Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Caracas – Venezuela

Trabajo Especial de Grado

Desarrollo de la Aplicación de Gestión Académica CONEST 2.0: Módulo Administrativo

Trabajo Especial de Grado

presentado ante la ilustre

Universidad Central de Venezuela

Por los Bachilleres

Belfort Urquiola Pablo José
C.I. 16224834
Ojeda Morales Lucía Verushka
C.I. 17389408

para optar por el título de
Licenciado en Computación

Tutores:
Prof. Rivas Sergio
Prof. Zambrano Jossie
Caracas, Octubre 2009

ACTA

Quienes suscriben, miembros del Jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los Bachilleres Lucía Verushka Ojeda Morales de C.I. 17.389.408 y Pablo José Belfort Urquiola de C.I. 16.224.834, titulado: **"Desarrollo de la Aplicación de Gestión Académica CONEST 2.0: Módulo Administrativo"**, a los fines de optar por el título de Licenciado en Computación, dejen constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del jurado, se fijó el día 29 de Octubre de 2009 a las 9:30 am en la Sala 1 de la Escuela de Computación, para que sus autores lo defendieran en forma pública, mediante una presentación oral de su contenido, luego de lo cual respondieron las preguntas formuladas. Finalizadas la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con una nota de _____ puntos.

En fé de lo cual se levanta la presente Acta, en Caracas los 29 días del mes de Octubre del año dos mil nueve dejándose también constancia de que actuó como Coordinadora del Jurado la Profesora Jossie Zambrano.

_____	_____
Prof. Jossie Zambrano (Tutor)	Prof. Sergio Rivas (Tutor)
_____	_____
Prof. Edgar González (Jurado)	Prof. Andrés Sanoja (Jurado)

Agradecimientos y Dedicatoria

A nuestros padres por su apoyo incondicional, consejos y palabras de aliento en todo momento de nuestras vidas. Por ser ejemplo y modelo a seguir para nosotros y alentarnos a ser cada vez mejores personas.

A nuestras familias por enseñarnos el significado del cariño, unión, comprensión y compromiso. Gracias a ellas, tenemos las bases necesarias para lograr cada meta que nos propongamos.

A mi sobrina Cristina, por regalarme cariño y alegría en momentos en los que el cansancio y el estrés comenzaban a agobiarme.

A Néstor, por ser esa persona especial que esperaba y necesitaba, por estar presente y pendiente de mí en cada momento, escucharme, apoyarme y ayudarme a seguir adelante.

A nuestros tutores Jossie y Sergio por ser nuestros guías, por ser consecuentes y estar siempre disponibles para brindarnos ayuda y apoyo en cada paso que dimos en este trabajo. A Jossie, por tener la visión de que podríamos ser un buen equipo y por hacer lo posible para conformarlo. A Sergio, por ayudarnos siempre con una sonrisa, en esos momentos cuando creíamos que algo no tenía solución.

A nuestros amigos y compañeros que nos apoyaron y alentaron en cada momento, siendo una vía de escape y despeje cada vez que lo necesitamos.

A Lucía, por ser una gran amiga y compañera de equipo. Por poner el empeño necesario para completar este trabajo y sobre todo, por recordarme todas esas cosas que mi mala memoria dejaba pasar por alto.

A Pablo, por estar ahí en todo momento, por conocerme y aguantarme durante todo este tiempo, por ser un gran compañero y también amigo. Por mantener la calma y ayudarme a tenerla en esos momentos que no creía poder con ella.

A la División de Control de Estudios, por su colaboración y participación constante durante el desarrollo de este trabajo.

A nuestra universidad, facultad y escuela, por permitirnos ser parte de ella y por ser testigo de nuestro crecimiento como profesionales.

A todos les agradecemos y dedicamos este Trabajo Especial de Grado, por ser parte de este, de nuestras vidas y de nuestro futuro.

Con mucho cariño,

Lucía y Pablo

Resumen

El objetivo de este Trabajo Especial de Grado consiste en generar la nueva versión del sistema CONEST[1], basándose en la experiencia del desarrollo de la versión anterior, realizando migraciones de tecnología y datos, además de realizar mejoras correspondientes a las entidades, procesos, gestiones administrativas, interfaces, entre otras actividades necesarias para la evolución del Software. Asimismo brindará un mejor soporte al personal involucrado, aumentando su satisfacción al usar nuevas tecnologías relacionadas con la web 2.0, que ofrecen beneficios tales como, accesibilidad para el usuario, usabilidad del sistema y personalización de la aplicación. En esta investigación se realiza un estudio de los modelos de datos, comparando el existente y el propuesto. El método de desarrollo de software utilizado es Programación Extrema (XP). Como producto final se obtienen las funcionalidades del Módulo de Administración del Sistema CONEST 2.0, relacionadas a la gestión administrativa de los estudiantes y docentes de la Facultad de Ciencias.

Palabras clave

CONEST, CONEST 2.0, Modelo Vista Controlador (MVC), Ruby on Rails, Rails 2.3, Migración de datos, Migración de tecnologías, División de Control de Estudios, Web 2.0, Programación Extrema (XP).

[1] CONEST (Control de Estudio) Sistema de Gestión Académica de la División de Control de Estudio de la Facultad de Ciencias de la Universidad Central de Venezuela

Índice de Contenido

Índice de Figuras

Índice de Tablas

Introducción

Internet ha tenido gran auge en las últimas décadas, modificado nuestro estilo de vida, lo cual ha incrementado el desarrollo de aplicaciones web, que ofrecen numerosos beneficios ya que pueden ser accedidos desde cualquier computador conectado a Internet. Todo esto ha provocado que empresas y organizaciones sustituyan sus programas de escritorio por aplicaciones web.

Una de las organizaciones que se ha sumado a la implantación de aplicaciones web es la División de Control de Estudios (DCE) de la Facultad de Ciencias de la Universidad Central de Venezuela, mediante el Sistema de Gestión Académica CONEST. La primera versión aparece aproximadamente en septiembre de 2006, automatizando sólo los procesos de calificación e inscripción. A partir de ese momento, gracias al aporte de docentes, estudiantes y personal administrativo, este sistema ha crecido, buscando automatizar la mayor cantidad de procesos posibles. Sin embargo, la falta de planificación y estándares al momento del desarrollo, han generado inconsistencias, redundancia en la interfaz, base de datos y código, aumentando la cantidad de tiempo y recursos al momento de hacer mantenimiento o cambios en el sistema.

Sumado a esto, la web y las tecnologías utilizadas en CONEST han evolucionado considerablemente. Estos avances han traído diferentes ventajas, las cuales no se han incorporado a este sistema.

El objetivo de este Trabajo Especial de Grado es el desarrollo de una nueva versión del Sistema de Gestión Académica CONEST, denominada CONEST 2.0, tomando como referencia la versión actual, buscando así, reducir los problemas existentes y aprovechar las ventajas de las nuevas tecnologías utilizadas durante su creación, además de contribuir a la disminución significativa de tiempo, dinero y esfuerzo al personal involucrado.

El presente documento se encuentra estructurado de la siguiente manera:

En el Capítulo I se describe el Problema de Investigación donde se presentan y explican las diferentes dificultades existentes en CONEST, así como el objetivo general y los objetivos específicos de este Trabajo Especial de Grado, su justificación y alcance.

El Capítulo II presenta el Marco Conceptual donde se muestran las bases en las que se fundamenta el desarrollo de este trabajo. Se describen las aplicaciones Web Cliente/Servidor, su funcionamiento, arquitectura y características principales. Se presentan las herramientas tecnológicas de software libre utilizadas para realizar esta aplicación, explicando las tecnologías del lado del cliente y del servidor necesarias para el desarrollo de software web.

El Capítulo III está constituido por el Marco Aplicativo, donde se presenta la adaptación del método Programación Extrema (XP) al problema de investigación, detallando las fases de Planificación, Diseño, Codificación y Pruebas dentro de cada una de las iteraciones definidas para dicho proceso.

