción *Estadísticas* le permite ver en

forma gráfica el número de alumnos que asesora, que opción es las más demandada, etc.

Un alumno antes de poder ingresar al sistema se debe dar de alta. Para esto en la página principal hay un link para registrarse, posteriormente en esta página el usuario ingresara todos sus datos, con esto el alumno tendrá su usuario y contraseña. Cuando ingrese al sistema el alumno tendrá 2 opciones. La opción de *Editar datos* en la que podrá cambiar sus datos personales o datos relativos a su titulación por si tuvo alguna equivocación al inicio o se modificaron posteriormente. La opción *Formatos* le permite descargar sus cartas así como subir sus avances.

1.3 Lenguaje Unificado de Modelado

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje grafico para visualizar, especificar, construir y documentar un sistema de software. Esta herramienta es de las más emocionantes en el mundo actual del desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas. Por tanto UML permite organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él.

UML es la creación de Grady Booch, Ivar Jacobson y James Rumbaugh y actualmente lo mantiene el grupo de administración de objetos (OMG).

El UML está compuesto por diversos componentes gráficos que se combinan para formar diagramas. Debido a que UML es un lenguaje cuenta con reglas para combinar tales elementos. La finalidad de los diagramas es presentar diversas perspectivas de un sistema a las cuales se les conoce como modelo. En UML 2.0 hay 13 tipos diferentes de diagramas. Para comprenderlos de manera concreta a veces es útil categorizarlos jerárquicamente, como se muestra en la figura 1.4.

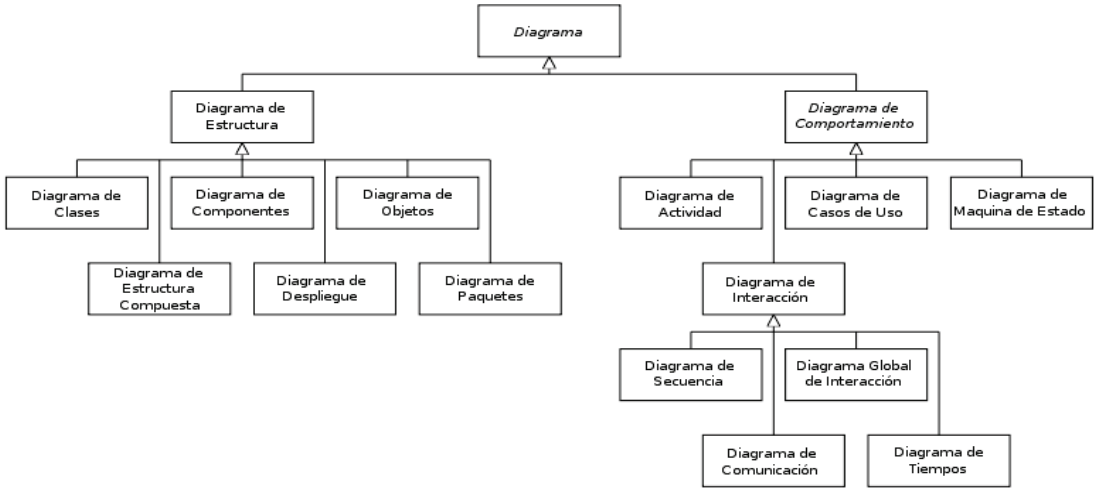

Figura. 1.4 Diagramas de UML 2.0

Diagramas de estructura: Enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases.
- Diagrama de componentes.
- Diagrama de objetos.
- Diagrama de estructura compuesta (UML 2.0).
- Diagrama de despliegue
- Diagrama de paquetes

Diagramas de comportamiento: Enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagramas de casos de uso
- Diagrama de estados

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

- Diagrama de secuencia
- Diagrama de comunicación
- Diagrama de tiempos (UML 2.0)
- Diagrama de vista de interacción (UML 2.0)

1.4 Casos de uso

El modelo de los casos de uso describe un sistema en términos de sus distintas formas de utilización, cada una de estas formas es conocida como 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 significaran cambios en los casos de uso. Lo importante en los casos de uso es saber cuáles son los requerimientos, los casos de uso son una forma de ver como el usuario interactuara con el sistema, con una colección de casos de uso se puede hacer el bosquejo de un sistema en términos de lo que los usuarios intenten hacer con él.

1.4.1 Representación de un modelo de casos de uso

Actores:

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

Caso de uso:

Es una forma de utilización del sistema, son las distintas acciones que un usuario puede hacer en un sistema. Cada caso de uso establece un conjunto de escenarios para realizar algo útil para un actor.

Al definir todos los actores y casos de uso en el sistema se define la funcionalidad completa del sistema.

Hay un actor que inicia un caso de uso y otro (posiblemente el que inicio, pero no necesariamente) que recibirá algo de valor de él. La representación gráfica es directa. Una elipse representa a un caso de uso, una figura agregada representa un actor. El actor que inicia se representa a la izquierda del caso de uso, y el que recibe a la derecha. El nombre del actor aparece justo debajo de él, y el nombre del caso de uso aparece ya sea dentro de la elipse o justo debajo de ella. Una línea asociativa conecta a un actor con el caso de uso, y representa la comunicación entre el actor y el caso de uso. La figura 1.5 ilustra un modelo de casos de uso.

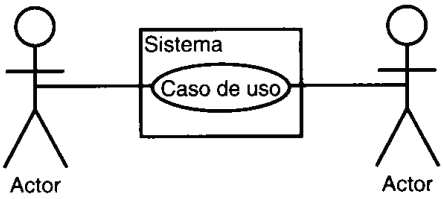

Figura 1.5 Actores y casos de uso

1.4.2 Relaciones entre los casos de uso y los actores

Para unir los casos de uso entre si además de unirlos con los actores se hace mediante los distintos tipos de relaciones. Hay 4 tipos de relaciones:

- Inclusión: Le permite a un caso de uso utilizar los pasos de otro caso de uso.
- Extensión: permite crear un nuevo caso de uso mediante la adicción de unos pasos a un caso de uso existente.
- Generalización: Se puede crear un caso de uso nuevo a partir de un caso de uso base. El caso de uso secundario hereda las acciones y el significado del primario y además agrega sus propias acciones. La relación de generalización puede establecerse entre actores, así como entre casos de uso.
- Asociación: Relación entre un actor y un caso de uso, denota la participación del actor en el caso de uso determinado.

Además dentro de la terminología de los casos de uso se incluye el término flujo básico, éste corresponde a la secuencia de eventos más común o típica del caso de uso. Posteriormente también se encuentran las variantes o flujos alternos que incluye flujos para el manejo de errores que por lo general corresponden a situaciones menos comunes. Normalmente un caso de uso sólo tiene un flujo básico pero varios alternos.

1.5 Identificación de los actores y los casos de uso para el sistema CoTER

Actores:

Los actores que se han identificado en base a quién va a hacer uso del sistema CoTER son los mostrados en la figura 1.6.

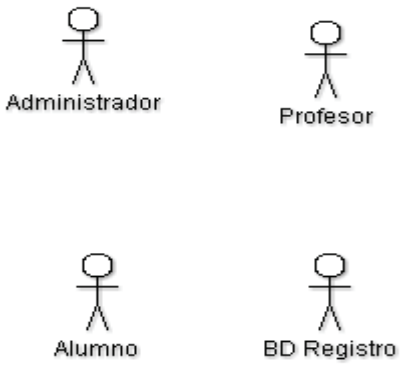

Figura 1.6 Actores del sistema CoTER

Los actores Usuario y Profesor representan al asesor y al asesorado respectivamente, el actor BD Registro representa a la BD que contiene toda la información del alumno así como del profesor, como esta BD tiene un proceso interactivo y externo al sistema se modela como un actor mas. Finalmente el actor Administrador es aquel quien puede dar de alta a un profesor en el sistema.

Casos de uso:

Los casos de uso o formas de utilización del sistema CoTER que se han identificado en esta etapa de análisis son los mostrados en la figura 1.7.

Figura 1.7 Casos de uso del sistema CoTER

Con el avance del proyecto se podrían identificar otros casos de uso e irse agregando al sistema.

El modelo de casos de uso con los actores y casos de uso del sistema es el mostrado en la figura 1.8.

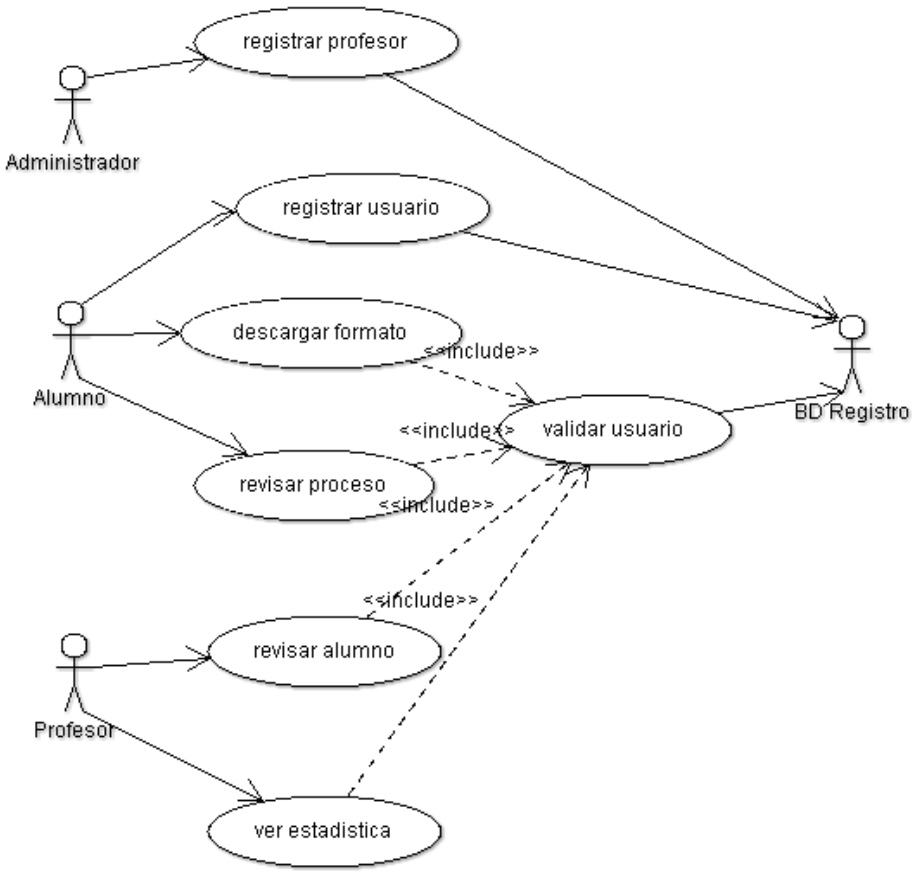

Figura 1.8 Modelo de casos de uso del sistema CoTER

1.11 Descripción de los casos de uso del sistema

Caso de uso:	Registrar profesor
Actores:	Administrador, BD Registro
Propósito:	Permite dar de alta un profesor en el sistema para su uso posterior.
Resumen:	Este caso de uso es iniciado por el administrador del sistema. Ofrece funcionalidad para crear, modificar y eliminar el registro de un profesor con el sistema.
Precondiciones:	El administrador del sistema se debió haber autenticado.

Flujo principal:	El administrador al ingresar al sistema con sus credenciales accede a la pantalla principal en la que tiene las siguientes opciones: “administrar profesor”, “editar sistema”, “Añadir ciclo escolar”. Si la actividad seleccionada es “administrar profesor” se presenta una pantalla donde se muestran todos los profesores del sistema en una lista editable si se quiere guardar se da clic en el botón guardar o si se quiere borrar en la imagen de la papelera de reciclaje. Además tiene la opción de añadir nuevo profesor en la que se presenta una nueva pantalla en donde el administrador tiene que introducir los datos personales del profesor, posteriormente se le tiene que asignar un usuario y una contraseña para que con estos datos pueda ingresar al sistema. Posteriormente el administrador puede dar un clic en el botón aceptar con lo que se guardan los datos del profesor y se redirige a la pantalla anterior. Si da un clic en el botón cancelar cancelar no guarda ninguna información y lo redirige a la pantalla anterior.

Caso de uso:	Registrar usuario
Actores:	Usuario, BD Registro
Propósito:	Permite dar de alta un alumno en el sistema para su uso posterior.
Resumen:	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad de añadir un usuario al sistema.
Precondiciones:	Ninguna.
Flujo principal:	El usuario al ingresar al sistema en la parte izquierda tiene la opción de registrarse. Cuando seleccione esta opción le saldrá una ventana en donde tendrá que ingresar todos sus datos personales como académicos, después si han sido llenados todos los datos necesarios y tienen un valor correcto tiene la opción de guardar con lo que el usuario se habrá dado de alta en el sistema y se redirigirá a una ventana donde le indicara que fue exitoso el registro, si un dato estuviere mal o no fuere llenado le saldrá un mensaje para que corrija los datos.

Caso de uso:	Descargar formato
Actores:	Usuario, BD Registro.
Propósito:	Permite ver una carta del alumno y posteriormente descargarla.
Resumen:	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad para que el usuario descargue sus formatos.

Precondiciones:	El usuario del sistema se debió haber autenticado.
Flujo principal:	El usuario al ingresar al sistema con sus credenciales accede a la pantalla principal en la que tiene las siguientes opciones: “editar datos“, “formatos”. Si la actividad seleccionada es “formatos” se presenta una pantalla donde se muestran todas las cartas que puede descargar así como los documentos que ha subido. Cuando da clic en alguna de las cartas el navegador le muestra la carta en formato PDF para que posteriormente el usuario la descargue.

Caso de uso:	Revisar alumno.
Actores:	Profesor, BD Registro
Propósito:	Permite ver un alumno de los que tiene asignados el profesor.
Resumen:	Este caso de uso es iniciado por el profesor. Ofrece funcionalidad para ver un alumno en particular así como sus cartas y documentos.
Precondiciones:	El profesor del sistema se debió haber autenticado.
Flujo principal:	El profesor al ingresar al sistema con sus credenciales accede a la pantalla principal en la que tiene las siguientes opciones: “Revisar alumnos”, “Estadisticas”. Si la actividad seleccionada es “Revisar alumnos” se presenta una pantalla donde se muestran todos los alumnos que tiene asignados en una lista, adelante del nombre del alumno se tiene el enlace para accesar a alumno donde se presentara una pantalla con los datos del alumno y sus cartas y documentos.

Caso de uso:	Ver estadísticas
Actores:	Profesor, BD Registro
Propósito:	Permite al profesor ver información totalizada de sus alumnos.
Resumen:	Este caso de uso es iniciado por el profesor. Ofrece funcionalidad para que un profesor revise cuántos alumnos tiene, que opción de titulación es la más demandada, así como información pertinente para sus reportes.
Precondiciones:	El profesor se debió haber autenticado.

Flujo principal:	El profesor al ingresar al sistema con sus credenciales accede a la pantalla principal en la que tiene las siguientes opciones: “Revisar alumnos”, “Estadísticas”. Si la actividad seleccionada es “estadísticas” se presenta una pantalla donde se muestran las gráficas con el número de alumnos por semestre y opción tanto si son asesorados o revisados.

Caso de uso:	Validar usuario
Actores:	Profesor, Usuario, Administrador, BD Registro
Propósito:	Permite al usuario validarse en el sistema.
Resumen:	Este caso de uso es iniciado por el usuario. Ofrece funcionalidad para que un usuario se valide en el sistema y pueda accesar a él.
Precondiciones:	Ninguna.
Flujo principal:	El usuario al estar en la página principal debe introducir tanto su clave de usuario como su contraseña para posteriormente dar un clic en el botón iniciar sesión donde si sus datos son correctos ingresará a la página principal, de lo contrario le volverá a solicitar sus credenciales.

1.6 Análisis de la persistencia de la información

1.6.1 Modelo Entidad – Relación

Modelo Entidad - Relación: Un diagrama o modelo entidad-relación es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes, sus relaciones y propiedades.

El Modelo Entidad - Relación es un concepto de modelado para bases de datos, propuesto por Peter Chen en 1976, mediante el cual se pretende 'visualizar' los objetos que pertenecen a la Base de Datos como entidades las cuales tienen unos atributos y se vinculan mediante *relaciones*.

El modelo es una representación conceptual de la información. Mediante una serie de procedimientos se puede pasar del modelo E-R a otros, como por ejemplo el modelo relacional el cual se verá en el apartado de diseño de este trabajo.

El modelado entidad-relación es una *técnica* para el modelado de datos utilizando diagramas entidad relación. No es la única técnica pero sí la más utilizada. Brevemente consiste en los siguientes pasos:

- Se parte de una descripción textual del problema o sistema de información a automatizar (los requisitos).

- Se hace una lista de los sustantivos y verbos que aparecen.
- Los sustantivos son posibles entidades o atributos.
- Los verbos son posibles relaciones.
- Analizando las frases se determina la cardinalidad de las relaciones y otros detalles.
- Se elabora el diagrama (o diagramas) entidad-relación.
- Se completa el modelo con listas de atributos y una descripción de otras restricciones que no se pueden reflejar en el diagrama.

Dado lo rudimentario de esta técnica se necesita cierto entrenamiento y experiencia para lograr buenos modelos de datos.
El modelado de datos no acaba con el uso de esta técnica. Son necesarias otras técnicas para lograr un modelo directamente implementable en una base de datos.

1.6.2 Base Teórica y Conceptual

El modelo Entidad - Relación se basa en los conceptos descritos a continuación para representar un modelo de la vida real.