Finalmente se presentan los anexos, las conclusiones y recomendaciones, y referencias bibliográficas utilizadas durante el desarrollo de este trabajo.

Capítulo I Problema de Investigación

El Sistema de Gestión Académica de Control de Estudios CONEST, es un sistema que automatiza las actividades administrativas referente a los estudios de pregrado que ofrece la Facultad de Ciencias de la Universidad Central de Venezuela. La División de Control de Estudios (DCE) es el ente encargado de gestionar los datos de los estudiantes, docentes y personal administrativo de la facultad.

Debido al crecimiento constante del Sistema CONEST, la evolución de las tecnologías web y la falta de planificación y estándares, este sistema presenta ciertas dificultades como, redundancias en la interfaz, inconsistencias en la base de datos y repetición de código, generando un incremento de esfuerzo en mantenimiento y soporte de la aplicación. La necesidad de disminuir o eliminar estos problemas es la principal motivación de este Trabajo Especial de Grado (TEG).

En este capítulo se describe el contexto del problema, luego se expone el objetivo general y los objetivos específicos de esta investigación, así como la justificación de realizar los cambios necesarios al sistema CONEST.

1. Planteamiento del Problema

El Sistema de Gestión Académica de Control de Estudios CONEST, es un sistema que ha sido desarrollado por la comunidad de la Escuela de Computación (alumnos y profesores), a través de Laboratorios, Pasantías, Seminarios y Trabajos Especiales de Grado, con la finalidad de incrementar y mejorar los servicios que brinda la División de Control de Estudios (DCE) a la comunidad de la Facultad de Ciencias compuesta por las licenciaturas de Biología, Computación, Física, Geoquímica, Matemática y Química que suman aproximadamente 3500 usuarios.

Esta aplicación automatiza los procesos de preinscripción de materias electivas y laboratorios; inscripciones; administración de la planta física; calificaciones; emisión de constancias y reportes; auditoría, registro y control de egreso; reincorporación por reglamento de permanencia; y tramitación de solicitudes estudiantiles. Todo esto mediante un conjunto de módulos que funcionan de manera integrada y están orientados a prestar servicios a sus respectivos usuarios, manteniendo una comunicación, mediante el envío de información a través de correos electrónicos sobre el estado de sus solicitudes e información de interés para ellos.

CONEST es una aplicación Web, presta servicios las 24 horas del día, los 365 días del año, en los cuales se puede acceder a todas sus funcionalidades de manera remota sin la instalación de ningún tipo de software especial, simplemente usando un navegador web como Mozilla Firefox, Internet Explorer, Opera, Safari, entre otros.

CONEST es una aplicación desarrollada con software libre[2]. La implementación del sistema fue realizada con el marco de trabajo Ruby on Rails siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador), brindando una independencia en los componentes que lo conforman, generando una mejor

[2] Software que brinda libertad a los usuarios sobre un producto y por tanto, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. (Seguridad en Internet, 2009).

organización del trabajo. Para garantizar la comunicación entre los equipos de trabajo, se cuenta con la herramienta Subversión (SVN), que ayuda a establecer sincronizaciones para la integración del código de acuerdo a las necesidades de cada grupo que interviene en éste proceso de desarrollo.

En cuanto a la base de datos, CONEST utiliza el Sistema Manejador de Bases de Datos MySQL, un repositorio, flexible y de alto rendimiento, lo que a futuro permite que la aplicación pueda ser modificada y adaptada para otros contextos académicos.

1.1. Problemas de CONEST

Debido a la constante evolución de las Tecnologías de Información y Comunicación, es necesario evaluar y actualizar las tecnologías inherentes al desarrollo de aplicaciones Web, tales como CONEST, que a pesar de cumplir con muchos de sus objetivos, presenta diferentes problemas como, dificultades con la integración de nuevos módulos, alto costo (tiempo y recurso humano) en mantenimiento, ausencia de estándares de programación, lo que conlleva a poca organización en el código, haciendo difícil su comprensión y la ubicación del origen de alguna falla específica, influyendo directamente en la eficacia y eficiencia del sistema. Algunos de los problemas existentes en CONEST serán listados y explicados a continuación:

1.1.1. Interfaz de Usuario

Dentro de los aspectos importantes para tomar en cuenta en la evaluación y actualización de CONEST, se encuentran las interfaces de usuario, ya que actualmente no están enfocadas hacia interfaces enriquecidas que propone la web 2.0, generando una alta carga cognitiva para el usuario, ya que la organización y la forma de mostrar las diferentes funcionalidades no es la más apropiada. Esta situación se evidencia en la Figura 1 presentada a continuación:

Figura 1 Página Principal Módulo Administrativo

Los enlaces son mostrados en una lista muy larga (más de 240 enlaces) y, que a pesar de estar ordenados por categorías y alfabéticamente, se pueden encontrar duplicados y hasta triplicados, lo que genera pérdida de tiempo en la búsqueda del enlace requerido por el usuario.

Existe redundancia en la interfaz, generando confusión en el usuario al tener nombres no coherentes con la funcionalidad que tiene en el sistema, en la Figura 2 se muestra un ejemplo de este problema.

Figura 2 Ejemplo de enlaces con diferentes nombres pero igual función

Al no existir una herramienta adecuada para la comunicación entre el grupo de programadores, se tiene como resultado poca consistencia en la interfaz y en los componentes que conforman la aplicación (Ver Figura 3).

Figura 3 Ejemplo de inconsistencias

No hay estandarización para realizar las búsquedas dentro de la aplicación, por ejemplo, se tienen diferentes formas de búsqueda a un usuario, aunque la instrucción sugiere indicar la cédula del estudiante, se puede buscar bien sea por número de cédula, nombre o apellido, teniendo como consecuencia confusiones de parte del usuario al no saber qué datos debe suministrar.

1.1.2. Base de Datos

En lo que a la base de datos se refiere, CONEST carece de estandarización de los atributos en las tablas, complicando tanto el desarrollo de la aplicación,

como las consultas en la base de datos.

Actualmente la nomenclatura utilizada en la base de datos (tablas y sus atributos) es muy específica a la situación actual de la Facultad de Ciencias (Pregrado), haciendo poco flexible la implementación del sistema en otros controles de estudios como el de Postgrado.

1.1.3. Programación

En el aspecto de programación, CONEST ha sido desarrollado por un grupo numeroso de personas (aproximadamente 50) con distintas formas de programar, y no existe un estándar formal de código, haciendo difícil su comprensión y modificación, lo que genera altos costos en tiempo y recursos en mantenimiento.

No existe un método de desarrollo de software global en la aplicación, lo que genera inconsistencias en la implementación de las funcionalidades del sistema, fomentando reuniones para la integración del producto de software.

No se cumplen algunas de las especificaciones del patrón Modelo Vista Controlador, como gestionar los requisitos del modelo y de la vista a través del controlador, separando los datos, de la lógica de negocio y la interfaz de usuario; teniendo como consecuencia que los cambios realizados en cualquier lugar pueden generar una cadena de cambios en todo el código.

Para mantener la calidad y vigencia del sistema, es necesario corregir las dificultades inherentes a la interfaz de usuario, base de datos y programación expuestos anteriormente, para ello es necesario realizar actualizaciones de plataforma, crear estándares de programación y una restructuración adecuada de la base de datos, aportando aspectos para la evolución del sistema.

CONEST está formado actualmente por cinco módulos, administrativo, docente, estudiante, nuevo ingreso y escuela. La nueva versión de CONEST denominada CONEST 2.0, establece una nueva filosofía que se encuentra orientada

a procesos, los cuales son calificación, inscripción y graduación, que a su vez afecta las entidades estudiantes y docentes, contemplando todas las funcionalidades existentes en CONEST. Esta filosofía permite la reorganización del sistema, aumentando su eficiencia y eficacia.