Entidad

Representa una “cosa” u "objeto" del mundo real con existencia independiente, es decir, se diferencia únicamente de cualquier otro objeto o cosa, incluso siendo del mismo tipo.

Ejemplos:
- Una persona. (Se diferencia de cualquier otra persona, incluso siendo gemelos).
- Un automóvil. (Aunque sean de la misma marca, el mismo modelo,..., tendrán atributos diferentes, por ejemplo, el número de bastidor).
- Una casa (Aunque sea exactamente igual a otra, aún se diferenciará en su dirección).

Una entidad puede ser un objeto con existencia física como: una persona, un animal, una casa, etc. (entidad concreta), o un objeto con existencia conceptual como: un puesto de trabajo, una asignatura de clases, un nombre, etc. (entidad abstracta).

Una entidad está descrita y se representa por sus características o atributos. Por ejemplo, la entidad Persona puede llevar consigo las características: Nombre, Apellido, Género, Estatura, Peso, Fecha de nacimiento, etc...

Atributos

Los atributos son las propiedades que describen a cada entidad en un conjunto de entidades.

Como ejemplo de la entidad persona ésta tiene como atributos su estatura, peso, color de piel.

Relación

Describe cierta dependencia entre entidades o permite la asociación de las mismas.

Como ejemplo de relación de las entidades antes vistas tenemos que una persona tiene una casa y una casa tiene una o varias personas, aquí vemos como se asocia la entidad persona con la entidad casa.

1.6.3 Requerimientos de la BD

El modelo de datos del sistema CoTER viendo los requisitos de la información que se necesita tener almacenada y de la información que no está de manera directa pero que aún así se necesita para las reglas de negocio es mostrado en el modelo Entidad – Relación ilustrado en la figura 1.9.

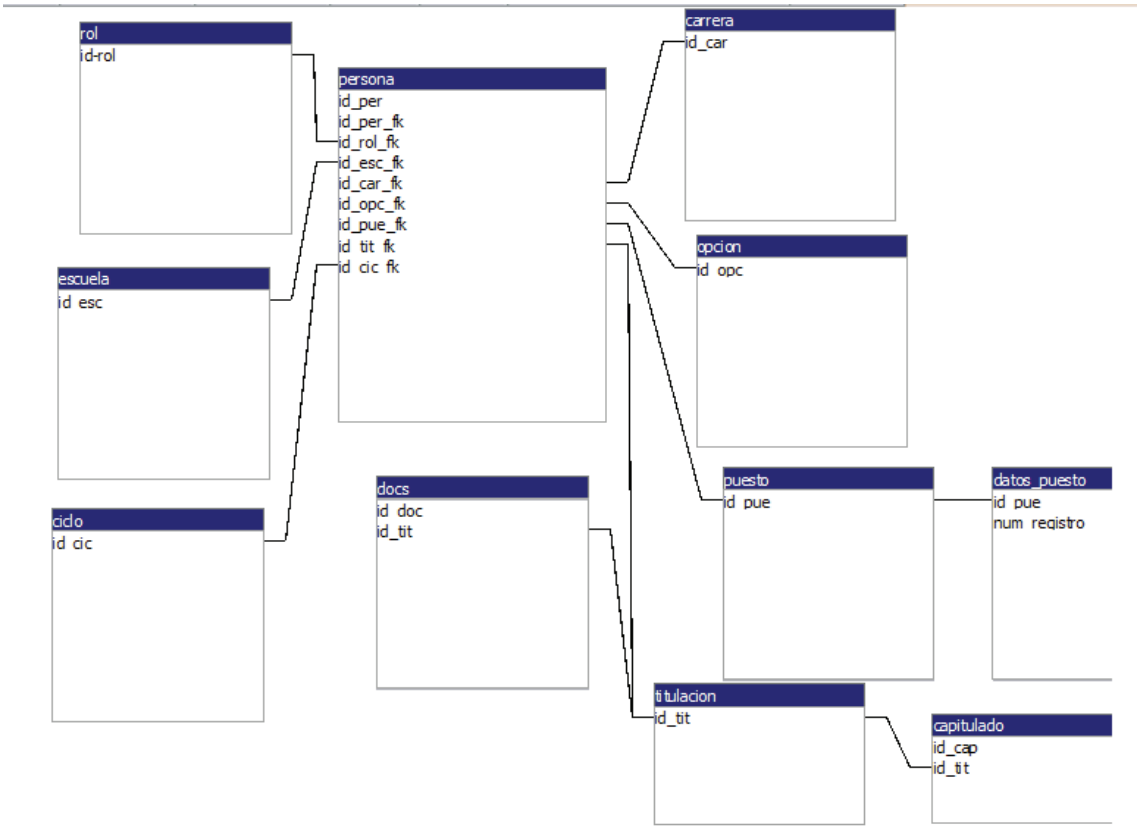

Figura 1.9 Modelo Entidad - Relación

1.6.3.1 Justificación de las entidades y relaciones

La entidad puesto surge de que un alumno puede tener algún trabajo, pero este trabajo es genérico ya que puede existir un tipo de trabajo como por ejemplo “Programador Java” que este en varias empresas por lo que los datos como lugar de trabajo, nombre de la empresa, etc. son particulares, esto da como resultado otra entidad derivada nombrada datos_puesto que viene siendo las

particularidades de un puesto de trabajo genérico, es decir un registro de la tabla puesto puede tener uno o más registros de la tabla datos_puesto. La entidad carrera representa a las carreras que se imparten en la FES Aragón. La entidad opción representa a las opciones de titulación. La entidad rol representa a los roles que puede tener un usuario del sistema. La entidad escuela representa los datos de la escuela como lo son: director, jefe de secretaria, etc. La entidad titulación representa los valores de un alumno cuando se va a titular como su fecha de registro, fecha de examen, etc., esta entidad tiene relación con otras dos entidades derivadas, la primera es docs la cual representa los informes que el usuario tiene, la relación es que un registro de la tabla titulación tiene uno o varios documentos, la segunda entidad derivada es capitulado que representa a todos los capítulos de un trabajo de titulación la relación aquí es que un registro de la tabla titulación tiene uno o más capítulos. La entidad ciclo representa a los semestres. La entidad persona es usada para los distintos usuarios del sistema por ende debe contener en sus atributos las características genéricas de estos usuarios. La entidad persona se relaciona con la entidad rol, escuela, ciclo, carrera, opción, puesto y titulación ya que un alumno debe tener todos estos datos.

1.7 Delimitación del sistema

El sistema CoTER cumple con las especificaciones de registrar usuarios y generar sus cartas correspondientes además de obtener estadisticas de los usuarios para cada profesor.

El sistema esta diseñado considerando que exista un usuario por opción de titulación, en una versión posterior se podrían hacen las modificaciones necesarias para implementar la opción de dos o más usuarios por titulación, además de incluir otras opciones.

1.8 Conclusiones del capítulo

En este primer capítulo se puede observar la importancia de conocer y comprender ampliamente las necesidades del usuario, para presentarle un esbozo del sistema y si esté cumple con lo requerido por el usuario.

Con la notación de UML se pude presentar el sistema en términos legibles para que el cliente lo entienda así como un modelo de construcción del sistema para los desarrolladores.

CAPÍTULO 2

DESARROLLO E IMPLEMENTACIÓN

En este capítulo se crea la estructura del proyecto con sus reglas de negocio establecidas en el capítulo anterior, además en esta etapa se especifican las arquitecturas, interfaces y componentes de software para implementar el sistema en un lenguaje de programación. También se debe tener en cuenta el contexto del ambiente de trabajo subyacente para que la implementación sea óptima.

2.1 Arquitectura del sistema

La arquitectura del sistema CoTER se va a realizar en lo que se conoce como la arquitectura de 3 capas, ya que al implementar un sistema de esta forma es más fácil mantenerlo además de ser extensible fácilmente. Esta arquitectura está presente en el patrón de diseño MVC.

2.1.1 Modelo Vista Controlador

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El estilo MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista. La figura 2.1 ilustra el patrón MVC.

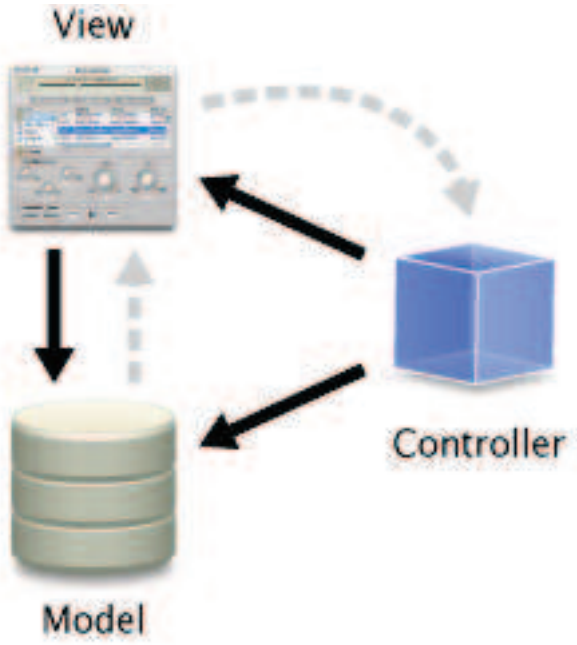

Figura 2.1 Patrón MVC

Historia

El estilo fue descrito por primera vez en 1979 por Trygve Reenskaug, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox. La implementación original está descrita Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador.

Descripción del patrón

- Modelo: Esta es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos: en líneas generales del MVC corresponde al modelo. La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre Vista y su correspondiente Controlador de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta información entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aún así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista

para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Ventajas de utilizar MVC

Obviamente una separación total entre lógica de negocio y presentación. A esto se le pueden aplicar opciones como el multilenguaje, distintos diseños de presentación, etc. sin alterar la lógica de negocio. La separación de capas como presentación, lógica de negocio, acceso a datos es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y más fácilmente mantenibles, lo que al final resulta en un ahorro de tiempo en desarrollo en posteriores proyectos.

Al existir la separación de vistas, controladores y modelos es más sencillo realizar labores de mejora como:

- Agregar nuevas vistas.
- Agregar nuevas formas de recolectar las órdenes del usuario (interpretar sus modelos mentales).
- Modificar los objetos de negocios bien sea para mejorar el performance o para migrar a otra tecnología.
- Las labores de mantenimiento también se simplifican y se reduce el tiempo necesario para ellas. Las correcciones sólo se deben hacer en un solo lugar y no en varios como sucedería si se tuviese una mezcla de presentación e implementación de la lógica del negocio.
- Las vistas también son susceptibles de modificación sin necesidad de provocar que todo el sistema se paralice. Adicionalmente el patrón MVC tiende a la especialización de cada rol del equipo, por tanto en cada liberación de una nueva versión se verán los resultados.

2.1.2 Frameworks

Un framework es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

También los frameworks pueden ser vistos como implementaciones de patrones de diseño que facilitan la reutilización de diseño y código.
Dado que MVC ha sido utilizado en muchas aplicaciones web, el desarrollo de frameworks que den soporte a áreas comunes en todas las aplicaciones MVC es necesario, apache Struts es uno de estos frameworks, en este proyecto se va a usar el framework Struts.

2.1.3 Struts

Struts es un framework para aplicaciones web java que implementa el modelo MVC. Fue creado por Craig McClanahan y donado a la Apache Software Foundation en el 2000 (pertenece a Apache Jakarta).

Beneficios

- Refuerza la modularidad y partición de la aplicación
- Incrementa la separación de roles
- Incrementa la manejabilidad del código
- Incrementa la extensibilidad del código
- Centra al desarrollador en la lógica de negocio.
- Reduce el tiempo de desarrollo de una aplicación, reduciendo su costo de desarrollo y mantenimiento

Arquitectura de Struts

Modelo: combinación de EJBs, Java Beans y clases derivadas de la clase de Struts *ActionForm*

Vista: representada por JSPs, para los cuales Struts proporciona librerías de JSP Tags. Aunque no obliga a usar JSPs, favorece su utilización.

Controlador: implementado a través del servlet *ActionServlet* que llama a las clases derivadas de *Action* que se definan.

Configuración: a través del fichero *web.xml* (común a toda aplicación web) y el fichero *struts-config.xml* (común a toda aplicación que haga uso de Struts)

Flujo de Control en Struts

La figura 2.2 muestra el flujo de control en Struts en base a una petición hecha desde el navegador cliente.

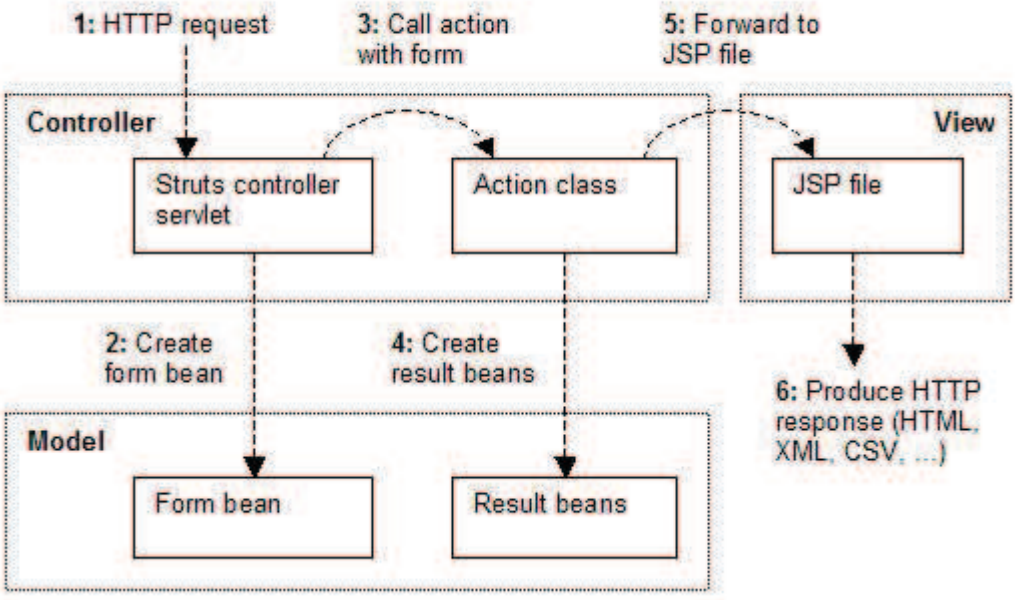

Figura 2.2 Flujo de control en Struts

La clase org.apache.struts.action.ActionServlet es el eje de Struts. Dada una petición de entrada HTTP:

- Crea un objeto ActionForm donde guarda y valida los parámetros de entrada.
- Decide que objeto Action se debe invocar y le pasa el objeto ActionForm creado.
- Transfiere control a la siguiente etapa de procesamiento de la petición (típicamente un JSP).
- El fichero de configuración web.xml contiene la asignación de pedidos para enviar las peticiones de llegada al ActionServlet, mientras que el fichero de configuración de Struts struts-config.xml contiene las asignaciones a las acciones
- Los form beans[1] creados por ActionServlet deben ser implementados por el programador, extendiendo org.apache.struts.action.ActionForm.
- El programador deberá definir un conjunto de métodos que lean y escriban las propiedades y además sobrescribir los métodos validate() y reset()

[1]Form beans. Clase especial en Java que contiene los valores del formulario enviado al servidor.

- Los objetos Action invocados deben ser desarrollados por el programador y extienden org.apache.struts.action.Action.
- Tienen un método execute() o (perform() en Struts 1.0) que ejecuta la lógica de negocio
- La acción devuelve un objeto ActionForward al servlet que especifica el siguiente paso a ejecutar, normalmente se transfiere el control a un JSP para que visualice los resultados.

Pasos para desarrollar una aplicación en Struts

1. Diseñar la aplicación en términos de las acciones, vistas y estados del modelo.
2. Añadir las librerías Java de Struts y los .tlds de sus etiquetas personalizadas al proyecto.
3. Configurar web.xml para que envíe peticiones HTTP al ActionServlet.
4. Configurar el ActionServlet definiendo elementos <actionmappings> y <form-beans> en struts-config.xml.
5. Definir clases Action.
6. Definir clases ActionForm.
7. Definir clases adicionales Java representando la lógica de negocio.
8. Definir páginas de presentación JSP
9. Explotar la aplicación.

Actions en Struts.

- Se crea una acción extendiendo la clase org.apache.struts.action.Action
- El ActionServlet ejecuta acciones invocando el método execute() de la clase Action
    - El método execute() contiene código para manipular el modelo
- Dentro del método execute() tienes acceso a:
    - Cabeceras y parámetros de peticiones HTTP
    - Atributos/beans guardados en los contextos
    - application/session/request scope
    - Struts ActionForm asociados con la acción (opcional)
    - El ActionMapping asociado a esta acción (opcional)
    - El objeto httpResponse
- El método execute() devuelve un objeto ActionForward que indica al ActionServlet a dónde transferir el control a continuación

Form Beans

- Un ActionForm es un JavaBean con propiedades que corresponden a los controles de un formulario HTML. Los parámetros son mapeados a propiedades del bean.
    - Proveen un mecanismo de buffer/validate/convert que se necesita para asegurar que el usuario introduce los datos esperados
    - Actúa como puente entre el navegador y el objeto de negocio

- El programador define un form bean extendiendo la clase org.apache.struts.action.ActionForm (o de forma declarativa usando org.apache.struts.action.DynaActionForm)
- Hay que definir cada una de las propiedades en la clase y escribir los getters/setters correspondientes, siguiendo las reglas de JavaBeans
- Después de escribir el código del form bean, es necesario asociarlo con una o más acciones a través del fichero de configuración de Struts struts-config.xml
- Cada vez que se llama a la acción , el ActionServlet poblará las propiedades con los valores de los parámetros recibidos en el formulario HTML
- Las propiedades no sólo pueden ser escalares sino que también pueden ser colecciones de valores

¿Por qué se necesitan Forms Beans?