2. Objetivo General

Desarrollar el módulo administrativo de la nueva versión del Sistema de Gestión Académica CONEST 2.0, tomando como referencia la versión anterior CONEST para realizar las migraciones, cambios y mejoras necesarias.

3. Objetivos específicos

- Implementar programa en Ruby que genere scripts[3], para la migración de los datos del sistema CONEST al sistema CONEST 2.0.

- Generar y desarrollar scripts, que contienen código SQL, para realizar la migración de los datos del sistema CONEST al sistema CONEST 2.0.

- Adaptar y aplicar el proceso de desarrollo de software Programación Extrema en la implementación de este sistema.

- Implementar componentes de programación bajo la filosofía de diseño MVC y así eliminar las redundancias donde sea adecuado.

- Aplicar estándares de programación de CONEST 2.0 para todas las instancias y procesos.

- Implementar componentes basados en la web 2.0, para mejorar la interacción con el sistema.

- Desarrollar código haciendo uso del marco de trabajo Ruby on Rails 2.3, para generar las funcionalidades de gestión administrativa como son agregar, consultar

[3] Script: Líneas de código que forman un conjunto de instrucciones diseñadas para ser usadas por un programa *(Internet Didáctica, 2007)*

y modificar los datos personales, académicos, laborales, deudas, horarios, constancias, entre otros, de las entidades estudiante y docente, así como la comunicación entre ellas y los procesos de calificación e inscripción correspondientes a la nueva versión de CONEST 2.0.

- Realizar pruebas de integridad y consistencia de los datos migrados del modelo de base de datos CONEST al modelo de CONEST 2.0.

- Aplicar pruebas unitarias para comprobar el correcto funcionamiento de los módulos de código en la aplicación.

- Realizar pruebas funcionales para verificar el comportamiento correcto del conjunto de funcionalidades dentro de la aplicación.

- Realizar pruebas de usabilidad, basándose en la opinión y necesidad del usuario en cuanto a las interfaces y funcionalidades implementadas.

4. Justificación

Las tecnologías con las que se ha desarrollado CONEST se encuentran en constante evolución y la demanda del usuario han aumentado, por lo que es necesario tomar en cuenta los avances que han ocurrido a través de los años y que permiten mejorar la facilidad de uso y eficiencia para mantener la calidad de este desarrollo web.

Este proyecto se encuentra alineado con el Plan Estratégico de la Universidad Central de Venezuela 2009 (UCV, 2009), que se refiere al desarrollo, ampliación y actualización de las plataformas tecnológicas utilizadas por estudiantes, docentes, personal administrativo, entre otros.

En consecuencia en este Trabajo Especial de Grado, se proponen soluciones que mejoren el sistema CONEST, adaptando las nuevas tecnologías, tomando en cuenta las deficiencias actuales, y así obtener una aplicación evolucionada para la gestión académica de Control de Estudio, la cual es denominada CONEST 2.0.

5. Alcance

El alcance de esta investigación abarca los siguientes puntos:

- Realizar la migración de los datos del sistema CONEST al sistema CONEST 2.0.
- Diseñar y desarrollar el conjunto de vistas, controladores, modelos y *helpers* necesarios para modelar las funcionalidades concernientes a las entidades estudiante y docente, detalladas a continuación:
  - Implementar las funcionalidades agregar, modificar y consultar los datos personales de un estudiante o un docente.
  - Implementar la funcionalidad de consultar los datos de la actuación académica de un estudiante, tales como eficiencia, promedio, inscripción, notas, entre otros.
  - Diseñar e implementar las funcionalidades que permitan realizar modificaciones a los datos académicos, de inscripción y calificación de un estudiante.
  - Realizar la verificación de pensum de un estudiante.
- Realizar las funcionalidades necesarias para la comunicación entre las entidades estudiantes y docentes con los procesos de calificación e inscripción, según sea el caso.

Capítulo II Marco Conceptual

La finalidad de este capítulo es presentar las bases conceptuales que sirven de fundamento para el desarrollo de este trabajo de investigación. El mismo se divide en dos secciones, las cuales se describen a continuación:

En la primera sección, se explican brevemente las Aplicaciones Web Cliente/Servidor. Adicionalmente se expone el patrón de diseño MVC (Modelo Vista Controlador), el cual es utilizado para el desarrollo de estas aplicaciones.

En la segunda sección, se muestran algunas herramientas tecnológicas necesarias para el desarrollo de una aplicación web, que a su vez constituyen la plataforma tecnológica del sistema ya desarrollado como antecedente a este trabajo de investigación. Entre las tecnologías del lado del cliente se hace referencia a HTML, CSS, JavaScript, AJAX y JQuery; en cuanto a las tecnologías del lado del servidor, se incluye Apache como servidor Web y Ruby on Rails como marco de trabajo para el desarrollo de Aplicaciones Web; por último se tienen las tecnologías del servidor de bases de datos, donde se expone a MySQL como Sistema Manejador de Bases de Datos Relacional.

6. Aplicaciones Web Cliente/Servidor

Una aplicación Web es un sistema informático que los usuarios utilizan accediendo a un servidor Web a través de Internet o de una Intranet. Las aplicaciones Web son populares debido a la practicidad del navegador Web como cliente ligero. La facilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. *(Aplicaciones Web, 2007)*.

Aunque muchas variaciones son posibles, una aplicación Web está comúnmente estructurada como una aplicación de tres-capas (Ver Figura 4). El navegador Web es la primera capa, la segunda capa es un servidor de aplicaciones usando alguna tecnología Web dinámica (ejemplo: PHP, Ruby on Rails, Java Servlets o ASP), y una base de datos como última capa. El navegador Web envía peticiones a la capa media, que la entrega valiéndose de consultas y actualizaciones a la base de datos generando una interfaz de usuario.

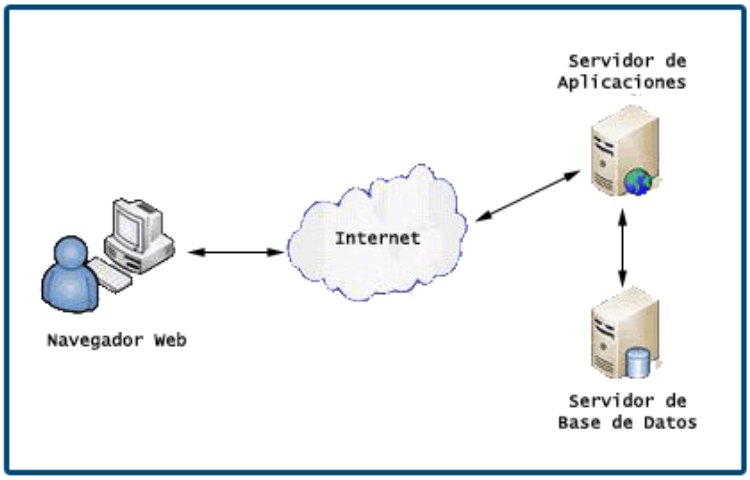

Figura 4 Aplicaciones Web de Tres Capas

Algunas de las ventajas que ofrecen las aplicaciones Web son las siguientes *(DIMAGIN Web Development, 2007)*:

- Se puede migrar de sistema operativo o cambiar el Hardware libremente sin

afectar el funcionamiento de las aplicaciones de servidor.

- No se requieren complicadas combinaciones de Hardware/Software para utilizar estas aplicaciones. Solo un computador con un navegador Web.

- Se facilita el trabajo a distancia. Se puede trabajar desde cualquier PC o computador portátil con conexión a Internet.

- Actualizar o hacer cambios en el Software es sencillo y sin riesgos de incompatibilidades. Existe solo una versión en el servidor lo que implica que no hay que distribuirla entre los demás computadores. El proceso es rápido y limpio.

- La habilidad de los usuarios a personalizar muchas de las características de la interfaz (como tamaño y color de fuentes, tipos de fuentes, ubicación de contenido, entre otros).

Así como tenemos ventajas, también se presentan ciertas desventajas en el uso de las Aplicaciones Web:

- La necesidad de conexión permanente y rápida a Internet hacen que el acceso a estas aplicaciones no esté al alcance de todos.

- La interactividad no se produce en tiempo real, en las aplicaciones Web cada acción del usuario conlleva un tiempo de espera hasta que se obtiene la reacción del sistema.

- Existen diferencias de presentación entre plataformas y navegadores. Sin embargo, la W3C[4] continúa trabajando en la estandarización de desarrollo de las aplicaciones Web, disminuyendo estas diferencias.

[4] W3C siglas de World Wide Web Consortium, es un consorcio internacional donde las organizaciones que son miembro, el personal a tiempo completo y el público en general, trabajan conjuntamente para desarrollar estándares Web. (W3C, 2008 a)

6.1. Patrón de Diseño MVC

Según Palasí (*2003*) el Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos, el Modelo, la Vista y el Controlador (de ahí su nombre). El Modelo está compuesto por el estado y los datos que la aplicación representa. La Vista es la interfaz de usuario que muestra información sobre el modelo y que representa el dispositivo de entrada que se usa para modificarlo. Finalmente, el Controlador es lo que une a los dos anteriores. Hace corresponder las peticiones que llegan del cliente con las acciones correspondientes y dirige las respuestas a las vistas adecuadas.

7. Herramientas Tecnológicas para el desarrollo de Aplicaciones Web

En los primeros tiempos de la computación cliente/servidor, cada aplicación tenía su propio programa cliente y su interfaz de usuario, éstos tenían que ser instalados separadamente en cada estación de trabajo de los usuarios. Una mejora al servidor, como parte de la aplicación, requería típicamente una mejora de los clientes instalados en cada una de las estaciones de trabajo, añadiendo un costo de soporte técnico y disminuyendo la eficiencia del personal. *(Eguiluz, J., 2009).*

7.1. Tecnologías del lado del Cliente

Las aplicaciones Web generan dinámicamente una serie de páginas en un formato estándar, soportado por navegadores Web. Se utilizan lenguajes interpretados del lado del cliente, tales como HTML, XHTML, CSS y JavaScript, los cuales serán explicados a continuación.

7.1.1. HyperText Markup Language (HTML)

HTML, traducido como "Lenguaje de Marcas de Hipertexto", es un lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web. *(Eguiluz, J., 2009 a)*

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

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas son definidas por la W3C. Utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determinan la forma en la que debe aparecer en su navegador el texto, así como también las imágenes y los demás elementos, en la pantalla del computador.

Lo más importante del código HTML es que no necesita de la instalación de una aplicación para crearlo y editarlo, además que es un lenguaje fácil de leer y escribir. Se puede utilizar editores de textos sencillos como Block de notas o Notepad que se encuentran en el sistema operativo Windows, o bien utilizar Gedit, Kedit, Kate, etc. en el caso de Linux. Sin embargo existen aplicaciones que permiten al programador o diseñador codificar la página Web y también diseñarla mediante interfaz gráfica generando el código automáticamente, como lo es Ms FrontPage incluido en la suite de Office y Dreamweaver de Macromedia de Adobe.

7.1.2. Extensible HyperText Markup Language (XHTML)

XHTML (Lenguaje de Marcado de Hipertexto Extensible) (*W3C, 2008 b)* es una versión más estricta y limpia de HTML, que nace precisamente con el objetivo de remplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML. XHTML extiende HTML 4.0 combinando la sintaxis de HTML, diseñado para mostrar datos, con la de XML[5], diseñado para describir los datos.

Ante la llegada al mercado de un gran número de dispositivos, XHTML surge

[5] El XML (eXtensible Markup Language) es un lenguaje con una importante función en el proceso de intercambio, estructuración y envío de datos en la Web. Describe los datos de tal manera que es posible estructurarlos utilizando para ello etiquetas. (W3C, 2008 c)

como el lenguaje cuyo etiquetado, más estricto que HTML, va a permitir una correcta interpretación de la información independientemente del dispositivo desde el que se accede a ella. XHTML puede incluir otros lenguajes como MathML, SMIL o SVG, al contrario que HTML.

XHTML, al estar orientado al uso de un etiquetado correcto, exige una serie de requisitos básicos a cumplir en lo que a código se refiere. Entre estos requisitos básicos se puede mencionar una estructuración coherente dentro del documento donde se incluirían elementos correctamente anidados, etiquetas en minúsculas, elementos cerrados correctamente, atributos de valores entrecomillados, etc. *(W3C, 2008 b)*

Algunos de los aspectos más importantes a la hora de utilizar XHTML son los siguientes:

- Los documentos deben estar bien formados: Un formato correcto en un documento XHTML es muy importante. Esto quiere decir que todos los elementos deben tener etiquetas de cierre, deben estar escritos de una forma determinada y además todos los elementos deben estar anidados correctamente.

- Los nombres de atributos y elementos deben ir en minúsculas: Tanto los elementos como los atributos deben ir en minúsculas para todos los elementos HTML y los nombres de atributos. Esto es importante ya que XML interpreta las mayúsculas y las minúsculas de forma diferente.

- Los valores de las etiquetas deben ir siempre entre comillas: Todos los valores de los atributos deben ir entre comillas, incluso aquellos que sean numéricos.

- Los elementos que no estén vacíos necesitan etiquetas de cierre.

7.1.3. Hojas de Estilo en Cascada (CSS)

Según Equiluz, J. (2009) las Hojas de Estilo en Cascada son un lenguaje de hojas de estilos creado para controlar la presentación de los documentos

electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos de su presentación.

La separación de los contenidos de su presentación 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.

Si el lenguaje HTML/XHTML se utiliza para marcar los contenidos, es decir, para distinguir lo que es un párrafo, lo que es un titular o lo que es una lista de elementos, el lenguaje CSS se utiliza para definir el aspecto de todos los contenidos, es decir, el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos, la tabulación con la que se muestran los elementos de una lista, etc. *(Eguiluz, J., 2009 b)*

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un selector y la declaración. A su vez la declaración está compuesta por una propiedad y el valor que se le asigne.

Una de las características principales de CSS es su flexibilidad y las diferentes opciones que ofrece para realizar una misma tarea. De hecho, existen 3 opciones para aplicar CSS en un documento HTML: *(Eguiluz, J., 2009 b)*

- Incluir CSS en el mismo documento HTML

Los estilos se definen en una zona específica del propio documento HTML. Se emplea la etiqueta <style> y solamente se pueden incluir en la cabecera del documento (<head>). Este método se emplea cuando se define un número pequeño de estilos o cuando se quieren incluir estilos específicos en un determinado documento HTML. El principal inconveniente es que si se quiere hacer

una modificación en los estilos definidos, es necesario modificar todas las páginas que incluyen el estilo que se va a modificar.

- Definir CSS en un archivo externo

Todos los estilos CSS se pueden incluir en un archivo de tipo CSS que los documentos HTML enlazan mediante la etiqueta <link>. Se pueden crear todos los archivos CSS que sean necesarios y cada documento HTML puede enlazar tantos archivos CSS como necesite. De las 3 formas de aplicar CSS a los documentos HTML, esta es la más utilizada (sobretodo mediante la etiqueta <link>.

La principal razón por la que es el método más utilizado es que es el más sencillo de mantener, ya que un solo cambio en un solo archivo CSS permite variar de forma instantánea los estilos de todos los documentos HTML que enlazan ese archivo.

- Incluir CSS en los elementos HTML

El último método para incluir estilos CSS en documentos HTML es el menos utilizado. Se realiza utilizando estilos directamente sobre aquellos elementos que lo permiten a través del atributo <style> dentro de <body>.

7.1.4. JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas Web dinámicas. *(Eguiluz, J., 2009 c)*

Una página Web dinámica es aquella que incorpora efectos como aparición y desaparición de texto, animaciones, acciones que se activan al pulsar botones u otros elementos 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. Su sintaxis es semejante a la del lenguaje Java y el Lenguaje C.