¿Por qué no simplemente se accede a los parámetros de una petición, en vez de usar un form bean como intermediario?

Razones:

- Los Form beans pueden ser validados antes de que una acción sea invocada
    - Si la propiedad validate de un elemento Action en el fichero struts-config.xml contiene el valor true (por defecto), el método validate será invocado
- Si un form bean falla en la validación puede hacer que Struts envíe al usuario de vuelta a la vista (JSP) desde la que se realizó el POST en primer lugar, junto con un mensaje de error
    - Es importante asignar a la propiedad input del elemento action en struts-config.xml una referencia a la vista que entregó el formulario
- Los form beans pueden ser usados por múltiples acciones o incluso una aplicación completa
    - Si se configura al ActionController para que guarde un form bean en el contexto de sesión se permitirá el uso del form bean en todas las peticiones web que llegan a la aplicación

Recursos de String/Manejo de Errores

Recursos de String:

- Todos los strings usados en mensajes en JSPs tanto de datos de aplicación como de errores puedes ser colocados en un ficheros de recursos (ApplicationResources.properties).
- Struts proporciona etiquetas personalizadas JSP tags para acceder a estos recursos
  <bean:message key="prompt.goodguess.heading"/>

Manejo de errores

- Se recomienda que el manejo de errores se lleve a cabo dentro del método execute() y el control se transfiera a la vista apropiada
  errors.add("passphrase",new ActionError("error.passphrase.required"));
- Struts proporciona un mecanismo separado para permitir la implementación de un manejador global de excepciones
- Struts proporciona etiquetas JSP que ayudan en la visualización de los errores en tu JSP:
  <html:errors property="passphrase"/>

Internacionalización i18n

- Struts soporta internacionalización a través de ficheros de recursos, sus librerías de etiquetas personalizadas y Java Locales
- Se pueden definir strings a visualizar en el fichero de recursos, y luego ser usados en los JSPs
- Los strings del idioma por defecto se guardan en el fichero ApplicationResources.properties
- Otros ficheros de recursos adicionales pueden ser definidos que incluyen el idioma a usar en el nombre del fichero.
  - Ejemplo:
    - ApplicationResources_eu.properties (Contiene mensajes en Euskera)
    - ApplicationResources_es.properties (Contiene mensajes en Castellano)
    - Para cambiar el idioma preferido en IE: Tools - Internet Options - Languages
- Los navegadores envían la cabecera HTTP Accept-Language en cada petición que indica cuál es el idioma preferido por el usuario
- Los ficheros de recursos deben colocarse en un lugar accesible desde el CLASSPATH de la aplicación web, por ejemplo, debajo del directorio WEBINF/classes

2.2 Seguridad en Aplicaciones web.

La seguridad es uno de los aspectos a resolver con importancia en estos días tal como dice Cobit y también diferentes ISO's, y por ello un punto importante dentro de cualquier proyecto.

2.2.1 Autenticación y autorización.

La seguridad se centra sobre dos conceptos básicos: autenticación y autorización. Usuarios se autentican al sistema probando que son lo que dicen ser, mientras la autorización permite/no-permite acceso a ciertas áreas de la aplicación.

La terminología en temas de seguridad puede ser algo confusa se acaba de hablar de autenticación y autorización. Ahora se agrega otro concepto: el rol de usuario. Un rol de usuario son los recursos que el usuario puede tener. A un rol de usuario se le puede pensar como una llave que el usuario tiene; al autenticarse se le proveen los roles que le corresponden que le permiten acceder a los recursos.

Un ejemplo típico es una aplicación que tiene dos roles solamente: uno para un usuario estándar y otro para usuarios administradores. Las páginas comunes tendrán asociadas el rol de usuario estándar, y las páginas de administración, el rol de administrador; de esta forma, sólo los usuarios con rol de administrador podrán acceder a estas últimas.

En J2EE los contenedores web tienen soporte para construir mecanismos de seguridad para sus aplicaciones. En concreto mediante el uso de Realm (también conocido por Security Police Domain). Este mecanismo forma parte de la especificación JEE v.5, por lo que cualquier servidor de aplicaciones "decente" proporcionará los mecanismos necesarios para su implantación y uso.

Los contenedores web J2EE ofrecen tres tipos de mecanismos de autenticación: basic, form-based, y autenticación mutua. La mayoría de las aplicaciones web usan el tipo de form-based ya que permite que la interfaz del usuario sea personalizable (HTML). La autorización es implementada por los contenedores a través de roles de seguridad definidos en el descriptor de la aplicación web (web.xml).

En este proyecto se utilizara en el tipo Form-based validar las credenciales del usuario.

2.2.2 SecurityFilter

El proyecto SecurityFilter, es un proyecto de código abierto que implementa un filtro para asegurar las aplicaciones. Se escoge este proyecto ya que imita la seguridad proporcionada por un contenedor web, además el archivo de configuración sigue al estándar (web.xml) lo cual lo hace fácil cambiar a SecurityFilter de seguridad manejada por un contenedor Web o viceversa. Por tanto se logra tener una seguridad robusta logrando dejar independiente el contenedor.

Filtros

Los servlets filters son componentes de software que, asociados a direcciones web, permiten interceptar tanto el pedido como la respuesta para realizar acciones acordes. Por lo general, un filtro no crea la respuesta al pedido sino que simplemente la modifica o realiza operaciones sobre ella o el pedido que la generó.

Los filtros, que si bien son llamados así, se verá que no necesariamente filtran datos o información, son útiles para realizar tareas recurrentes, como pueden ser:

Seguridad: un filtro revisa si el pedido está autorizado y, en caso negativo, lo bloquea o lo redirige.
Auditoría: un filtro que lleve un registro de las veces que determinado recurso fue accedido, puede tomar el tiempo que tardo en procesarse, etc.
Conversión: un filtro que transforme una respuesta genérica a un formato concreto.
Encriptación: un filtro que encripte la información que es recibida o enviada de/y al cliente.
Compresión: un filtro que comprima la información enviada al cliente para ahorrar ancho de banda.
Disparador: un filtro que, al acceder a determinado recurso, dispare un evento.

Un filtro puede estar asociado con cero o más pedidos, y un pedido puede tener asociados cero o más filtros, creando una cadena de filtros que se ejecutan antes de llamarse al servlet en sí.

Los filtros agregan modularidad a la aplicación, concentrando funcionalidad que puede abarcar varias acciones en un solo lugar. Además el hecho de poder configurarlos y/o activarlos o desactivarlos desde un archivo de configuración ayuda a empaquetar funcionalidades sin necesidad de modificar el código.

SecurityFilter usa la misma arquitectura que Form-Based donde se configura a través de descriptores en la aplicación.

Configuración de SecurityFilter

Se van a ver los elementos del archivo de configuración de SecurityFilter, por defecto llamado **securityfilter-config.xml**.

El elemento **security-constraint** define una restricción de seguridad. En este elemento es donde se asocian los roles requeridos a recursos web.
Sus sub-elementos son:

display-name (opcional): permite definir un nombre visible a la restricción (para ser usado generalmente por herramientas de desarrollo).
web-resource-collection (al menos uno): en este elemento se definen los recursos que se están restringiendo.
aunth-constraint (opcional): aquí se definen los roles a asociar a los recursos. Si este elemento no es especificado, significa que los recursos son públicos.
user-data-constraint (opcional): este elemento es aceptado por la sintaxis de configuración de SecurityFilter pero no es usado. Según la especificación sirve para especificar el tipo de conexión requerido entre el cliente y el servidor para los recursos definidos. Mediante este elemento se podría requerir que determinados recursos sólo estén disponibles bajo una conexión encriptada.

Se vera ahora como definir el recurso **web-resource-collection**.

Sus sub-elementos son:

web-resource-name: el nombre de los recursos que se están definiendo.
descripction (opcional): una descripción.
url-pattern (cero o más): patrón de direcciones web.
http-method (cero o más): métodos HTTP que se quieren restringir con la definición (GET, POST, etc.). Si se emite este sub-elemento, por defecto la restricción abarca a todos los métodos HTTP.

El elemento **auth-constraint** se define mediante sus dos sub-elementos:

description (opcional): Una descripción.
role-name (cero o más): los roles asociados a la restricción.

Es menester mostrar un ejemplo.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name> Paginas privadas </web-resource-name>
    <descripction>Prevenimos acceder a las paginas JSP directamente
       por el     usuario</description>
    <url-pattern>*.jsp</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>private</role-name>
  </aunth-constraint>
</security-constraint>
```

Este sencillo aunque muy útil ejemplo previene que el usuario accede directamente a recursos de extensión JSP; en la práctica, previene que acceda a las paginas JSP directamente.

Se pasara ahora al siguiente elemento del archivo de configuración: **login-config,** el cual define como es autenticado el usuario.

Se vera los sub-elementos que lo componen:

auth-method (opcional): el tipo de autenticación. Aquí se usara **FORM.** SecurityFilter también soporte **BASIC**, donde el formulario para ingresar lo provee el navegador.
real-name (opcional): es el nombre del reino que se usara para autenticar usuarios.
Este elemento es usado solamente para autenticación **BASIC**.
form-login-config (opcional): define la configuración de las páginas que se usaran para la autenticación de tipo FORM.

Ahora se verán los sub-elementos de este último elemento:

form-login-page: la dirección de la página de autenticación.

form-error-page: la dirección de la página de fallo en la autenticación.
form-default-page: la dirección por defecto de ingreso luego de que un usuario se autentica correctamente.
form-logout-page (opcional): la dirección por defecto luego de que un usuario sale del sistema.
remenber-me: elemento usado para la autenticación automática de usuarios.

Un ejemplo típico de este elemento podría ser:

```
<login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
     <form-login-page>/index.do</form-login-page>
     <form-error-page>jsp/login/error_login.jsp</form-error-page>
     <form-default-page>/principal.do</form-default-page>
     <form-logout-page>/salida.do</ form-logout-page>
   </form-login-config>
 </login-config>
```

Y ahora el último elemento del archivo de configuración: **realm** (al menos uno). Este define la clase que efectuara la autenticación. Si se define más de un elemento, cada clase que se defina será usada como adaptador de la clase del elemento subsiguiente. En este caso dado que se usa Tomcat, se necesita de esta funcionalidad para usar un adaptador a la clase de autenticación de Catalina(el servidor de servlets de Tomcat).

El proyecto SecurityFilter toma la configuración y demás(como usar el valor **j_security_check** como nombre de acción) de la configuración de seguridad de J2EE. De esta forma es más fácil migrar desde y hacia la seguridad manejada por el servidor J2EE.

2.3 Diseño de la Base de Datos

En el capítulo anterior se vio el modelo Entidad - Relación el cual representaba los requerimientos de la base de datos para el sistema CoTER. Ahora este modelo se debe trasformar al modelo relacional.

2.3.1 Modelo relacional.

Se trata de un modelo bastante potente y a la vez bastante simple. Este modelo permite almacenar todos los datos en tablas. Es decir las entidades y las relaciones del modelo entidad –relación pasaran a ser tablas en este modelo. Las entidades pasan a ser tablas como tal y las relaciones se meten dentro de las tablas como un atributo más de éstas, esto para que haya interacción entre las tablas. Este modelo es en el que se basan la mayoría de los SGBD comerciales y libres en uso hoy en día. El SGBD que usara el sistema CoTER es MySQL.

Cada tabla contiene sus atributos (columnas) y tuplas (filas).

Atributo: se trata de cada una de las columnas de la tabla. Vienen definidas por un nombre y pueden contener un conjunto de valores. En el modelo entidad relación representan las características de una entidad.
Tupla: se trata de cada una de las filas de la tabla. Es importante señalar que no se pueden tener tuplas duplicadas en una tabla.

Dado que no se pueden tener tuplas duplicadas surge el concepto de llave:
Llave: Es el o los atributos que identifican de manera única una tupla en una tabla.

Además se tienen distintos tipos de llaves

Llave primaria: formada por un solo elemento.
Llave foránea: es el atributo que viaja entre dos tablas para que tengan interacción.

2.3.2 Normalización

Qué es la normalización

La normalización es el proceso mediante el cual se transforman datos complejos a un conjunto de estructuras de datos más pequeñas, que además de ser más simples y más estables, son más fáciles de mantener. También se puede entender la normalización como una serie de reglas que sirven para ayudar a los diseñadores de bases de datos a desarrollar un esquema que minimice los problemas de lógica. Cada regla está basada en la que le antecede. La normalización se adoptó porque el viejo estilo de poner todos los datos en un solo lugar, como un archivo o una tabla de la base de datos, era ineficiente y conducía a errores de lógica cuando se trataban de manipular los datos.

La normalización también hace las cosas fáciles de entender. Los seres humanos tienen la tendencia de simplificar las cosas al máximo. Se hace con casi todo, desde los animales hasta con los automóviles. Al ver una imagen de gran tamaño se hace más simple agrupando cosas similares juntas. Las guías que la normalización provee crean el marco de referencia para simplificar una estructura de datos compleja.

Otra ventaja de la normalización de base de datos es el consumo de espacio. Una base de datos normalizada ocupa menos espacio en disco que una no normalizada. Hay menos repetición de datos, lo que tiene como consecuencia un mucho menor uso de espacio en disco.

El proceso de normalización tiene un nombre y una serie de reglas para cada fase. Esto puede parecer un poco confuso al principio, pero poco a poco se va entendiendo el proceso, así como las razones para hacerlo de esta manera.

La normalización comprueba que las tablas (también denominadas relaciones en terminología propia del modelo relacional de datos) definidas cumplen unas

determinadas condiciones. Se pretende garantizar la no existencia de redundancia y una cierta coherencia en la representación mediante un esquema relacional de las entidades y relaciones del modelo conceptual (diagrama E-R).

Grados de normalización

Existen básicamente tres niveles de normalización: Primera Forma Normal (1NF), Segunda Forma Normal (2NF) y Tercera Forma Normal (3NF). Cada una de estas formas tiene sus propias reglas. Cuando una base de datos se conforma a un nivel, se considera normalizada a esa forma de normalización. No siempre es una buena idea tener una base de datos conformada en el nivel más alto de normalización, puede llevar a un nivel de complejidad que pudiera ser evitado si estuviera en un nivel más bajo de normalización.

En la figura 2.3 se describe brevemente en que consiste cada una de las reglas, y posteriormente se explican con más detalle.

Regla	Descripción
Primera Forma Normal (1FN)	Incluye la eliminación de todos los grupos repetidos.
Segunda Forma Normal (2FN)	Asegura que todas las columnas que no son llave sean completamente dependientes de la llave primaria (PK).
Tercera Forma Normal (3FN)	Elimina cualquier dependencia transitiva. Una dependencia transitiva es aquella en la cual las columnas que no son llave son dependientes de otras columnas que tampoco son llave.

Figura 2.3 Reglas de normalización

Primera Forma Normal

La regla de la Primera Forma Normal establece que las columnas repetidas deben eliminarse y colocarse en tablas separadas.

Poner la base de datos en la Primera Forma Normal resuelve el problema de los encabezados de columna múltiples. Muy a menudo, los diseñadores de bases de datos inexpertos harán algo similar a la tabla no normalizada. Una y otra vez, crearán columnas que representen los mismos datos. La normalización ayuda a clarificar la base de datos y a organizarla en partes más pequeñas y más fáciles de entender. En lugar de tener que entender una tabla gigantesca y monolítica que tiene muchos diferentes aspectos, sólo se tiene que entender los objetos pequeños y más tangibles, así como las relaciones que guardan con otros objetos también pequeños.

Segunda Forma Normal

La regla de la Segunda Forma Normal establece que todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas. Una dependencia parcial es un término que describe a aquellos datos que no dependen de la llave primaria de la tabla para identificarlos.

Una vez alcanzado el nivel de la Segunda Forma Normal, se controlan la mayoría de los problemas de lógica. Se puede insertar un registro sin un exceso de datos en la mayoría de las tablas.

Tercera Forma Normal

Una tabla está normalizada en esta forma si todas las columnas que no son llave son funcionalmente dependientes por completo de la llave primaria y no hay dependencias transitivas. Comentamos anteriormente que una dependencia transitiva es aquella en la cual existen columnas que no son llave que dependen de otras columnas que tampoco son llave.

Cuando las tablas están en la Tercera Forma Normal se previenen errores de lógica cuando se insertan o borran registros. Cada columna en una tabla está identificada de manera única por la llave primaria, y no debe haber datos repetidos. Esto provee un esquema limpio y elegante, que es fácil de trabajar y expandir.

Un dato sin normalizar no cumple con ninguna regla de normalización.

¿Qué tan lejos debe llevar la normalización?

La siguiente decisión es ¿qué tan lejos debe llevar la normalización? La normalización es una ciencia subjetiva. Determinar las necesidades de simplificación depende de nosotros. Si la base de datos va a proveer información a un solo usuario para un propósito simple y existen pocas posibilidades de expansión, normalizar los datos hasta la 3FN quizá sea algo exagerado. Las reglas de normalización existen como guías para crear tablas que sean fáciles de manejar, así como flexibles y eficientes. A veces puede ocurrir que normalizar los datos hasta el nivel más alto no tenga sentido.

¿Se están dividiendo tablas sólo para seguir las reglas o estas divisiones son en verdad prácticas?. Éstas son el tipo de cosas que los diseñadores de la base de datos, necesitan decidir, y la experiencia y el sentido común pueden auxiliar para tomar la decisión correcta. La normalización no es una ciencia exacta, más bien subjetiva.

Existen seis niveles más de normalización que no se han discutido aquí. Ellos son Forma Normal Boyce- Codd, Cuarta Forma Normal (4NF), Quinta Forma Normal (5NF) o Forma Normal de Proyección-Unión, Forma Normal de Proyección-Unión Fuerte, Forma Normal de Proyección-Unión Extra Fuerte y Forma Normal de Clave de Dominio. Estas formas de normalización pueden llevar las cosas más allá de lo que se necesite. Éstas existen para hacer una base de datos realmente relacional. Tienen que ver principalmente con dependencias múltiples y claves relacionales.

2.3.3 Modelo relacional del sistema CoTER

En esta sección se verán las tablas del sistema CoTER.

Diccionario de datos

Un diccionario de datos es un conjunto de metadatos que contiene las características lógicas y puntuales de los datos que se van a utilizar en el sistema que se programa, incluyendo nombre, descripción, alias, contenido y organización.

Identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información, se desarrolla durante el análisis de flujo de datos y auxilia a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño.

En un diccionario de datos se encuentra la lista de todos los elementos que forman parte del flujo de datos de todo el sistema. Los elementos más importantes son flujos de datos, almacenes de datos y procesos. El diccionario de datos guarda los detalles y descripción de todos estos elementos.

El modelo relacional del sistema CoTER basado en el modelo Entidad – Relación del capítulo anterior se puede ver en el diccionario de datos que se muestra a continuación.

Diccionario de datos del sistema CoTER

Tabla Persona. Encargada de almacenar toda la información referente a las personas que hacen uso del sistema, tanto alumnos como profesores.

Columna	Tipo de Dato	Descripción
id_per	smallint	El identificador de cada tupla. Esta columna representa la llave primaria.
user	varchar	El usuario asignado a la persona que quiere accesar al sistema.
contrasena	varchar	La contraseña del usuario.
nombre	varchar	El nombre del usuario.
apellido_paterno	varchar	El apellido paterno del usuario.
apellido_materno	varchar	El apellido materno del usuario.
sexo	varchar	El sexo del usuario.
no_cuenta	varchar	El número de cuenta sólo para los alumnos, los demás usuarios del sistema es nulo.
e-mail	varchar	Correo electrónico del usuario.
celular	varchar	El número de celular del usuario.
calle	varchar	Calle perteneciente a la dirección del usuario.
numext	varchar	Número exterior perteneciente a la dirección del usuario.
numint	varchar	Numero interior perteneciente a la dirección del

		usuario.
colonia	varchar	Colonia perteneciente a la dirección del usuario.
delegacion	varchar	Delegación perteneciente a la dirección del usuario.
cp	varchar	Código postal perteneciente a la dirección del usuario.
tel	varchar	Teléfono de casa del usuario.
generacion	varchar	Generacion sólo para los alumnos, para los demás usuarios es nulo.
tipo	varchar	Sólo para profesores: si es asesor lleva el valor "1" si es revisor el valor "2".