La integración de JavaScript y HTML es muy flexible, ya que existen al menos 2 formas para incluir código JavaScript en las páginas Web según Equiluz, J. (2009), como las siguientes:

- Incluir JavaScript en el mismo documento HTML

El código JavaScript se encierra entre etiquetas <script> y se incluye en cualquier zona del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta <head>).

El principal inconveniente es que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.

- Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta <script>. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite. Además del atributo type, este método requiere definir el atributo src, que es el que indica la URL correspondiente al archivo JavaScript. Cada etiqueta <script> solamente puede enlazar un único archivo, pero se pueden incluir tantas etiquetas <script> como sean necesarias.

Los archivos JavaScript son documentos normales de texto con la extensión .js, que se pueden crear con cualquier editor de texto como Notepad, Wordpad, etc.

La principal ventaja de enlazar un archivo JavaScript externo es que se

simplifica el código HTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio Web y que cualquier modificación realizada en el archivo JavaScript se ve reflejado inmediatamente en todas las páginas HTML que lo enlazan.

7.1.5. jQuery

Es una librería de JavaScript la cual está pensada para interactuar con los elementos de una página web por medio del árbol DOM[6]. Esta librería permite manejar eventos, desarrollar animaciones, y agregar interacción con la tecnología AJAX a las páginas web. *(Tutorial jQuery, 2007)*.

7.2. Tecnología del lado del Servidor de Bases de Datos

Según Menéndez (2000) los servidores de bases de datos surgen con motivo de la necesidad de manejar grandes y complejos volúmenes de datos, al tiempo que se requiere compartir la información con un conjunto de clientes de manera segura. Ante este enfoque, un Sistema Manejador de Base de Datos deberá ofrecer soluciones de forma fiable, rentable y de alto rendimiento.

7.2.1. Sistemas Manejadores de Bases de Datos (SMBD)

Según Di Campoy (1999) un SMBD consiste de un conjunto de datos relacionados entre sí y un conjunto de herramientas de software (y/o hardware) para tener acceso a esos datos. Es el SMBD que permite definir, procesar y administrar la Base de Datos (BD) y sus aplicaciones.

El SMBD puede organizar, procesar y presentar los datos seleccionados de una BD. Esta capacidad permite a quienes toman decisiones rastrear, probar y consultar el contenido de la base de datos para extraer las respuestas a las preguntas no recurrentes y no previstas en informes regulares.

[6] DOM (Document Object Model) contiene la especificación que determina cómo los objetos (texto, imágenes, enlaces, etc.) en una página web son representados *(ALEGSA, 2009).*

Entre las funciones del SMBD se tiene la definición y manipulación de los datos, seguridad, integridad, recuperación y concurrencia de los datos, diccionario de datos y en general, ejecutar todas estas funciones de la forma más eficiente posible.

El objetivo primordial de un SMBD es proporcionar un entorno para recuperar información y almacenar nueva información en la BD, para lo cual debe proporcionar a los usuarios una visión abstracta de los datos. Es decir, los detalles de cómo se almacenan y se mantienen los datos, son transparentes para los usuarios.

En la actualidad existe una variedad de sistemas manejadores de bases de datos; muchos de ellos son propietarios y algunos son software libre. Entre los SMBD comerciales o propietarios se tienen Microsoft SQL Server, Oracle, IBM Informix, Sybase, etc. y entre los de software libre están, MySQL y PostgreSQL, los más conocidos actualmente por su alto rendimiento, entre otros.

Hoy en día existen muchas empresas y sitios web que necesitan mantener de forma eficiente un gran volumen de datos. Muchos de ellos optan por soluciones comerciales, aunque muchas otras confían en el software libre optando por una solución como PostgreSQL o MySQL.

A continuación se explicará brevemente el sistema de gestión de bases de datos de software libre más utilizado actualmente, el cual proporciona soluciones a miles de personas, de forma totalmente gratuita.

7.2.2. MySQL

MySQL es un sistema manejador de bases de datos relacional, multihilo y multiusuario. MySQL AB es una compañía comercial de software libre, que distribuye y soporta MySQL, y además lo desarrolla bajo un esquema de licencia dual. Por un lado se tiene la licencia GNU GPL (General Public License GNU) y por otro, aquellas empresas que desean utilizar este sistema en productos privados,

pueden comprar la licencia que les permita su uso. *(Manual de referencia de MySQL 5.0., s.f.)*

Actualmente MySQL es el sistema de gestión de bases de datos más popular del mundo con más de 6 millones de instalaciones alrededor del mundo, con aproximadamente 40 mil descargas del software. MySQL es ampliamente utilizado en aplicaciones Web, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores. Su popularidad en las aplicaciones Web está muy ligada a PHP, que a menudo aparece en combinación con MySQL, sin embargo existen varias APIs que permiten, a aplicaciones escritas en diversos lenguajes de programación, acceder a las bases de datos MySQL, como son C, C++, C#, Pascal, Delphi (vía dbExpress), Eiffel, Smalltalk, Java (con una implementación nativa del driver de Java), Lisp, Perl, Python, Ruby, REALbasic (Mac), FreeBASIC, y Tcl.

7.3. Tecnologías del lado del Servidor Web

En el punto anterior se explicaron algunos de los diferentes lenguajes utilizados del lado del cliente, sin embargo existen diferentes tecnologías que se han desarrollado para coordinar dichos lenguajes con las tecnologías del lado del servidor, como por ejemplo PHP, Ruby on Rails, JSP, ASP, etc.

7.3.1. Contenedores Web - Apache

El servidor HTTP Apache es un software servidor HTTP de código abierto para plataformas Unix, Windows, Macintosh y otras, que implementa el protocolo HTTP y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en el código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. Su nombre se debe a que originalmente Apache consistía solamente en un conjunto de "parches" a aplicar al servidor de NCSA. *(Servidor HTTP Apache, 2007).*

Apache presenta entre otras características mensajes de error altamente

configurables, bases de datos de autenticación y negociado de contenido, pero es criticado por la falta de una interfaz gráfica que ayude en su configuración.

Apache tiene amplia aceptación en la red: en el 2005, Apache fue el servidor HTTP más usado, siendo usado en el 48% de los sitios Web en el mundo.

La mayoría de las vulnerabilidades de seguridad descubiertas y resueltas puede en la mayoría de los casos ser abusada solamente por los usuarios locales y no puede ser accionada remotamente. Sin embargo, algunas de las ediciones mencionadas se pueden accionar remotamente en ciertas situaciones, o explotar por los usuarios locales malévolos en las disposiciones de recibimiento compartidas que utilizan PHP como módulo de Apache.

7.3.2. Tecnologías Web 2.0

La web 2.0 es la representación de la evolución de las aplicaciones tradicionales utilizada para desarrollar aplicaciones interactivas, en donde el contenido de la página es compartido, producido o cambiado por el usuario (Van Der Henst, 2005). Para que una página cumpla con la web 2.0 debe cumplir con dos o más de los siguientes elementos: (O'Reilly, 2005).

- Instantaneidad: permite obtener, procesar y validar datos desde una misma página (Simplifica la lógica de negocio y permite realizar el mismo trabajo con menos páginas). Este elemento se puede conseguir utilizando tecnologías como AJAX, JavaScript o jQuery.

- Interactividad avanzada: se refiere a la interacción más completa del usuario con una página web, de forma sencilla y usable, se puede lograr mostrando elementos que hagan más accesible la aplicación.

- Conectividad: contiene la capacidad de conexión con bases de datos, archivos de intercambio, entre otros.

- Lógica de negocio avanzada: se trata de poder realizar tareas de proceso de

información complejas, guardar información en base de datos, almacenar sesiones y datos de usuarios, llevar seguimientos, entre otras. Estas lógicas son logradas por tecnologías del lado del servidor por lenguajes de programación como PHP, Ruby, ASP, entre otros.