id_per_fk	smallint	Llave foránea a esta misma tabla, representa la relación de que un profesor puede tener varios alumnos asignados.
id_rol_fk	smallint	Llave foránea a la tabla rol, representa de que varias personas pueden tener un mismo rol.
id_esc_fk	smallint	Llave foránea a la tabla escuela, representa de que una escuela puede tener varios alumnos.
id_car_fk	smallint	Llave foránea a la tabla carrera, representa de que una carrera tiene varios alumnos.
id_opc_fk	smallint	Llave foránea de que un opción de titulación tiene varios alumnos.
id_pue_fk	smallint	Llave foránea a la tabla puesto, representa de que un tipo de trabajo tiene varios alumnos.
id_tit_fk	smallint	Llave foránea a la tabla titulación, representa de que un trabajo de titulación puede tener varios alumnos.
id_cic_fk	smallint	Llave foránea a la tabla ciclo, representa de que un semestre puede tener varios alumnos registrados para titularse.

Tabla Puesto. Encargada de almacenar la información general de un puesto de trabajo.

Columna	Tipo de Dato	Descripción
id_pue	smallint	El identificador de cada tupla. Esta columna representa la llave primaria.
nombre	varchar	El nombre del puesto.
descripcion	varchar	Una breve descripción del puesto.

Tabla datos_puesto. Encargada de almacenar la información específica para cada tipo de puesto.

Columna	Tipo de Dato	Descripcion
id_pue	smallint	El identificador compuesto de cada tupla. Esta columna junto con la columna num_registro forma la llave primaria.

num_registro	smallint	El identificador compuesto de cada tupla. Esta columna junto con la columna id_pue forma la llave primaria. Además esta columna representa el conteo del mismo tipo de trabajo.
direccion	varchar	La direccion de la empresa.
tel	varchar	Teléfono del trabajo.
extension	varchar	Extension telefónica si la hay, si no es nulo.
nom_empresa	varchar	Nombre de la empresa.
antigüedad	varchar	Antigüedad del usuario en su trabajo.

Tabla roles. Encargada de almacenar la información referente al rol que puede tener un usuario en el sistema, como por ejemplo Administrador del sistema, alumno o profesor y dependiendo de cada rol tendrá distintas acciones a ejecutar.

Columna	Tipo de Dato	Descripcion
id_rol	smallint	El identificador del rol. Esta columna representa la llave primaria.
nombre	varchar	El nombre del rol
descripcion	varchar	Para qué sirve este rol. Es decir las acciones que pude llevar a cabo.

Tabla carrera. Encargada de almacenar la información referente a todas las carreras que se imparten en la FES Aragón.

Columna	Tipo de Dato	Descripcion
id_car	smallint	El identificador de una carrera en particular. Esta columna representa la llave primaria.
nombre	Varchar	El nombre de la carrera.
descripcion	Varchar	Breve descripción de la carrera.

Tabla opción. Encargada de almacenar la información referente a todas las opciones de titulación de la FES Aragón.

Columna	Tipo de Dato	Descripcion
id_opc	smallint	El identificador de una opción en particular. Esta columna representa la llave primaria.
nombre	varchar	El nombre de la carrera.
tiempo		El tiempo que tiene esta opción para acabar el trabajo.
descripcion	varchar	Breve descripcion de la opción.
articulo	varchar	Campo extra por si tiene que guardar una información posterior.

Tabla escuela. Encargada de almacenar la información que llevan las cartas del proceso de titulación.

Columna	Tipo de Dato	Descripcion
id_esc	smallint	El identificador de cada tupla. Esta columna forma la llave primaria.
nombre	smallint	Nombre de la escuela.
campus	varchar	Sólo para las FES: sirve para indicar que campus es.
nombre_corto	varchar	Nombre de la escuela es formato corto.
direccion_f1	varchar	Dirección de la escuela en formato uno.
direccion_f2	varchar	Dirección dela escuela en formato 2.
director	varchar	Director de la escuela.
jefe_carrera	varchar	Jefe de la carrea de ICO.
jefe_secretaria	varchar	Jefe de secretaria académica.

Tabla titulación. Encargada de almacenar la información del registro de titulación.

Columna	Tipo de Dato	Descripcion
id_tit	smallint	El identificador de cada tupla. Esta columna forma la llave primaria.
fecha_registro	date	Fecha de registro de la titulación.
fecha_terminacion	date	Fecha de terminación del trabajo.
fecha_examen	date	Fecha del examen profesional.
tema	varchar	Tema del trabajo.
descripcion	text	Descripción del trabajo.
estudios	varchar	Si es egresado el valor es “e”, si es alumno el valor es “a”.
mencion	varchar	Si puede aspirar el alumno a mención honorifica el valor es “s” de lo contrario “n”.

Tabla capitulado. Encargada de almacenar la información referente a todas las opciones de titulación de la FES Aragón.

Columna	Tipo de Dato	Descripcion
id_cap	smallint	El identificador de un capítulo en particular. Esta columna representa la llave primaria.
nombre	varchar	El nombre del capítulo.
indice		El número de capitulo según el índice.
id_tit	varchar	Llave foránea a la tabla titulación ya que un trabajo de titulación puede tener varios capítulos.

Tabla docs. Encargada de almacenar los archivos que el usuario suba.

Columna	Tipo de Dato	Descripcion
id_doc	smallint	El identificador de un archivo en particular. Esta

		columna representa la llave primaria.
archivo	longblob	Los datos binarios del archivo
fsubida	date	La fecha en que el usuario subió el archivo.
comentarios	varchar	Breve descripcion del archivo..
titulo	varchar	Título del archivo.
id_tit	smallint	Llave foránea a la tabla titulación, ya que un trabajo de titulación puede tener varios archivos.

Tabla ciclo. Encargada de almacenar la información referente a todas las opciones de titulación de la FES Aragón.

Columna	Tipo de Dato	Descripcion
id_cic	smallint	El identificador de un ciclo en particular. Esta columna representa la llave primaria.
valor	varchar	El ciclo escolar en si.

2.4 Diseño de la GUI

Los diseños de las pantallas se harán conforme a las tecnologías estándares en este caso como es un sistema Web se hará uso de XHTML, CSS, JavaScript y Ajax.

2.4.1 XHTML

Definiéndolo de forma sencilla, *"HTML es lo que se utiliza para crear todas las páginas web de Internet"*. Más concretamente, HTML es el *lenguaje* con el que se *"escriben"* la mayoría de páginas web.

Los diseñadores utilizan el lenguaje HTML para crear sus páginas web, los programas que utilizan los diseñadores generan páginas escritas en HTML y los navegadores que utilizan los usuarios muestran las páginas web después de leer su contenido HTML.

Aunque HTML es un lenguaje que utilizan los ordenadores y los programas de diseño, es muy fácil de aprender y escribir por parte de las personas. En realidad, HTML son las siglas de *HyperText Markup Language* y más adelante se verá el significado de cada una de estas palabras.

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado World Wide Web Consortium[1], más conocido como **W3C**. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de forma muy similar en cualquier navegador de cualquier sistema operativo.

[1] Enlace al sitio del W3C http://www.w3.org.

El propio **W3C** define el lenguaje HTML como *"un lenguaje reconocido universalmente y que permite publicar información de forma global"*. Desde su creación, el lenguaje HTML ha pasado de ser un lenguaje utilizado exclusivamente para crear documentos electrónicos a ser un lenguaje que se utiliza en muchas aplicaciones electrónicas como buscadores, tiendas online y banca electrónica.

Breve historia de HTML

La historia completa de HTML es tan interesante como larga, por lo que a continuación se muestra su historia resumida a partir de la información que se puede encontrar en la Wikipedia.

El origen de HTML se remonta a 1980, cuando el físico **Tim Berners-Lee**, trabajador del CERN[1] (*Organización Europea para la Investigación Nuclear*) propuso un nuevo sistema de *"hipertexto"* para compartir documentos.

Los sistemas de *"hipertexto"* habían sido desarrollados años antes. En el ámbito de la informática, el *"hipertexto"* permitía que los usuarios accedieran a la información relacionada con los documentos electrónicos que estaban visualizando. De cierta manera, los primitivos sistemas de *"hipertexto"* podrían asimilarse a los enlaces de las páginas web actuales.

Tras finalizar el desarrollo de su sistema de *"hipertexto"*, Tim Berners-Lee lo presentó a una convocatoria organizada para desarrollar un sistema de *"hipertexto"* para Internet. Después de unir sus fuerzas con el ingeniero de sistemas **Robert Cailliau**, presentaron la propuesta ganadora llamada *WorldWideWeb (W3)*.

El primer documento formal con la descripción de HTML se publicó en 1991 bajo el nombre *"HTML Tags"*[2] (*Etiquetas HTML*) y todavía hoy puede ser consultado online a modo de *reliquia informática*.

La primera propuesta oficial para convertir HTML en un estándar se realizó en 1993 por parte del organismo IETF[3] (*Internet Engineering Task Force*). Aunque se consiguieron avances significativos (en esta época se definieron las etiquetas para imágenes, tablas y formularios) ninguna de las dos propuestas de estándar, llamadas HTML y HTML+ consiguieron convertirse en estándar oficial.

En 1995, el organismo IETF organiza un grupo de trabajo de HTML y consigue publicar, el 22 de septiembre de ese mismo año, el estándar HTML 2.0. A pesar de su nombre, HTML 2.0 es el primer estándar oficial de HTML.

---

1 Enlace al sitio del CERN http://www.cern.ch.
2 Enlace a la descripcion de HTML http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html.
3. Enlace al sitio del IETF http://www.ietf.org.

A partir de 1996, los estándares de HTML los publica otro organismo de estandarización llamado W3C. La versión HTML 3.2 se publicó el 14 de Enero de 1997 y es la primera recomendación de HTML publicada por el W3C.

Esta revisión incorpora los últimos avances de las páginas web desarrolladas hasta 1996, como *applets* de Java y texto que fluye alrededor de las imágenes. HTML 4.0 se publicó el 24 de Abril de 1998 (siendo una versión corregida de la publicación original del 18 de Diciembre de 1997) y supone un gran salto desde las versiones anteriores.

Entre sus novedades más destacadas se encuentran las hojas de estilos CSS, la posibilidad de incluir pequeños programas o *scripts* en las páginas web, mejora de la accesibilidad de las páginas diseñadas, tablas complejas y mejoras en los formularios.

La última especificación oficial de HTML se publicó el 24 de diciembre de 1999 y se denomina HTML 4.01. Se trata de una revisión y actualización de la versión HTML 4.0, por lo que no incluye novedades significativas.

Desde la publicación de HTML 4.01, la actividad de estandarización de HTML se detuvo y el W3C se centró en el desarrollo del estándar XHTML. Por este motivo, en el año 2004, las empresas Apple, Mozilla y Opera mostraron su preocupación por la falta de interés del W3C en HTML y decidieron organizarse en una nueva asociación llamada WHATWG[1] (*Web Hypertext Application Technology Working Group*).

La actividad actual del WHATWG se centra en el futuro estándar HTML 5, cuyo primer borrador oficial[2] se publicó el 22 de enero de 2008. Debido a la fuerza de las empresas que forman el grupo WHATWG y a la publicación de los borradores de HTML 5.0, en marzo de 2007 el W3C decidió retomar la actividad estandarizadora de HTML[3].

De forma paralela a su actividad con HTML, W3C ha continuado con la estandarización de XHTML, una versión *avanzada* de HTML y basada en XML. La primera versión de XHTML se denomina XHTML 1.0 y se publicó el 26 de Enero de 2000 (y posteriormente se revisó el 1 de Agosto de 2002).