- Multimedia: es la capacidad de servir vídeos, música, presentaciones, o textos de forma amigable e interactiva (por ejemplo con buscadores de contenidos, barra de reproducción/seguimiento, maximizar/minimizar contenido y compartir).

7.3.3. Tecnologías Web – Ruby on Rails

Ruby on Rails, también conocido como RoR o Rails, es un framework de aplicaciones Web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación[7], de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de librerías y aplicaciones Ruby. *(Ruby on Rails, 2008).*

- **Características de Ruby**

Ruby tiene un conjunto de características entre las que se encuentran las siguientes: *(Acerca de Ruby, s.f.).*

- Manejo de excepciones, como Java y Python, para facilitar el manejo de errores.

- Un colector de basura para todos los objetos de Ruby. No es necesario mantener contadores de referencias en bibliotecas externas.

[7] Metaprogramación: consiste en escribir programas que manipulan otros programas como datos. Esto permite al programador ahorrar tiempo en la producción de código. (Metaprogramación, 2009)

- Escribir extensiones en C para Ruby es más fácil que hacer lo mismo para Perl o Python, con una API muy elegante para utilizar Ruby desde C. Esto incluye llamadas para embeber Ruby en otros programas, y así usarlo como lenguaje de scripting. También está disponible una interfaz SWIG.

- Puede cargar bibliotecas de extensión dinámicamente si lo permite el sistema operativo.

- Tiene manejo de hilos (threading) independiente del sistema operativo. De esta forma, tienes soporte multi-hilo en todas las plataformas en las que corre Ruby, sin importar si el sistema operativo lo soporta o no.

- Ruby es portable: se desarrolla mayoritariamente en GNU/Linux, pero corre en varios tipos de UNIX, Mac OS X, Windows 95/98/Me/NT/2000/XP, DOS, BeOS, OS/2, etc.

- **Filosofía**

Los principios fundamentales de Ruby on Rails incluyen "No te repitas" (del inglés Don't repeat yourself, DRY) y "Convención sobre configuración". *(Acerca de Ruby, s.f.)*

"No te repitas" significa que las definiciones deberían hacerse una sola vez. Dado que Ruby on Rails es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en ActiveRecord, las definiciones de las clases no necesitan especificar los nombres de las columnas; Ruby puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante.

"Convención sobre configuración" significa que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, si hay una clase Historia en el modelo, la tabla correspondiente de la base de datos es historias,

pero si la tabla no sigue la convención debe ser especificada manualmente. Así, cuando se diseña una aplicación partiendo de cero sin una base de datos preexistente, el seguir las convenciones de Rails significa usar menos código.

- **Arquitectura MVC de Rails**

Las piezas de la arquitectura Modelo Vista Controlador en Ruby on Rails son las siguientes: *(Acerca de Ruby, s.f.)*

- **Modelo**

En las aplicaciones Web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos.

En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase Imagen tiene una definición has_many :comentarios, y existe una instancia de Imagen llamada a, entonces a.comentarios devolverá un arreglo con todos los objetos Comentario cuya columna imagen_id (en la tabla comentarios) es igual a a.id.

Las rutinas de validación de datos (p.e. validates_uniqueness_of :checksum) y las rutinas relacionadas con la actualización (p.e. after_destroy :borrar_archivo, before_update :actualizar_detalles) también se especifican e implementan en la clase del modelo.

El modelo representa:

- Las Tablas de la Base de Datos.
- Migraciones (Expresan cambios en las BD)

- Observadores

- **Vista**

En MVC, Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador. Con frecuencia en las aplicaciones Web la vista consiste en una cantidad mínima de código incluido en HTML.

Existen en la actualidad muchas maneras de gestionar las vistas. El método que se emplea en Rails por defecto es usar Ruby Embebido (archivos .html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP.

Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del controlador y los fragmentos pueden invocarse unos a otros.

- **Controlador**

En MVC, las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones Web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador Web.

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la Web, por lo general en la forma http://aplicacion/ejemplo/metodo, que invoca a EjemploController#metodo, y presenta los datos usando el archivo de plantilla

/app/views/ejemplo/metodo.html.erb, a no ser que el método redirija a algún otro lugar.

Rails también permite construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes a través de plantillas.

- **Componentes de Rails**

Los componentes del framework Rails son los siguientes *(The Pragmatic Programmers LLC, 2007)*:

- ***ActiveRecord***

El Active Record es la capa de ORM[8] suministrada por Rails. Mantiene el estándar ORM de la siguiente manera: las tablas equivalen a las clases, las filas a objetos, y las columnas a atributos. Difiere de la mayoría de las bibliotecas de ORM en el sentido que es configurable. Una de las ventajas del Active Record es que disminuye la cantidad de configuraciones que los desarrolladores deben realizar y además les provee la implementación de la vista de las tablas maestras para su creación, modificación y eliminación. *(The Pragmatic Programmers LLC, 2007)*

El Active Record proporciona al usuario de Rails ciertas funcionalidades como relaciones entre los objetos, agregación, herencia, métodos de búsqueda, validaciones, transacciones, callbacks, entre otras.

Las validaciones que presta el Active Record permiten agregar restricciones de dominio y lógica de negocios en el modelo y los Callbacks permiten activar lógica antes o después de una modificación del estado del objeto, es decir, antes o después de su creación, actualización o destrucción.

[8] ORM (Object-Relational Mapping) es un componente de software que permite trabajar con los datos de una base de datos relacional como si ellos fueran parte de una base de datos orientada a objetos.

- ***ActionPack***

Es un componente de Rails que provee la separación necesaria para escribir de manera clara y delimitada el código destinado al control y a la presentación sin que esta afecte la alta interacción que existe entre las vistas y los controladores. *(The Pragmatic Programmers LLC, 2007)*

- ***ActionView***

En Rails, *ActionView* es la capa responsable de crear toda o parte de la página que será desplegada en el navegador.

En Rails, el contenido dinámico es generado por plantillas, las cuales vienen en tres estilos. El esquema de plantillas más común es el rhtml, el cual es simplemente pedazos de código de Ruby embebido dentro de la vista de HTML usando una herramienta de Ruby llamada ERb. Este enfoque es muy flexible, pero algunos puristas del área a veces manifiestan que esto viola el espíritu de MVC. Al introducir dicho código en la vista nos arriesgamos a asignar una lógica que debería ir dentro del modelo o del controlador. Este desacuerdo sobre el código dinámico en las vistas por parte de los expertos en el área es en gran parte injustificado, debido a que las vistas vienen teniendo dicho código desde las arquitecturas originales de MVC. El hecho de mantener una clara separación entre la vista y los controladores es un trabajo estricto del desarrollador.

El segundo esquema de plantillas es llamado rxml, el cual se construye usando documentos XML y código de Ruby. Y por último, Rails también provee vistas rjs. Dichas vistas le permiten al programador crear fragmentos de JavaScript dentro del servidor que luego serán ejecutados en el navegador. Estas vistas presentan ventajas a la hora de crear interfaces dinámicas de Ajax.

- ***ActionController***

ActionController es la capa responsable de recibir las peticiones web y tomar

decisiones con respecto a qué se debe ejecutar o si se debe redirigir la petición a otra acción. Una acción es definida como método público dentro de los controladores que están automáticamente disponibles. *(Brando, 2008)*

El controlador también alberga un número importante de servicios adicionales: *(The Pragmatic Programmers LLC, 2007)*

- Es responsable de transformar las peticiones externas a acciones internas. De la misma manera maneja muy bien la creación de URLs amigables.

- Administra el proceso de captura, el cual otorga a la aplicación el orden de magnitud para el incremento del desempeño.

- Administra módulos de ayuda, el cual extiende las capacidades de las plantillas de las vistas sin necesidad de construir el código.

- Administra sesiones, lo que les otorga a los usuarios la impresión de una interacción progresiva con las aplicaciones.

- ***ActionMailer***

Este componente permite enviar correos electrónicos desde la aplicación que se está utilizando usando vistas y modelos especiales de la clase Mailer. *(Rails Documentation, 2008).*

- ***ActiveResource***

ActiveResource es la capa responsable de la implementación de sistemas RESTful del lado del cliente. A través de *ActiveResource* es posible consumir servicios RESTful mediante el uso de objetos que actúan como un *proxy* para los servicios remotos. *(Brando, 2008).*

- **Ajax en Rails**

Ajax es una técnica que permite usar Javascript y XML para procesar peticiones de un navegador Web a un servidor Web como procesamiento en segundo plano sin cargar otras páginas Web adicionales. Rails proporciona diferentes facilidades que hacen más fácil implementar aplicaciones Ajax. *(The Pragmatic Programmers LLC, 2007).*

Rails es anfitrión tanto del framework Prototype en Javascript (una serie de herramientas que proporcionan llamadas Ajax y otra funcionalidad habitual en las tareas cliente-servidor) y script.aculo.us, una librería en Javascript con mejoras en la interfaz de usuario (controles avanzados en los formularios, efectos visuales, arrastrar y soltar).

- **Soporte de Bases de Datos**

Dado que la arquitectura Rails favorece el uso de bases de datos se recomienda usar un SGBD para almacenamiento de datos. Rails soporta la librería SQLite si no es posible emplear una base de datos. El acceso a la base de datos es totalmente abstracto desde el punto de vista del programador, y Rails gestiona los accesos a la base de datos automáticamente (aunque, si se necesita, se pueden hacer consultas directas en SQL) Rails intenta mantener la neutralidad con respecto a la base de datos, la portabilidad de la aplicación a diferentes sistemas de base de datos y la reutilización de bases de datos preexistentes. Sin embargo, debido a la diferente naturaleza y prestaciones de los SGBDs el *framework* no puede garantizar la compatibilidad completa. Se soportan diferentes SGBDs, incluyendo MySQL, PostgreSQL, SQLite, IBM DB2, Oracle y Microsoft SQL Server. *(Ruby on Rails, 2008).*

Rails es un completo entorno para desarrollar aplicaciones Web con base de datos de acuerdo con la estructura Modelo-Vista-Controlador. Desde el Ajax en la vista, a la petición y respuesta en el controlador, hasta el modelo, Rails da un entorno de desarrollo de Ruby. Para probarlo, solo se necesita una base de datos y

un servidor Web. *(Ruby on Rails, 2008).*

7.3.4. Tecnologías Web – Rails 2.3

Alrededor de julio de 2004 David Heinemeier Hansson lanzó la versión pública del framework Ruby on Rails. Después de tres años, se publicó Ruby on Rails 2.0 con una cantidad importante de cambios. Para estos cambios han contribuido más de 1400 desarrolladores de todo el mundo con aproximadamente 1600 parches para el framework. Luego de toda esta colaboración se han publicado tres versiones más de Rails, la 2.1, la 2.2 y la 2.3, donde a continuación se presentan algunos de los cambios más significativos que surgieron. *(Brando, 2008)*

- ***ActiveRecord***

Las siguientes propias de este componente tuvieron cambios significativos en Rails 2.3:

- Método SUM

En la nueva versión de Rails se pueden utilizar expresiones en los métodos de cálculo de *ActiveRecord*, como por ejemplo el método sum.

También cambian los valores de retorno en este método. En las versiones previas, al utilizar el método sum de *ActiveRecord* para calcular la suma de una determinada columna para todos los registros de una tabla, se retornaba **nil** si ningún registro correspondía con las condiciones expresadas en el método de invocación. En cambio en Rails 2.3 el valor de retorno por defecto en esta condición es 0.

- has_one

El método *has_one* ahora tiene la opción *through*. Esta funciona como *has_many :through*, pero en este caso representa la asociación a un objeto simple *ActiveRecord*.

El método *has_one :through* puede también tomar :source_type como argumento. Por ejemplo, se tiene una clase Cliente que está relacionada con la clase TarjetaContacto que a su vez está relacionada con las clases Persona y Negocios. Gracias al argumento :source_type la clase Cliente tiene como tipos de contacto personas y negocio.

- Incremento y Decremento

Los métodos de *ActiveRecord* increment, increment!, decrement y decrement! ahora en Rails 2.3 tienen un parámetro opcional, donde se indica cual es el valor que se quiere restar o sumar en la función dada.

- Find

En el método find de Rails ahora se puede pasar un objeto como parámetro, haciendo una correspondencia en el modelo que implique la búsqueda.

- Parámetro last

Hasta ahora sólo podíamos usar tres operadores para buscar datos usando el método find de *ActiveRecord* el :first, :all y el id del objeto, actualmente en Rails 2.3 se cuenta con otro operador denominado :last que busca el último registro en la BD o según una condición dada.

- Eager Loading

Un avance de Rails 2.3 es que se puede hacer una búsqueda en una tabla incluyendo en los parámetros las tablas relacionadas a ésta, esto a través de la clave foránea establecida por la convención de Rails.

- Relaciones de Sólo Lectura

Se agregó en Rails 2.3 una nueva característica a las relaciones entre modelos. Con el fin de impedir alterar los modelos ahora podemos usar ‘:readonly => true’ para asociar modelos. De esta manera los registros protegidos no podrán

ser alterados desde este modelo.

- Métodos add_timestamps y remove_timestamps

  Estos nuevos métodos agregan y eliminan las columnas de timestamp.

- Cálculos

ActiveRecord::Calculations en Rails 2.3 no sólo acepta el nombre de la columna sino también el nombre de la tabla, siendo esto muy útil cuando se tiene una relación entre dos tablas que tienen una columna con el mismo nombre.

- Objetos Corruptos

Ahora en Rails 2.3 se pueden rastrear los cambios realizados por *ActiveRecord*. Se puede saber si un objeto ha sido modificado o no. En caso de que haya cambiado, podemos ver qué cambios ocurrieron con exactitud como también comparar su estado anterior con el actual.

- Actualizaciones parciales

En las versiones previas de Rails cuando se ejecutaba el método *save* de un objeto *ActiveRecord*, se actualizaban todos los campos en la base de datos. Ahora se actualizan en la base de datos los campos que fueron cambiados en la aplicación. Si ningún cambio ha sido actualizado entonces el *ActiveRecord* no ejecutará ningún cambio en la base de datos.

- :select en has_one y belongs_to

Estos métodos ahora tienen la opción de hacer un :select en el modelo, el valor por defecto es "" (vacío), pero se puede editar para recuperar sólo las columnas que se quieren utilizar.

El método belongs_to ya no tiene la opción :order.

- ***ActiveSupport***

Existen diferentes cambios generados en *ActiveSupport* como se tiene:

- ActivezSupport::CoreExtensions::Date::Calculations

- Time#end_of_day que retorna la fecha actual con la hora fijada en 11:59:59 PM.

- Time#end_of_week en donde se retorna el fin de semana (Sunday 11:59:59 PM).

- Time#end_of_quarter retorna un objeto date representando el fin del trimestre, es decir, retorna el último día de los meses: marzo, junio, septiembre o diciembre, dependiendo de la fecha actual.

- Time#end_of_year retorna 31 de Diciembre a las 11:59:59 PM.

- Time#in_time_zone este método utiliza Time.zone en vez de basarse en el timezone del sistema operativo. Se le puede pasar como parámetro tanto un objeto TimeZone como un String.

- DateTime#to_f

Es un nuevo método que retorna la fecha como un flotante representando el número de segundos desde el 1 de Enero de 1970 a la medianoche.