XHTML 1.0 es una adaptación de HTML 4.01 al lenguaje XML, por lo que mantiene casi todas sus etiquetas y características, pero añade algunas restricciones y elementos propios de XML. La versión XHTML 1.1 ya ha sido publicada en forma de borrador y pretende modularizar XHTML. También ha sido publicado el borrador de XHTML 2.0, que supondrá un cambio muy importante respecto de las anteriores versiones de XHTML.

1 Enlace al sitio del WHATWG http://www.whatwg.org/.
2 Enlace al borrador oficial http://www.w3.org/TR/html5/.
3 Enlace al estándar de HTML http://www.w3.org/2007/03/html-pressrelease.

Especificación oficial

El organismo W3C elabora las normas que deben seguir los diseñadores de páginas web para crear las páginas HTML. Las normas oficiales están escritas en inglés y se pueden consultar de forma gratuita en las siguientes direcciones:

- Especificación oficial de HTML 4.01 (http://www.w3.org/TR/html401/)
- Especificación oficial de XHTML 1.0 (http://www.w3.org/TR/xhtml1/)

El estándar XHTML 1.0 incluye el 95% del estándar HTML 4.01, ya que sólo añade pequeñas mejoras y modificaciones menores. Afortunadamente, no es necesario leer las especificaciones y recomendaciones oficiales de HTML para aprender a diseñar páginas con HTML o XHTML. Las normas oficiales están escritas con un lenguaje bastante formal y algunas secciones son difíciles de comprender.

.

2.4.2 CSS

CSS (Cascading Style Sheets, Hojas de estilo en cascada) es un lenguaje creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados *"documentos semánticos"*). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para *marcar* los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

Breve historia de CSS

Las hojas de estilos aparecieron alrededor del año 1970. Desde la creación de los lenguajes de etiquetas, se observó la necesidad de definir un mecanismo que permitiera aplicar de forma consistente diferentes estilos a los documentos electrónicos.

El gran impulso de los lenguajes de hojas de estilos se produjo con el boom de Internet y el crecimiento exponencial del lenguaje HTML para la creación de

documentos electrónicos. La guerra de navegadores y la falta de un estándar para la definición de los estilos dificultaban la creación de documentos con la misma apariencia en diferentes navegadores.

El organismo W3C, encargado de crear todos los estándares relacionados con la web, propuso la creación de un lenguaje de hojas de estilos específico para el lenguaje HTML y se presentaron nueve propuestas.

Las dos propuestas que se tuvieron en cuenta fueron la CHSS (*Cascading HTML Style Sheets*) y la SSP (*Stream-based Style Sheet Proposal*). La propuesta CHSS fue realizada por Håkon Wium Lie y SSP fue propuesto por Bert Bos. Entre finales de 1994 y 1995 Lie y Bos se unieron para definir un nuevo lenguaje que tomaba lo mejor de cada propuesta y lo llamaron CSS (*Cascading Style Sheets*).

En 1995, el W3C decidió apostar por el desarrollo y estandarización de CSS y lo añadió a su grupo de trabajo de HTML. A finales de 1996, el W3C publicó la primera recomendación oficial, conocida como "CSS nivel 1".

A principios de 1997, el W3C decide separar los trabajos del grupo de HTML en tres secciones: el grupo de trabajo de HTML, el grupo de trabajo de DOM (Document Object Model, Modelo de objetos del documento) y el grupo de trabajo de CSS.

El 12 de Mayo de 1998, el grupo de trabajo de CSS publica su segunda recomendación oficial, conocida como "CSS nivel 2". La versión de CSS que utilizan todos los navegadores de hoy en día es CSS 2.1, una revisión de CSS 2 que aún se está elaborando (la última actualización es del 19 de julio de 2007). Al mismo tiempo, la siguiente recomendación de CSS, conocida como "CSS nivel 3", continúa en desarrollo desde 1998 y hasta el momento sólo se han publicado borradores.

La adopción de CSS por parte de los navegadores ha requerido un largo periodo de tiempo. El mismo año que se publicó CSS 1, Microsoft lanzaba su navegador Internet Explorer 3.0, que disponía de un soporte bastante reducido de CSS. El primer navegador con soporte completo de CSS 1 fue la versión para Mac de Internet Explorer 5, que se publicó en el año 2000. Por el momento, ningún navegador tiene soporte completo de CSS 2.1.

Especificación oficial

La especificación o norma oficial que se utiliza actualmente para diseñar páginas web con CSS es la versión CSS 2.1, actualizada por última vez el 19 de julio de 2007 y que se puede consultar libremente en http://www.w3.org/TR/CSS21/

Desde hace varios años, el organismo W3C trabaja en la elaboración de la próxima versión de CSS, conocida como CSS 3. Esta nueva versión incluye miles de cambios importantes en todos los niveles y es mucho más avanzada y compleja que CSS 2.

No obstante, pasarán muchos años hasta que se publique la versión definitiva completa de CSS 3 y hasta que los principales navegadores del mercado incluyan la mayor parte del nuevo estándar.
El sitio web del organismo W3C dispone de una sección en la que se detalla el trabajo que el W3C está desarrollando actualmente en relación a CSS[1] y también dispone de un blog en el que se publican todas las novedades relacionadas con CSS[2].

2.4.3 JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Breve historia

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

---

[1]Sitio web del organismo W3C en relación a CSS http://www.w3.org/Style/CSS/current-work.
[2] Sitio web del W3C en relación al blog de CSS http://www.w3.org/blog/CSS).

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA (*European Computer Manufacturers Association*).

ECMA creó el comité TC39 con el objetivo de *"estandarizar un lenguaje de script multiplataforma e independiente de cualquier empresa"*. El primer estándar que creó el comité TC39 se denominó **ECMA-262**, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.

La organización internacional para la estandarización (ISO) adoptó el estándar ECMA-262 a través de su comisión IEC, dando lugar al estándar ISO/IEC-16262.

JavaScript y navegadores

Los navegadores más modernos disponibles actualmente incluyen soporte de JavaScript hasta la versión correspondiente a la tercera edición del estándar ECMA-262.

La mayor diferencia reside en el *dialecto* utilizado, ya que mientras Internet Explorer utiliza JScript, el resto de navegadores (Firefox, Opera, Safari, Konqueror) utilizan JavaScript.

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.

2.4.4 Ajax

El término AJAX se presentó por primera vez en el artículo "Ajax: A New Approach to Web Applications"[1] publicado por Jesse James Garrett el 18 de Febrero de 2005. Hasta ese momento, no existía un término normalizado que hiciera referencia a un nuevo tipo de aplicación web que estaba apareciendo.

En realidad, el término AJAX es un acrónimo de *Asynchronous JavaScript + XML*, que se puede traducir como "JavaScript asíncrono + XML".

El artículo define AJAX de la siguiente forma:

"Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes."

Las tecnologías que forman AJAX se muestran en la figura 2.4.

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

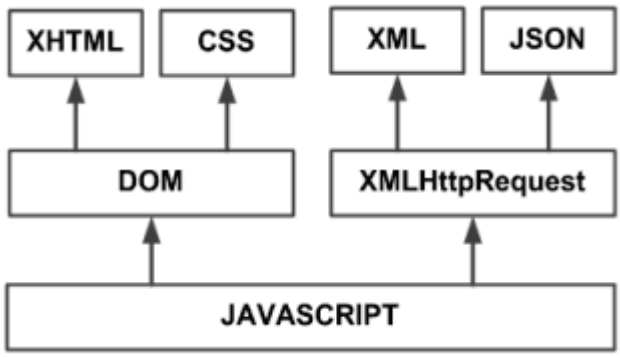

Figura 2.4. Tecnologías agrupadas bajo el concepto de AJAX

Desarrollar aplicaciones AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores.

En las aplicaciones web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

[1] Enlace al artículo de Jesse James.
http://www.adaptivepath.com/publications/essays/archives/000385.php

En el siguiente esquema de la figura 2.5, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX:

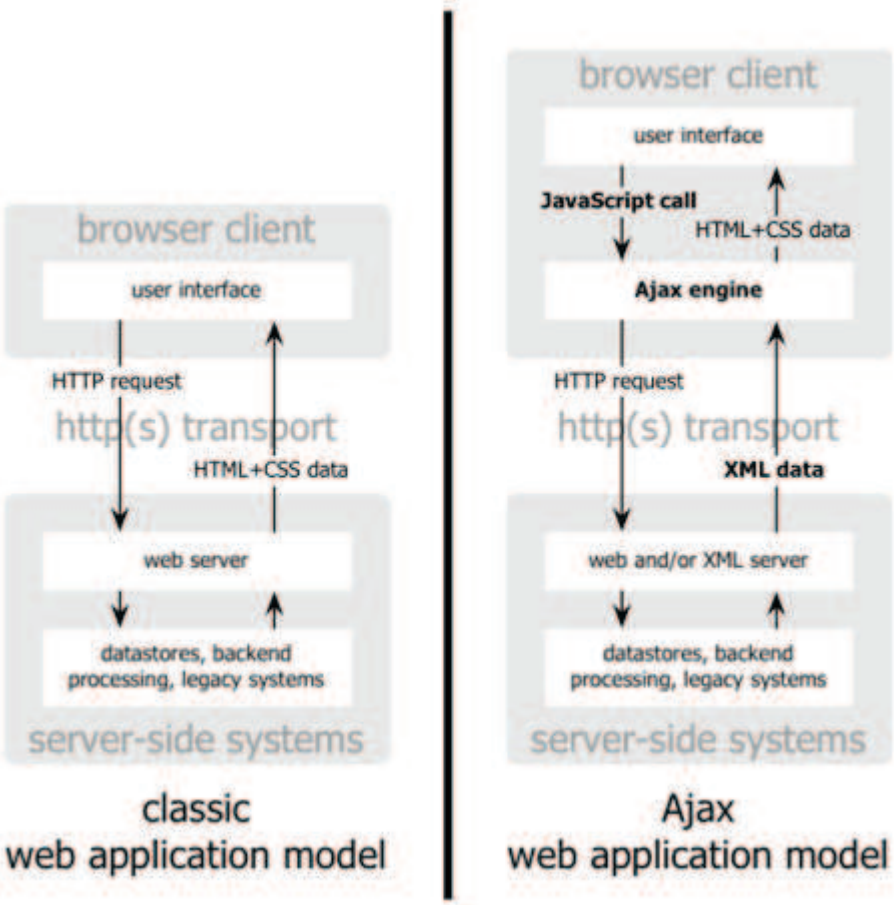

Figura 2.5 Comparativo de peticiones

Esta técnica tradicional para crear aplicaciones web funciona correctamente, pero no crea una buena sensación al usuario. Al realizar peticiones continuas al servidor, el usuario debe esperar a que se recargue la página con los cambios solicitados. Si la aplicación debe realizar peticiones continuas, su uso se convierte en algo molesto

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

El siguiente esquema de la figura 2.6 se muestra la diferencia más importante entre una aplicación web tradicional y una aplicación web creada con AJAX. La imagen superior muestra la interacción síncrona propia de las aplicaciones web tradicionales. La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX.

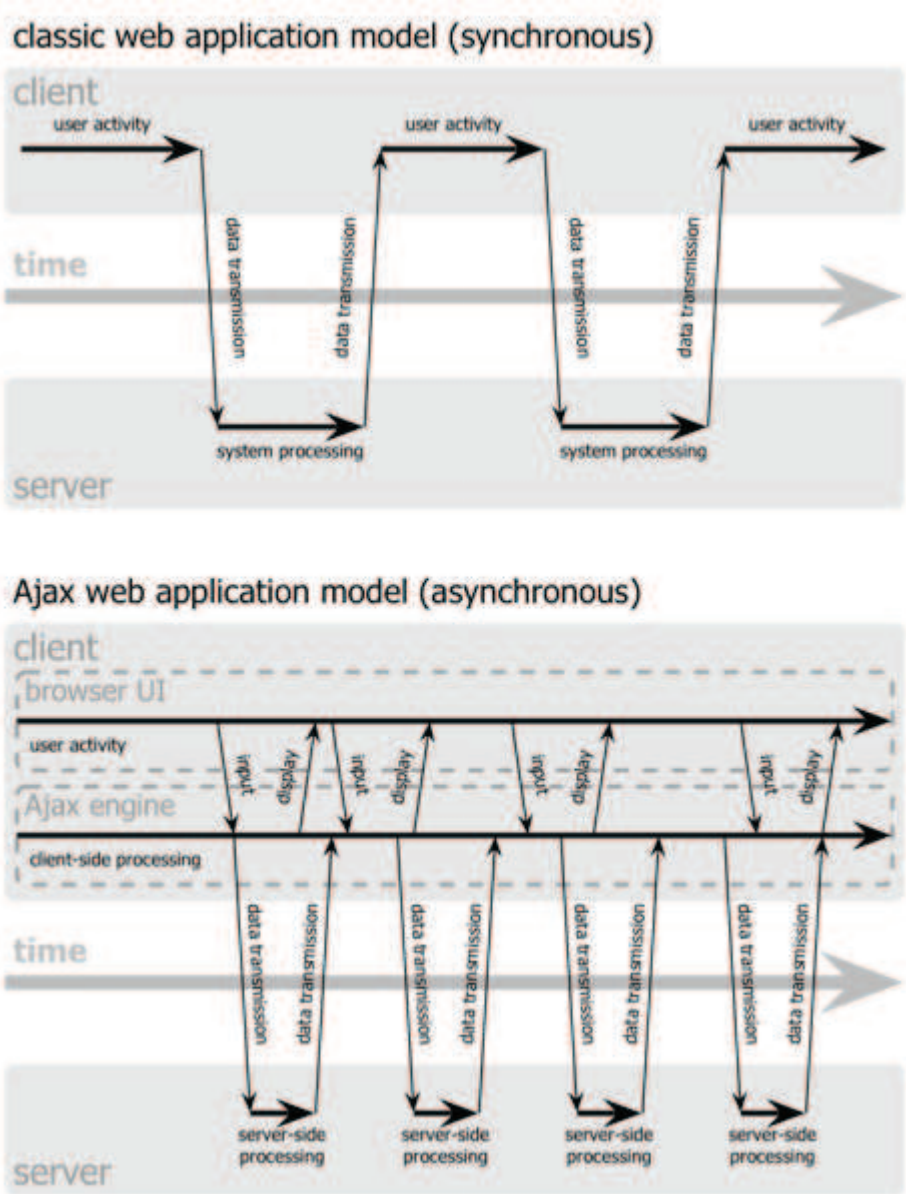

Figura 2.6. Comparación entre las comunicaciones síncronas de las aplicaciones web tradicionales y las comunicaciones asíncronas de las aplicaciones AJAX (Imagen original creada por Adaptive Path y utilizada con su permiso)

Las peticiones HTTP al servidor se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Desde su aparición, se han creado cientos de aplicaciones web basadas en AJAX. En la mayoría de casos, AJAX puede sustituir completamente a otras técnicas como Flash. Además, en el caso de las aplicaciones web más avanzadas, pueden llegar a sustituir a las aplicaciones de escritorio.

2.5 JAVA

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java aún no lo es).

Se escogió este lenguaje debido que su popularidad se cuenta con una gran gama de manuales y tutoriales, además de que el lenguaje en si es muy potente y existen miles de clases prefabricadas para ya no reinventar la rueda.

2.6 MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB —desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y el copyright del código está en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet. MySQL AB fue fundado por David Axmark, Allan Larsson y Michael Widenius.

Se escogió este SGBD por su sencillez de uso y por ser libre.

2.7 Entorno de Desarrollo Integrado

Los IDE's (Integrated Development environment) tal y como su nombre indica son entornos de desarrollo integrados. En un mismo programa es posible escribir el código java, compilarlo y ejecutarlo sin tener que cambiar de aplicación.

NetBeans

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones

basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

NetBeans IDE

El IDE NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. Es un producto libre y gratuito sin restricciones de uso.

El NetBeans IDE es de código abierto escrito completamente en Java usando la plataforma NetBeans. Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, Web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

La versión actual es NetBeans IDE 6.5, fue lanzada el 19 de Noviembre de 2008. Extiende las características existentes del Java EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). Adicionalmente, el NetBeans Enterprise Pack soporta el desarrollo de Aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies (for BPEL), y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++, mientras el PHP Pack, soporta PHP 5.

Modularidad. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.
Sun Studio, Sun Java Studio Enterprise, y Sun Java Studio Creator de Sun Microsystems han sido todos basados en el IDE NetBeans.

Desde Julio de 2006, NetBeans IDE es licenciado bajo la Common Development and Distribution License (CDDL), una licencia basada en la Mozilla Public License (MPL).

2.8 Pantallas del sistema

La pantalla inicial figura 2.8 mostrara en la parte izquierda un recuadro en el que se puedan autenticar los usuarios ya registrados en el sistema y en la parte derecha otro recuadro para que los usuarios nuevos se registren en el sistema.

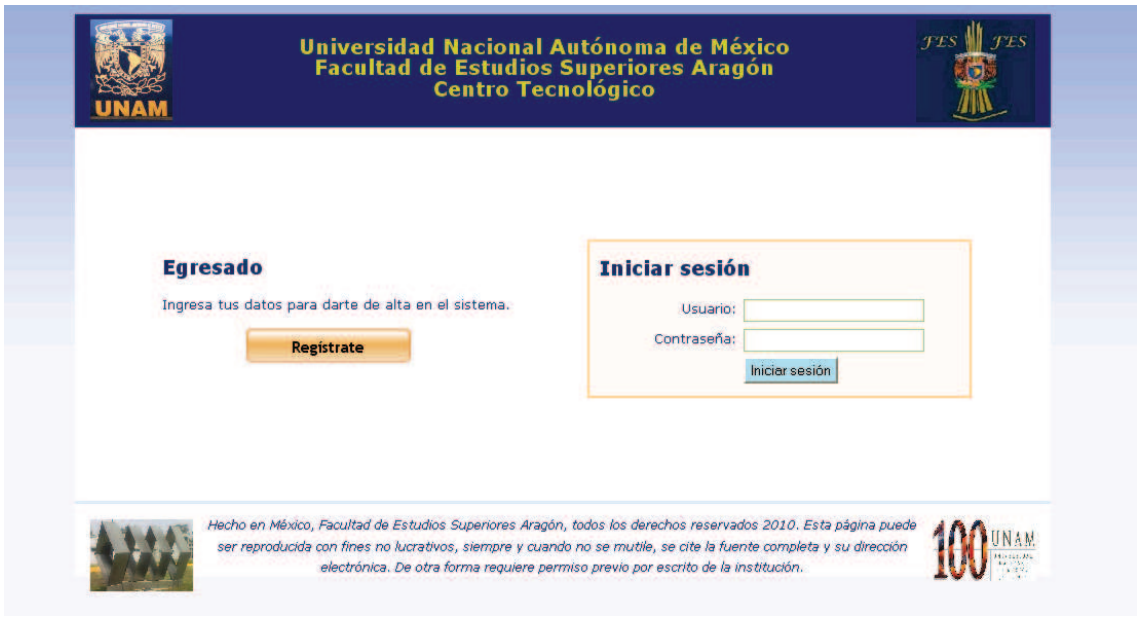

Figura 2.7 Pantalla inicial del sistema CoTER

Cuando un usuario se quiera dar de alta tendrá la siguiente pantalla figura 2.9 donde tendrá que ingresar todos sus datos.

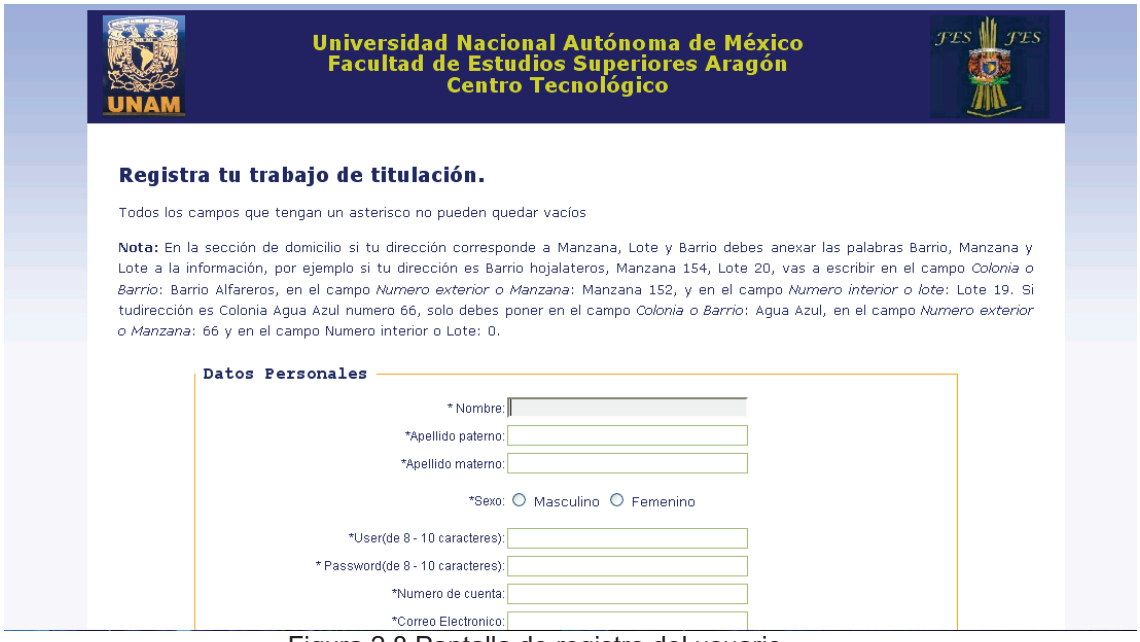

Figura 2.8 Pantalla de registro del usuario

La pantalla de la figura 2.8 depende del tipo de usuario es decir el rol que tenga asignado, ya que si es un administrador del sistema tendrá opciones como: dar de alta profesor, eliminar profesor o modificar profesor, si es un usuario alumno, tendrá las opciones de: descargar sus formatos llenados con sus datos

o modificar sus datos, finalmente si es profesor tendrá las opciones de revisar alumno o ver las estadísticas de sus alumnos que servirán como datos para reportes del catedrático.

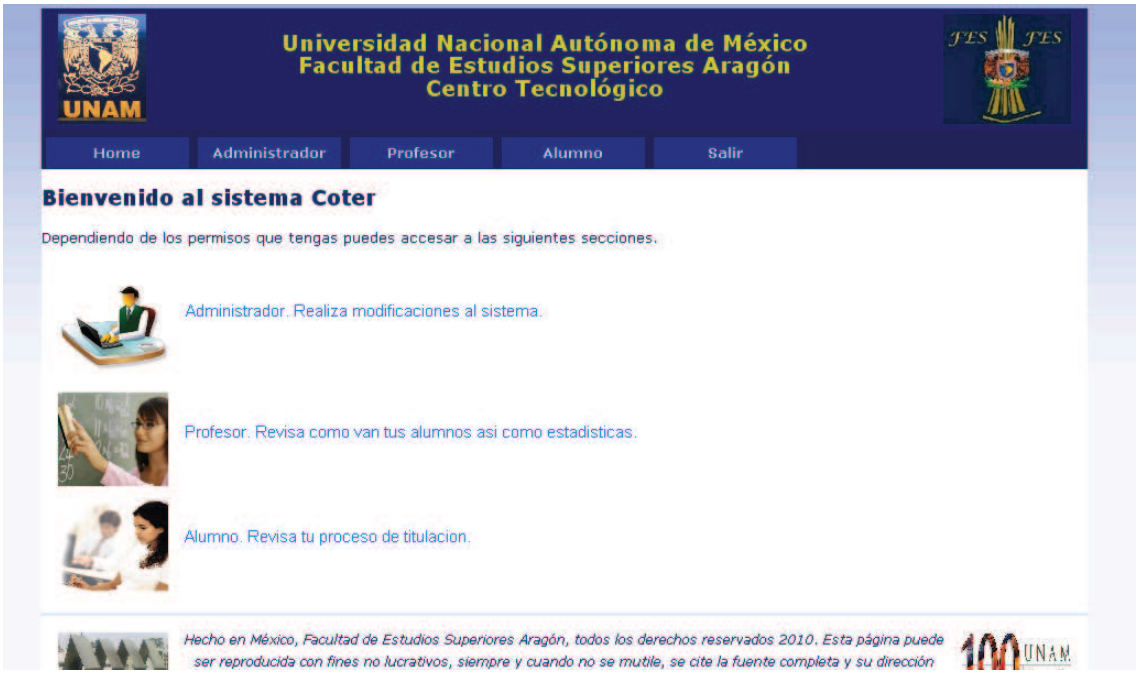

Figura 2.10 Pantalla de home del sistema CoTER

2.9 Conclusiones del capitulo

En este capítulo se pudo apreciar la importancia del capítulo anterior esto es de tener un buen análisis ya que al ir desarrollando el sistema este al tener ciertos cambios no afectaban tanto su desarrollo además de que las tecnologías en las que se implementó pueden de igual manera ser sustituidas por otras sin ningún problema.

Finalmente también se apreció que es importante conocer las distintas tecnologías con las que se pude contar al desarrollar sistemas de software ya que estas pueden facilitar el trabajo y que se acabe los proyectos en un tiempo menor.

CAPÍTULO 3

Pruebas y mantenimiento

En esta etapa se hace uso del sistema y se comprueba que lleve a cabo las funciones para las cuales fue creado.
Las pruebas se deben llevar a cabo por los distintos usuarios del sistema además se pretende conocer lo que los usuarios piensan del sistema.
Posteriormente la etapa de mantenimiento implica corrección de errores además de hacer ampliaciones o modificaciones

3.1 Pruebas de software

Las pruebas de software, en inglés *testing* son los procesos que permiten verificar y revelar la calidad de un producto software. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un programa de computadora o videojuego. Básicamente es una fase en el desarrollo de software consistente en probar las aplicaciones construidas.

Las pruebas de software se integran dentro de las diferentes fases del ciclo del software dentro de la Ingeniería de software. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir que errores tiene.

Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema.

"El testing puede probar la presencia de errores pero no la ausencia de ellos"
Edsger Dijkstra

Hay muchos planteamientos a la hora de abordar el proceso de pruebas de software, pero para verificar productos complejos de forma efectiva requiere de un proceso de investigación más que seguir un procedimiento al pie de la letra. Una definición de "testing" es: *proceso de evaluación de un producto desde un punto de vista crítico*, donde el "tester" (persona que realiza las pruebas) somete el producto a una serie de acciones inquisitivas, y el producto responde con su comportamiento como reacción. Por supuesto, nunca se debe testear el software en un entorno de producción. Es necesario testear los nuevos programas en un entorno de pruebas separado físicamente del de producción. Para crear un entorno de pruebas en una máquina independiente de la máquina de producción es necesario crear las mismas condiciones que en la máquina de producción. Existen a tal efecto varias herramientas vendidas por los mismos fabricantes de hardware (IBM, Sun, HP etc.). Esas utilidades reproducen automáticamente las bases de datos para simular un entorno de producción.

En general, los informáticos distinguen entre errores de programación (o "bugs") y defectos de forma. En un defecto de forma, el programa no realiza lo que el usuario espera. Por el contrario, un error de programación puede describirse como un fallo en la semántica de un programa de ordenador. Éste podría presentarse, o no, como un defecto de forma si se llegan a dar ciertas condiciones de cálculo.

Una práctica común es que el proceso de pruebas de un programa sea realizado por un grupo independiente de "testers" al finalizar su desarrollo y antes de sacarlo al mercado. Una práctica que viene siendo muy popular es distribuir de forma gratuita una versión no final del producto para que sean los propios consumidores los que la prueben. En ambos casos, a la versión del

producto en pruebas y que es anterior a la versión final (o "master") se denomina beta, y a dicha fase de pruebas, beta testing.

Puede además existir una versión anterior en el proceso de desarrollo llamada alpha, en la que el programa, aunque incompleto, dispone de funcionalidad básica y puede ser testeado.

Finalmente y antes de salir al mercado, es cada vez más habitual que se realice una fase de RTM testing (Release To Market), dónde se comprueba cada funcionalidad del programa completo en entornos de producción.

Otra práctica es que el proceso de pruebas se realice desde el mismo momento en que empieza el desarrollo y continúe hasta que finaliza.

3.2 La importancia de la detección oportuna

En la cadena de valor del desarrollo de un software específico, el proceso de prueba es clave a la hora de detectar errores o fallas. Conceptos como estabilidad, escalabilidad, eficiencia y seguridad se relacionan a la calidad de un producto bien desarrollado. Las aplicaciones de software han crecido en complejidad y tamaño, y por consiguiente también en costos. Hoy en día es crucial verificar y evaluar la calidad de lo construido de modo de minimizar el costo de su reparación. Mientras antes se detecte una falla, más barata es su corrección.

El proceso de prueba es un proceso técnico especializado de investigación que requiere de profesionales altamente capacitados en lenguajes de desarrollo, métodos y técnicas de pruebas y herramientas especializadas. El conocimiento que debe manejar un ingeniero de prueba es muchas veces superior al del desarrollador de software.

3.3 Prueba unitaria

En programación, una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

La idea es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto.

3.3.1 Características

Para que una prueba unitaria sea *buena* se deben cumplir los siguientes requisitos:

- Automatizable: no debería requerirse una intervención manual. Esto es especialmente útil para integración continúa.
- Completas: deben cubrir la mayor cantidad de código.

- Repetibles o Reutilizables: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para *integración continua*.
- Independientes: la ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

3.3.2 Ventajas

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

1. Fomentan el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
2. Simplifica la integración: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.
3. Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
4. Separación de la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos mock (mock object) para simular el comportamiento de objetos complejos.
5. Los errores están más acotados y son más fáciles de localizar: dado que se tienen pruebas unitarias que pueden desenmascararlos.

3.3.3 Limitaciones

Es importante darse cuenta de que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además, puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio.

3.4 Mantenimiento de software

El mantenimiento de software o manutención de software es una de las actividades más comunes en la ingeniería de software, es el proceso de mejora

y optimización del software después de su entrega al usuario final (es decir; revisión del programa), así como también corrección y prevención de los defectos.

El mantenimiento de software es también una de las fases en el ciclo de vida de desarrollo de sistemas (SDLC, sigla en inglés de *system development life cycle*), que se aplica al desarrollo de software. La fase de mantenimiento es la fase que viene después del despliegue (implementación) del software en el campo.

La fase de mantenimiento de software involucra cambios al software en orden de corregir defectos y dependencias encontradas durante su uso tanto como la adición de nueva funcionalidad para mejorar la usabilidad y aplicabilidad del software.

El mantenimiento del software involucra varias técnicas específicas. Una técnica es el rebanamiento estático, la cual es usada para identificar todo el código de programa que puede modificar alguna variable. Es generalmente útil en la refabricación del código del programa y fue específicamente útil en asegurar conformidad para el problema del año 2000.

La fase de mantenimiento de software es una parte explícita del modelo en cascada del proceso de desarrollo de software el cual fue desarrollado durante el movimiento de programación estructurada en computadores. El otro gran modelo, el Desarrollo en espiral desarrollado durante el movimiento de ingeniería de software orientada a objeto no hace una mención explícita de la fase de mantenimiento. Sin embargo, esta actividad es notable, considerando el hecho de que dos tercios del costo del tiempo de vida de un sistema de software involucran mantenimiento.

En un ambiente formal de desarrollo de software, la organización o equipo de desarrollo tendrán algún mecanismo para documentar y rastrear defectos y deficiencias. El Software tan igual como la mayoría de otros productos, es típicamente lanzado con un conjunto conocido de defectos y deficiencias. El software es lanzado con esos defectos conocidos porque la organización de desarrollo en las utilidades y el valor del software en un determinado nivel de calidad compensan el impacto de los defectos y deficiencias conocidas.

Las deficiencias conocidas son normalmente documentadas en una carta de consideraciones operacionales o notas de lanzamiento (release notes) es así que los usuarios del software serán capaces de trabajar evitando las deficiencias conocidas y conocerán cuándo el uso del software sería inadecuado para tareas específicas.

Con el lanzamiento del software (software release), otros defectos y deficiencias no documentados serán descubiertas por los usuarios del software. Tan pronto como estos defectos sean reportados a la organización de desarrollo, serán ingresados en el sistema de rastreo de defectos.

Las personas involucradas en la fase de mantenimiento de software esperan trabajar en estos defectos conocidos, ubicarlos y preparar un nuevo lanzamiento del software, conocido como un lanzamiento de mantenimiento, el cual resolverá los temas pendientes.

3.4.1 Tipos de mantenimiento

A continuación se señalan los tipos de mantenimientos existentes, definidos tal y como se especifican para la metodología de MÉTRICA:

- Perfectivo: son las acciones llevadas a cabo para mejorar la calidad interna de los sistemas en cualquiera de sus aspectos: reestructuración del código, definición más clara del sistema y optimización del rendimiento y eficiencia.
- Evolutivo: son las incorporaciones, modificaciones y eliminaciones necesarias en un producto software para cubrir la expansión o cambio en las necesidades del usuario.
- Adaptativo: son las modificaciones que afectan a los entornos en los que el sistema opera, por ejemplo, cambios de configuración del hardware, software de base, gestores de base de datos, comunicaciones, etc.
- Correctivo: son aquellos cambios precisos para corregir errores del producto software.

Cabe señalar que, de estos 4 tipos de mantenimiento, solamente el correctivo y el evolutivo entran en el ámbito de MÉTRICA versión 3, ya que los otros dos requieres actividades y perfiles distintos a los del proceso de desarrollo.

Es importante tener en cuenta el efecto del Iceberg, es decir , en el momento en el que se le hace mantenimiento a un Software no se cuenta muchas veces con el factor económico (¿Cuánto dinero se invertirá en el mantenimiento?), y una vez se comienza a desarrollar la fase de mantenimiento en la aplicación, comienzan a surgir nuevos requerimientos, el efecto del iceberg (en la superficie se ve solo una parte de lo que realmente es su tamaño).

3.5 Conclusiones del capítulo

Bueno en este capítulo se apreció que es muy importante probar los distintos módulos antes de anexarlos con los demás para que el sistema no sea inestable y luego no se conozca cuáles son las fallas.

También se apreció que por lo general un sistema de software en su etapa de mantenimiento se le añaden algunos módulos para hacerlo más accesible y amigable al usuario.

Anexo a

a. Manual de usuario

Como se mencionó anteriormente el sistema CoTER cuenta con tres usuarios el administrador, el profesor y el alumno. Se mostrara en primer lugar las acciones del administrador.

a.1 Administrador

Siempre al iniciar el sistema se muestra la pantalla de login como se ve en la figura 3.1.

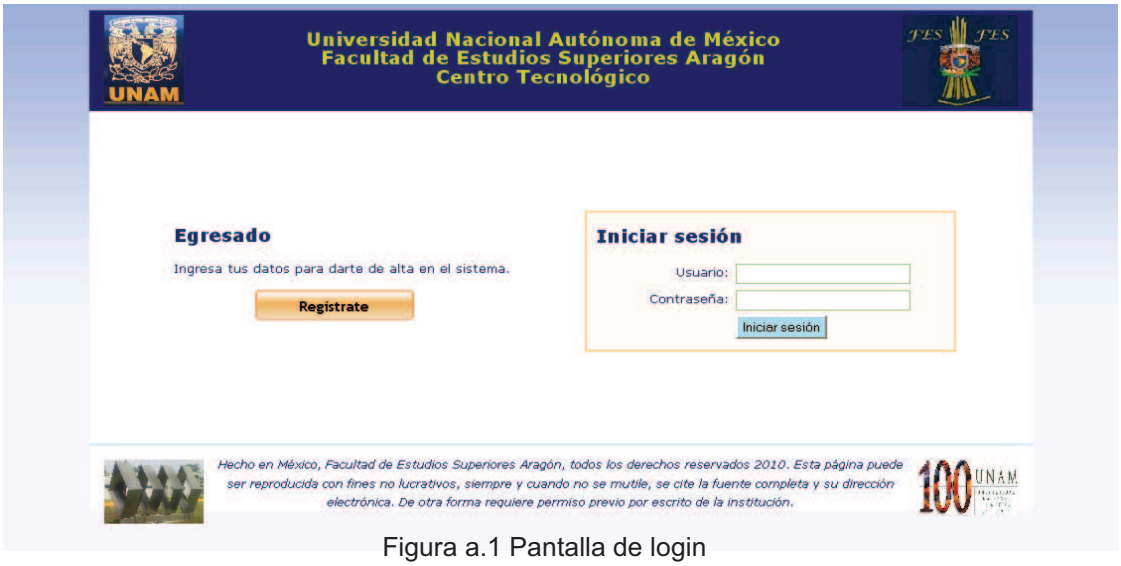

Figura a.1 Pantalla de login

Para entrar al sistema introducir usuario y contraseña y dar clic en el botón *"iniciar sesión"*. Una vez introducidos el usuario y contraseña correctos aparecerá la pantalla principal (home) como se muestra en la figura a.2 que indica que se ha autenticado con éxito.

Figura a.2 Pantalla Home

En esta pantalla de la figura a.2 se habilita el menú superior en el cual dependiendo de los permisos que tenga el usuario autenticado podrá acceder o no a los distintos enlaces, en este caso el administrador puede sólo accesar al enlace home, administrador y salir. El enlace *"home"* siempre lo traerá de nuevo a esta pantalla. El enlace *"salir"* lo saca del sistema y lo lleva a la pantalla de login. El enlace *"administrador"* al igual que el enlace que se encuentra en la sección de contenido donde está una imagen y adelante el texto administrador lleva a la página del administrador mostrada a continuación.

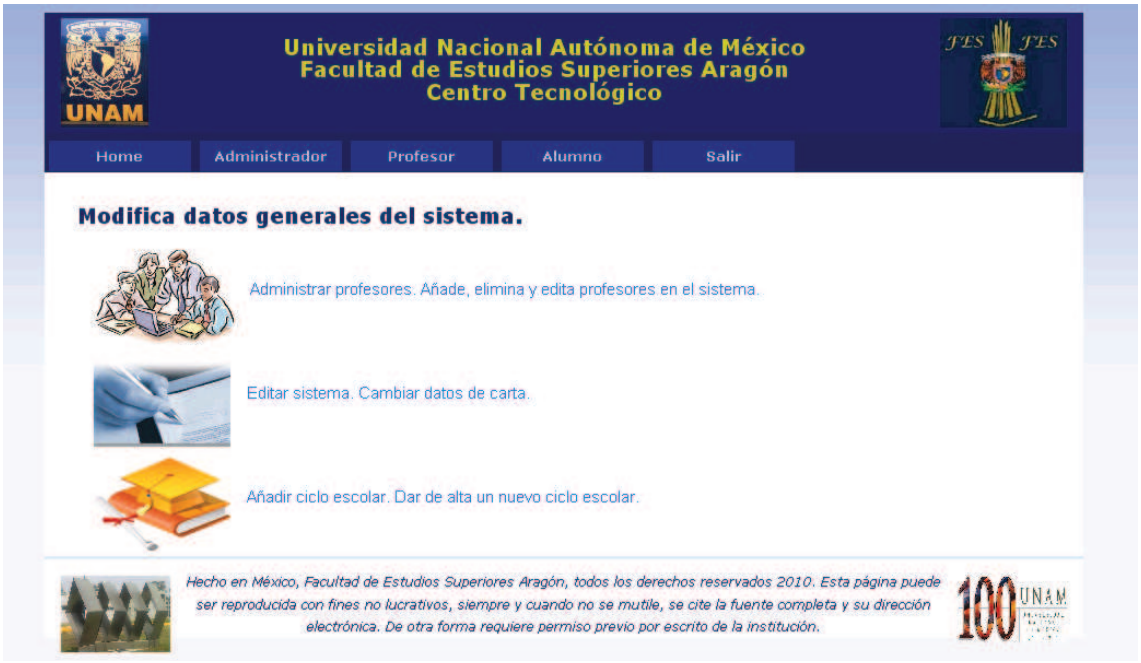

Figura a.3 Pantalla de administrador

Añadir profesores

En la pantalla de la figura a.3 en la sección de contenido se cuenta con tres imágenes que son enlaces a las distintas acciones que puede hacer un administrador. Para añadir, editar o eliminar un profesor de clic en el enlace *"administrar profesores"*. Se carga la página de la figura a.4.

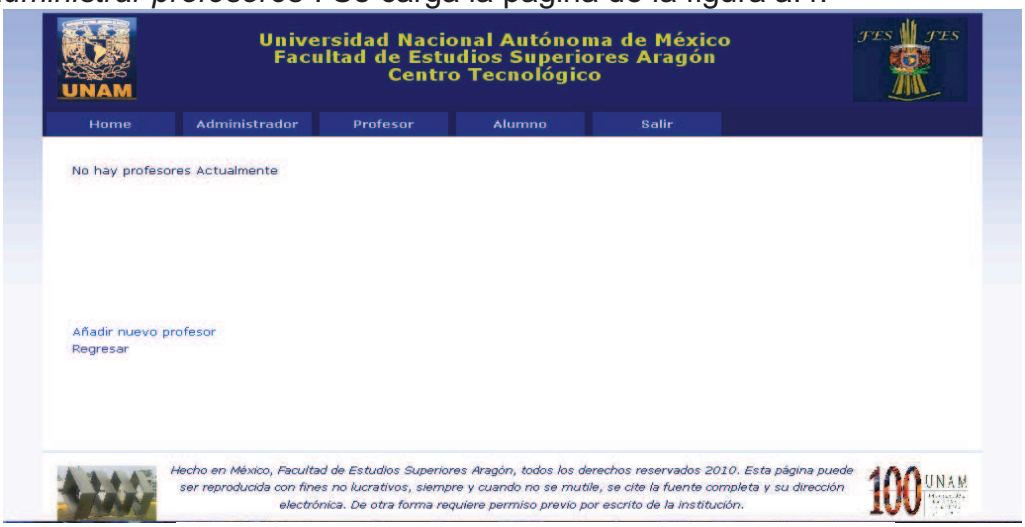

Figura a.4 Pantalla de administrar profesores

Como se puede apreciar en la figura a.4 no hay ningún profesor en el sistema, para agregar un nuevo profesor dar un clic en el enlace *"añadir nuevo profesor"*, se carga la pagina mostrada en la figura a.5.

Figura a.5 Pantalla de agregar profesor

Todos los campos son indispensables para poder dar de alta un profesor, por lo que el sistema mostrara un mensaje de datos faltantes al haber omitido cualquiera de ellos, además también emitirá un mensaje si los datos no son los solicitados como lo ilustra la figura a.6.

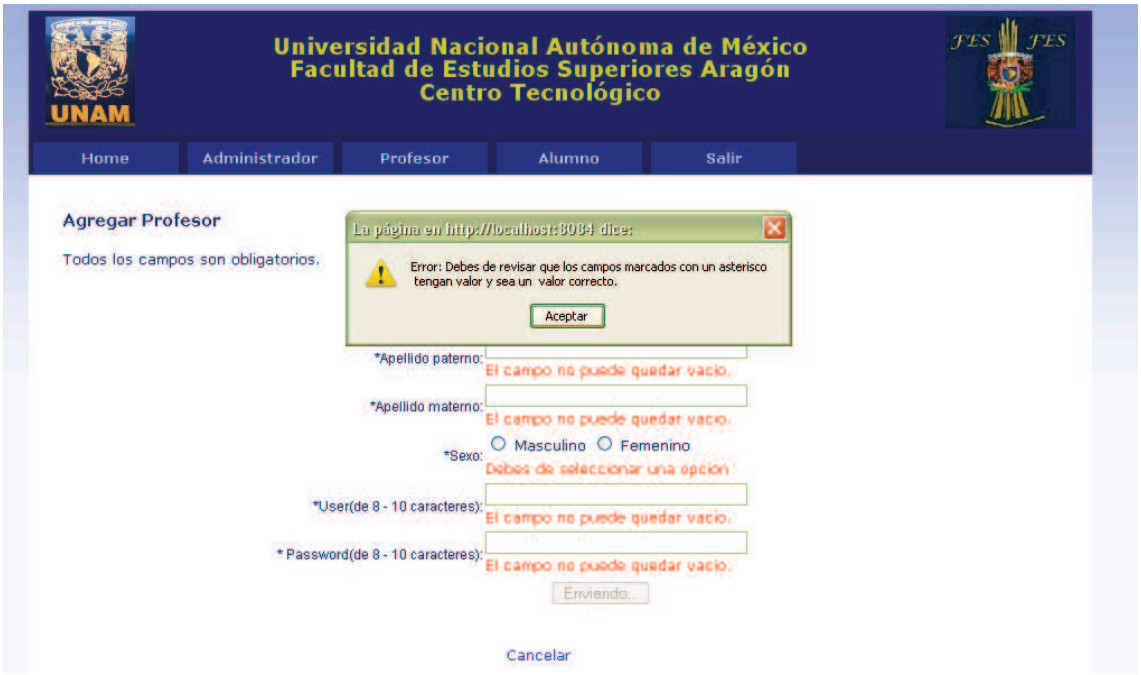

Figura a.4 Mensaje de datos omitidos o no válidos

Si no se omitió ningún dato oprima el botón aceptar, el sistema le regresara a la pantalla anterior listando al nuevo profesor. Si no desea guardar nada dar clic en el enlace “Cancelar”, el sistema lo redirigirá a la pantalla anterior.

Editar profesores

Una vez autenticado y situado en la pantalla de administrar profesores, aparece un listado con los profesores en el sistema, esta lista se encuentra en modo editable para hacer los cambios pertinentes como se muestra en la figura a.7.

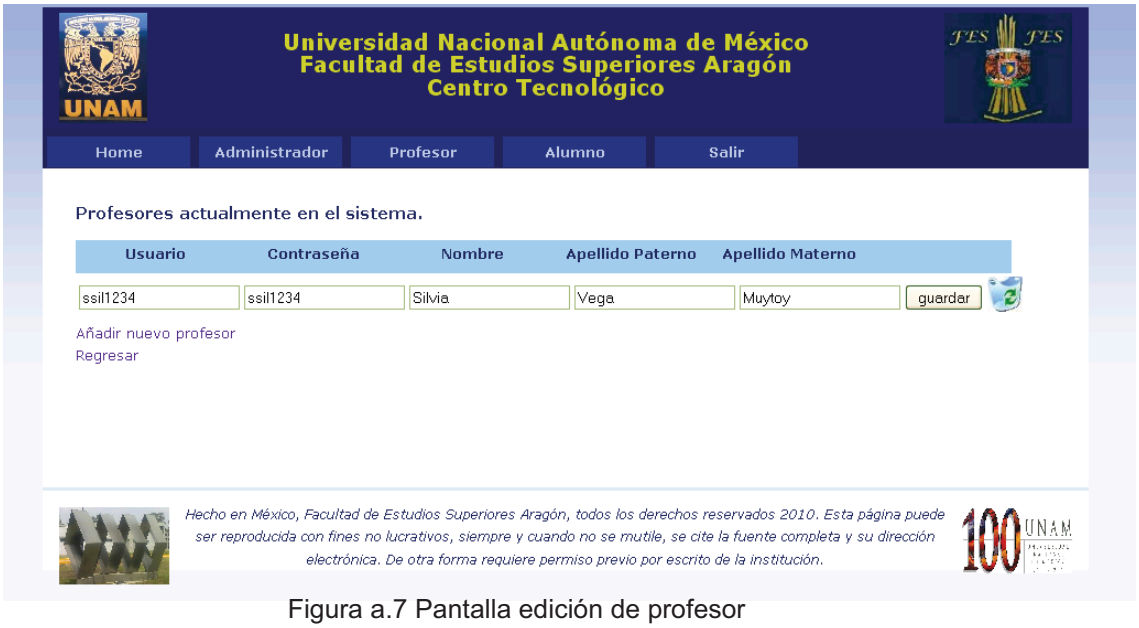

Figura a.7 Pantalla edición de profesor

Para guardar los cambios de clic en el botón guardar, aparecerá un cuadro de diálogo notificándole que los cambios se han hecho, el sistema actualizara la página con los nuevo cambios.

Eliminar profesores.

Una vez autenticado y situado en la pantalla de administrar profesores, al lado de cada registro de la lista de profesores aparece una imagen de una papelera de reciclaje, para eliminar al profesor de clic en esta imagen, el sistema muestra un mensaje de confirmación, dar clic en el botón aceptar para eliminar o cancelar.

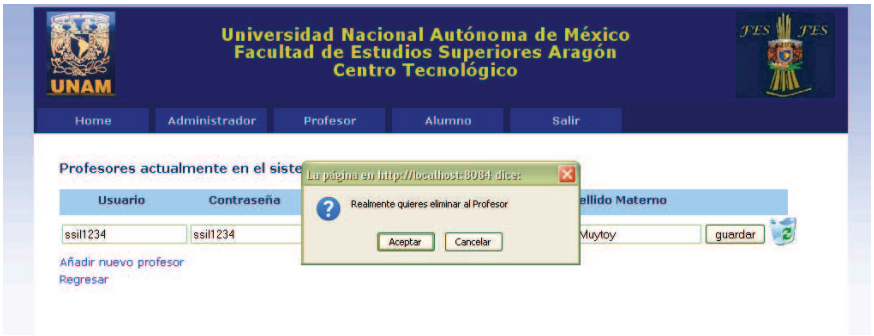

Figura a.8 cuadro de confirmación de eliminar profesor

Modificar valores de carta.

Una vez autenticado y situado en la pantalla de administrador dar clic en la sección de *"Editar sistema"*, aparece la pantalla de la figura a.9.

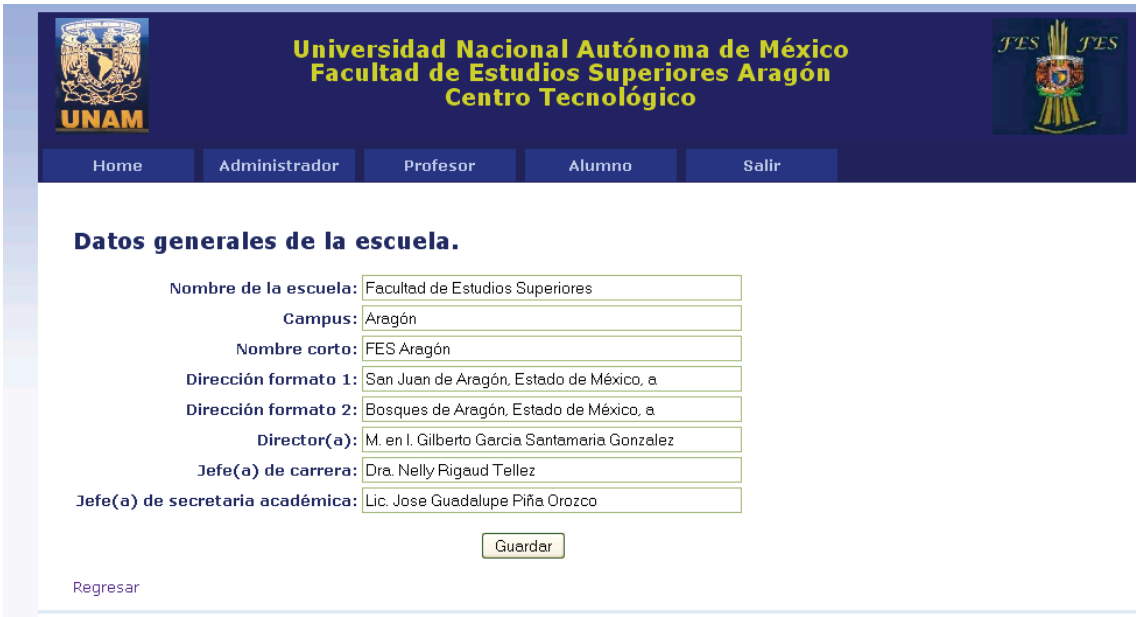

Figura a.9 pantalla de edición de valores de carta.

Los valores en los campos de texto son editables, modifique los valores y de clic en el botón guardar, los valores de los campos se actualizarán con el nuevo valor.

Añadir ciclo escolar.

Una vez autenticado y situado en la pantalla de administrador de clic en la sección de *"Nuevo ciclo escolar"*, aparece la pantalla de la figura a.10.

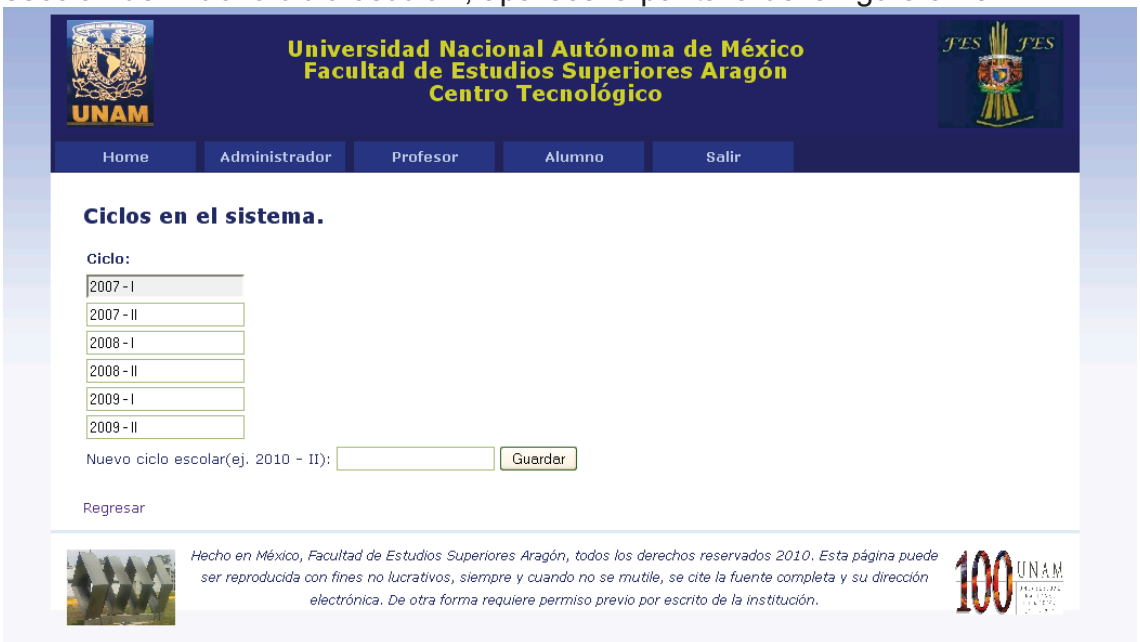

Figura a.10 Pantalla de añadir nuevo ciclo escolar

La pantalla muestra una lista de los ciclos en el sistema, y debajo de estos la opción de agregar uno nuevo, delante de la leyenda *"Nuevo ciclo escolar (ej. 2010 - II)",* situado en el cuadro de texto introduzca el nuevo ciclo escolar y dar clic en el botón guardar, el sistema agregara el nuevo ciclo y lo mostrara en la lista.

a.2 Alumno

Registrarse por primera vez

En la pantalla de login (figura a.1) de clic en el botón *"Regístrate"* , aparecerá la ventana de la figura a.11.

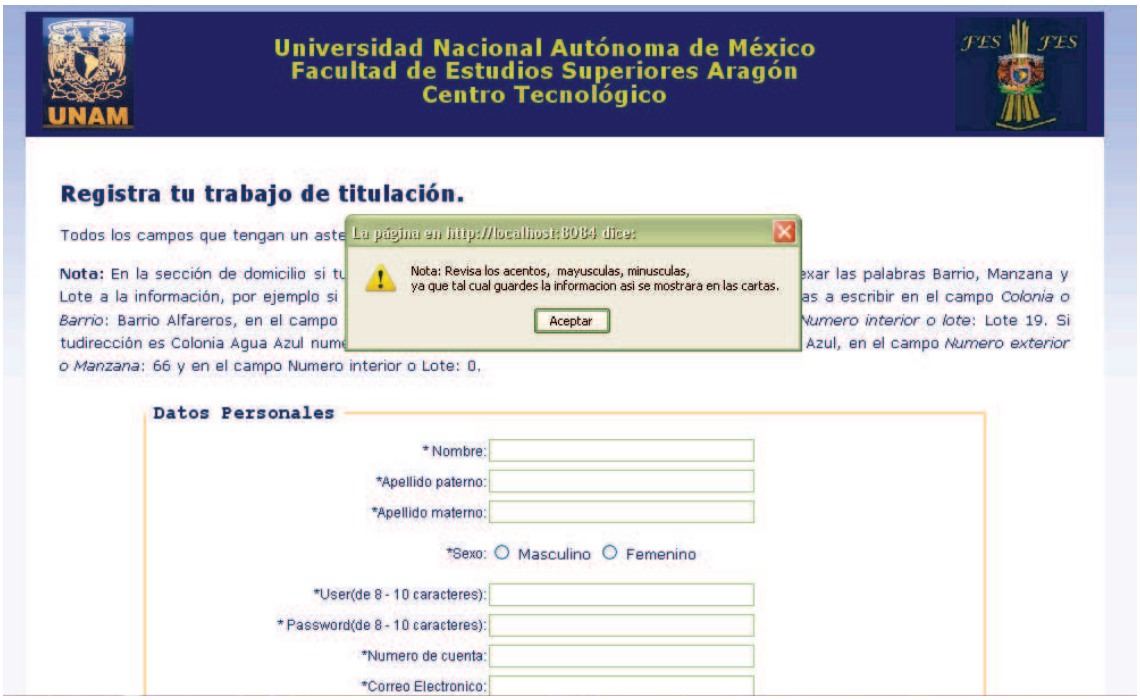

Figura a.11 Pantalla de Registrarse por primera vez.

El sistema le mostrara mensaje a tener en cuenta de clic en aceptar y proceda a llenar los campos de acuerdo a las instrucciones. Para guardar su registro dar clic en el botón *"Acepto",* Si un campo es omitido o el valor no es correcto el sistema le mostrara un mensaje indicándole que los datos no son correctos además de indicarle exactamente cuál es el campo con el valor incorrecto. Si los datos están bien el sistema le mostrara la figura a.12 indicándole que el registro fue exitoso. El alumno con el usuario y contraseña que proporciono ya puede entrar al sistema. Además el profesor ya tiene asignado este alumno. Para ir a la pantalla de login dar clic en el enlace *"Ir a página de inicio".*

Figura a.12 Pantalla de registro exitoso

Editar datos

Una vez autenticado y situado en la pantalla de home vaya a la sección de alumno dar clic en la correspondiente imagen, el sistema le mostrara la pantalla de la figura a.13.

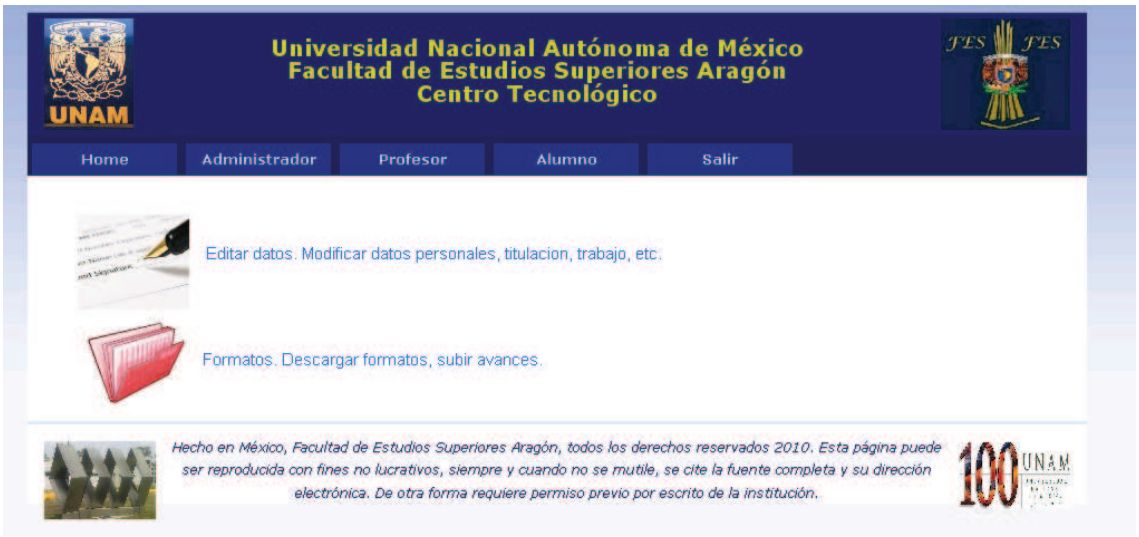

Figura a.13 Pantalla de alumno

En la sección de contenido se encuentran las acciones del usuario. Dar clic en la imagen correspondiente a *"Editar datos"*, El sistema le mostrara la pantalla de la figura a.14.

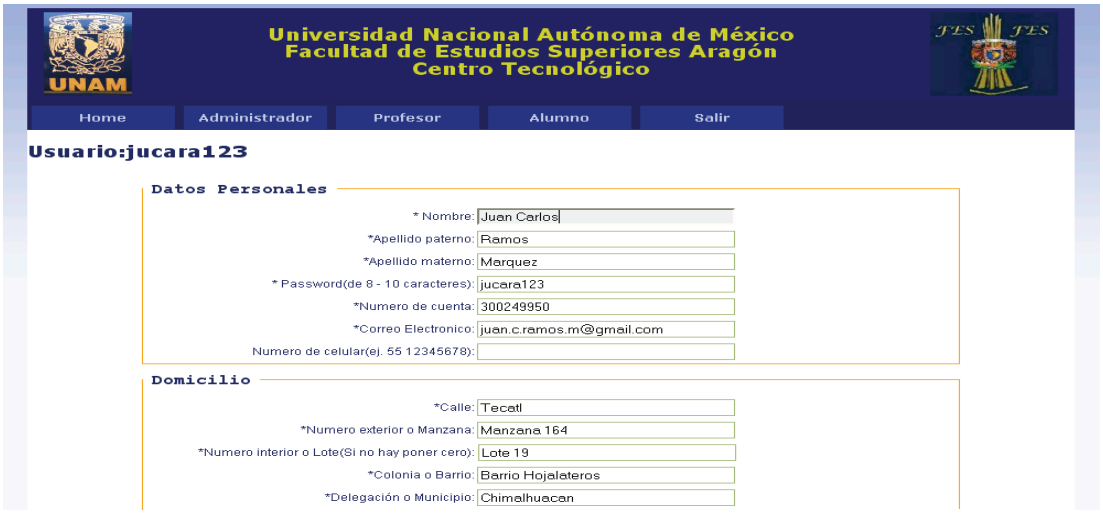

Figura a.14 Pantalla de edición de datos.

El sistema le cargara todos sus datos en modo editable, hacer los ajustes necesarios y de clic en el botón *"Acepto"*, si los datos están bien el sistema le actualizara los datos, de lo contrario le mostrar un mensaje indicando los errores.

Descargar cartas y subir avances

Una vez autenticado y situado en la pantalla de alumno vaya a la sección de formatos y dar clic en la imagen correspondiente. El sistema le cargara la pantalla mostrada en la figura a.15.

Figura a.15 Pantalla de formatos

En la sección de contenido habrá dos subsecciones, la de cartas y la de reportes, para descargar una carta ir a la sección de cartas y dar clic en el enlace a la carta que se quiera, el navegador mostrara la respectiva carta en formato PDF. Para subir un avance en la sección de reportes, llenar los campos de título, y comentarios y dar clic en el botón *"examinar"* para situarse en el archivo que se quiera subir, posteriormente dar clic en el botón "*enviar",* el sistema actualizara esta pantalla y mostrara el reporte que se subió como se puede ver en la pantalla de la figura a.16.

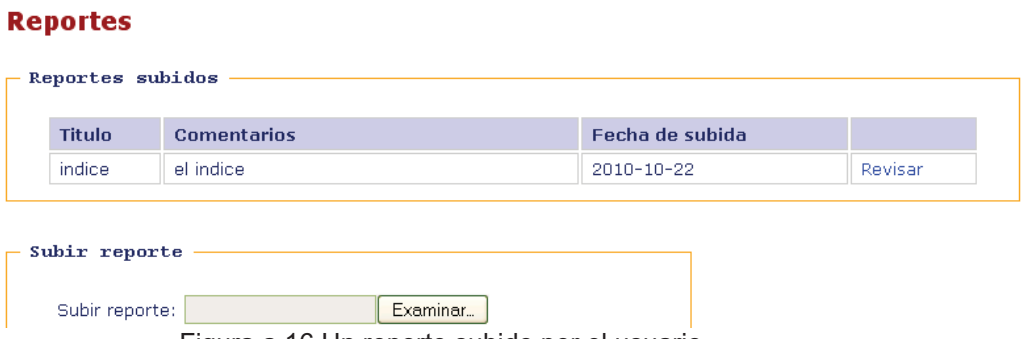

Figura a.16 Un reporte subido por el usuario.

a.3 Profesor

Revisar alumnos

Una vez autenticado y situado en la pantalla de home ir a la sección de profesor y dar clic en la correspondiente imagen, el sistema le mostrara la pantalla de la figura a.17.

Figura a.17 Pantalla de Profesor

En la sección de contenido se encuentran las acciones del Profesor. Dar clic en la imagen correspondiente a *"Revisar alumnos"*, El sistema le mostrara la pantalla de la figura a.18.

Figura a.18 Lista de alumnos

En la sección de contenido se verán los alumnos que se tiene asignados, para revisar uno en particular dar clic en el enlace *"Revisar"*, el sistema le mostrara la ventana de la figura a.19.

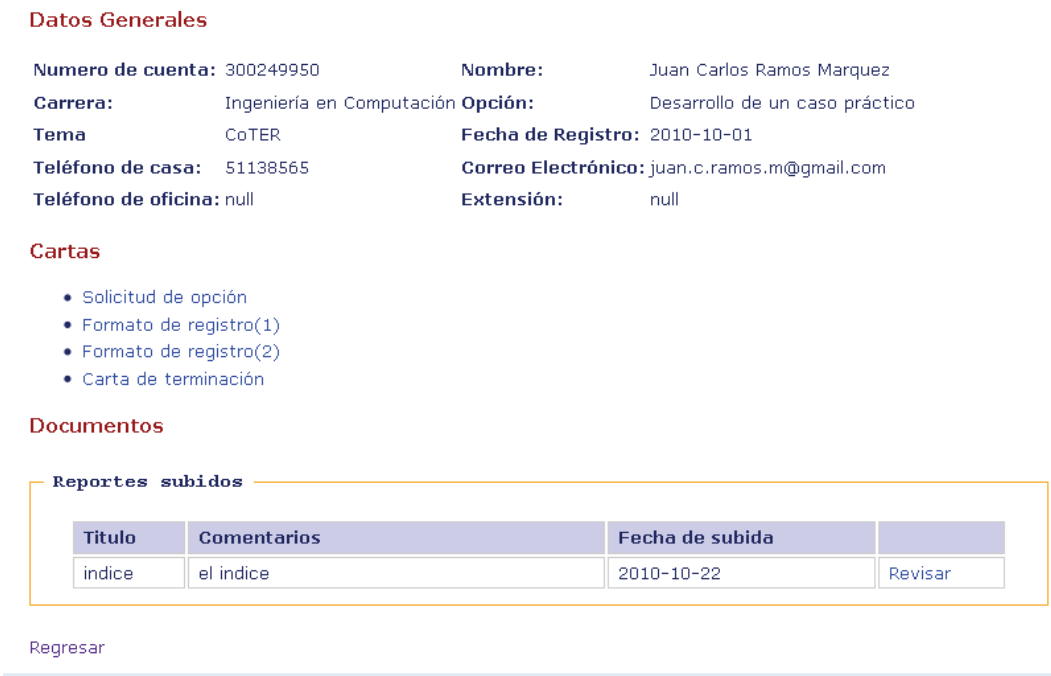

Figura a.19 Pantalla revisión de alumno

En la sección de contenido hay 3 subsecciones, datos generales, cartas y documentos. En la sección de datos generales el sistema muestra la información puntal del alumno como número de cuenta, nombre, carrera, tema, etc. En la sección de cartas hay enlaces a las respectivas cartas del alumno, para revisar dar clic en el enlace correspondiente, el sistema le mostrara la carta en formato PDF. Finalmente en la sección de documentos se pueden ver los archivos que el alumno ha enviado, para revisar uno de clic en el enlace *"revisar"*.

Ver estadísticas

Una vez autenticado y situado en la pantalla de Profesor ir a la sección de estadísticas y dar clic en la correspondiente imagen, el sistema mostrara la pantalla de la figura a.20.

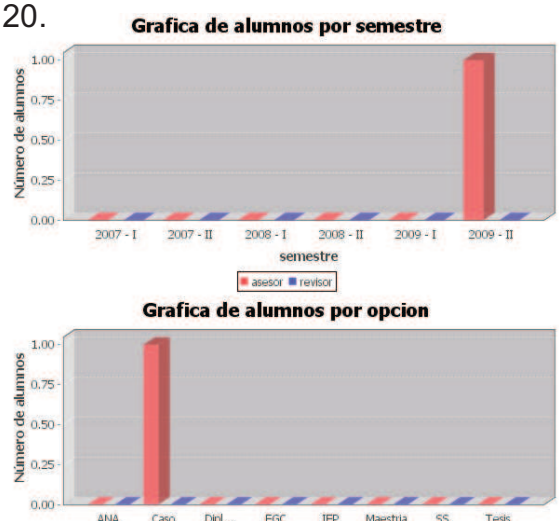

Figura a.20 Estadisticas del sistema

En la sección de contenido se pueden ver las gráficas correspondientes así como información pertinente.

a.4 Salir del sistema

Para salir del sistema no importando cual fuere el usuario, se debe dar un clic en la opción salir del menú superior.

Como se pude apreciar el uso del sistema es intuitivo, por lo que este manual es de es de lectura rápida además muestra un uso completo y sencillo del sistema.

CONCLUSIONES

El sistema se desarrolló bajo metodologías de construcción de software así como con herramientas de diseño de base de datos, herramientas de modelado, entornos de desarrollo integrado, los cuales dan un gran soporte en la construcción de estos proyectos, por lo cual es importante conocer este tipo de herramientas para hacer más eficiente el trabajo.

Durante la creación del sistema pude ver como es muy importante tener en cuenta cada etapa y desarrollarla adecuadamente al problema para que los cambios al sistema no le afecten mucho.

La principal problemática que encontré fue que se deben conocer bien las tecnologías en las que se va a implantar el sistema es decir conocer sus alcances y limitaciones ya que durante el desarrollo pude que no satisfagan todas las necesidades y también hay que conocer una amplia gama de tecnologías ya que algunas aceleran más el proceso de desarrollo que otras.

También puse en práctica los conocimientos adquiridos durante mi formación así como los adquiridos en la práctica profesional. Comprobé como se juntan varias ramas de la carrera: programación orientada a objetos, ingeniería de software, bases de datos para solucionar un problema de la vida real.

Esta primera versión del sistema satisface los requerimientos del usuario pero aún se puede ampliar más el sistema y cargar más módulos.

BIBLIOGRAFÍA

- Deitel, H.M. (2004). *Java como programar* (Quinta ed.). México D.F; Prentice Hall
- Eguiluz, Javier (2008). *Introducción a JavaScript.* México, D.F. Librosweb
- Eguiluz, Javier (2008). *Introducción a XHTML.* México, D.F. Librosweb
- Maxwell, Jhon (2007). *AJAX un juego de niños.* México D.F. PC – Cuadernos.
- Muller, Peter (2006). *CSS un juego de niños.* México D.F. PC – Cuadernos.
- Quijano, Andrés (2006) *Programación Web java.* Buenos Aires, Argentina. MP Ediciones.
- Schmuller, Joseph (2005). *Aprendiendo UML en 24 horas.* Estado de México. Prentice Hall.