- Memoization

Memoization es un patrón para inicializar un método una vez y luego guardando su valor para su uso repetido. Por ejemplo:

```
extend ActiveSupport::Memoizable

def full_name
  "#{first_name} #{last_name}"
end

memoize :full_name
```

- ***ActiveResource***
  - TimeOuts

ActiveResource usa HTTP para acceder a la API de RESTful y por esto es susceptible a problemas con servidores lentos o servidores no alcanzables. En algunos casos, las llamadas a *ActiveResource* pueden expirar (timeout). Ahora se puede controlar el tiempo de expiración con la propiedad timeout.

- ***ActionPack***
  - Auto link

El método auto_link recibe cualquier texto como parámetro, y si el texto contiene alguna dirección de email o website, retorna el mismo texto, pero con hipervínculos.

  - Etiquetas

Se incluyó el método label en la creación de los formularios por medio del scaffold. Este método retorna un *string* con el título de la columna dentro de un tag HTML.

  - Feeds de Atom

Atom es el nombre de estilos y metadatos en formato XML. En otras palabras es un protocolo para publicar contenidos en Internet que son actualizados

con frecuencia, como por ejemplo un blog o página de noticias. Los Feeds se publican en XML y en Atom son identificados como tipo de medios application/atom+xml.

Los parámetros aceptados por este parámetro son :language, :root_url, :url, entre otros y se pueden incluir espacios de nombre en el elemento raíz del feed.

- action_name

Para saber cual vista fue invocada durante su ejecución, se utiliza el método action_name.

- Session (:ON)

Es posible activar o desactivar sesiones en Rails 2.3. Esto se puede hacer a nivel general o específico en cuanto a los controladores.

Para desactivar las sesiones en todos los controladores se coloca sesión :off en la clase *ApplicationController*. Si queremos hacer una activación de sesiones en un solo controlador se agrega sesión :on al controlador deseado.

- ***ActionController***

- ActionController::Routing

Reconocimiento de Rutas

En versiones anteriores a Rails el reconocimiento de rutas hacía un recorrido de todas la rutas una a una, consumiendo así mucho tiempo. En la nueva versión de Rails se genera un árbol para las rutas y el reconocimiento de rutas se hace mediante un prefijo, dejando de lado las rutas similares.

Se pueden modelar las rutas para que correspondan con el idioma determinado en el explorador por medio de la opción :as dentro de map.resources

para personalizar estas rutas.

- ***ActionView***

De este componente se agregaron y cambiaron las siguientes funciones:

- ActionView::Helpers::DateHelper

Ahora todos los métodos del módulo para trabajar con fechas (date_select, time_select, select_datetime, etc.) aceptan opciones HTML.

- ActionView::Helpers::FormTagHelper

submit_tag

Se agregó la opción :confirm a los parámetros del método #submit_tag. Esta opción funciona de la misma forma que el método link_to.

- ActionView::Helpers::TextHelper

simple_format

El nuevo método simple_format recibe como parámetro cualquier texto y le da formato HTML. Su función toma el texto y reemplaza los saltos de línea (\n) por el tag HTML "
". En caso que existan dos saltos de línea, uno después del otro, separa el texto con tags "<p>".

- Protección contra *Cross Site Scripting*

Para prevenir el Cross Site Scripting en Rails 2.3 cuenta con un método llamado protect_from_forgery, el cual se agrega en el ApplicationController, que asegura que todos los formularios de la aplicación están recibiendo datos que vienen de ella, y no desde un link en otro sitio. Para hacerlo se inlcuye un *token* basado en la sesión en todos los formularios y peticiones *ajax* generadas por Rails, y luego verificando la autenticidad de este *token* en el controlador. *(Brando, 2008)*

- ***ActionMailer***

ActionMailer ahora soporta el uso de plantillas de correos, es decir, los correos HTML pueden ser modificados como en las vistas del buscador, todo esto es posible suministrando a la plantilla un nombre apropiado. Por ejemplo la clase Correo espera que se utilice la dirección layouts/correo.html.erb.

Capítulo III Marco Aplicativo

En este capítulo se presenta una adaptación del proceso de desarrollo de software, basado en el método de Programación Extrema (XP) para la construcción de la Aplicación Web. En este sentido, se describe el contexto de desarrollo y cada una de las fases del método utilizado para la implementación de las entidades docentes y estudiantes, así como la comunicación entre éstas y los procesos de inscripción y calificación.

8. Proceso de Desarrollo XP

A continuación se explica el método de Programación Extrema, la adaptación realizada para este Trabajo Especial de Grado y las iteraciones necesarias para el desarrollo de la aplicación CONEST 2.0.

8.1. Programación Extrema (XP)

XP se trata de un método diseñado para entregar al cliente el software cómo y cuando lo establezca, respondiendo rápidamente a las necesidades del mismo, incluso cuando los cambios sean al final del ciclo de la programación.

Según Ron Jeffries (Pressman, 2007) "la Programación Extrema es una disciplina de desarrollo de software que se basa en valores de simplicidad, comunicación, retroalimentación y audacia".

A continuación se listan las características de este método de desarrollo, las cuales son puestas en práctica las actividades de XP. (Jeffries, 2001)

- **Características de XP**
  - Desarrollo iterativo e incremental
  - Pruebas unitarias continuas
  - Programación en parejas
  - Comunicación constante con el cliente
  - Corrección de errores
  - Refactorización de código
  - Simplicidad
  - Propiedad de código compartida

El proceso de desarrollo se divide principalmente en iteraciones las cuales contienen 4 actividades (Planificación, Diseño, Codificación y Pruebas), explicadas en la siguiente sección.

8.2. Adaptación de XP

A continuación se describen las tareas que involucran las actividades en la adaptación del proceso XP que se utilizó durante el desarrollo de la aplicación Web.

8.2.1. Iteraciones

El método XP propone dividir el trabajo en iteraciones, las cuales se enfocan en versiones parciales del sistema hasta llegar al producto final. Los nuevos requerimientos son recibidos progresivamente y son incluidos a una nueva iteración.

Las iteraciones pueden ser de dos tipos principalmente: por objetivos o por lapsos de tiempo. En el desarrollo de este Trabajo Especial de Grado, las iteraciones están basadas en intervalos de tiempo, estimados entre dos y tres semanas. Durante el tiempo fijado para cada iteración, se realizan las implementaciones indicadas en las historias de usuario, de no completarse algún requerimiento en este lapso, el mismo es agregado a la próxima iteración.

8.2.2. Planificación

Según XP, la actividad de planificación comienza creando una serie de Historias de Usuario, en las cuales se describen en una o dos oraciones los requerimientos del sistema en terminología del cliente, proporcionando a su vez una estimación del tiempo necesario para el desarrollo. (Pressman, 2007)

En este trabajo, el formato para escribir las Historias de Usuario se presenta en la Tabla 1, en donde se indican las fechas correspondientes al inicio y fin de cada iteración, además, se pueden observar tres columnas que muestran información de cada historia de usuario. La primera corresponde al número que

sirve como identificador, la segunda es la fecha de inicio y la tercera es una breve descripción de los requerimientos.

Fecha Inicio: Fecha Fin:

Número	Fecha	Descripción

Tabla 1 Formato de Historias de Usuario

Frecuentemente el equipo de desarrollo entrega varias versiones del sistema al cliente, y así en cada versión se puede ir incorporando las funcionalidades que no se han contemplado en entregas anteriores.

8.2.3. Diseño

El método de desarrollo Programación Extrema, propone la fase de diseño como una guía de implementación para una historia de usuario determinada.

Para esta fase, XP recomienda la creación de prototipos y/o diagramas, cuando establecer un diseño para una historia de usuario resulte complicado. Asimismo, promueve aplicar refactorización, que permita realizar mejoras al diseño del código después que se ha escrito. (Pressman, 2007)

Siguiendo las prácticas de XP, para el desarrollo de este trabajo, son diseñados diagramas de objetos de dominio, así como otros diagramas o prototipos, que permitan comprender y solucionar el problema que aplica a cada iteración. Al mismo tiempo, se aplica refactorización para modificar el diseño del código del sistema por otro que se adapte mejor a la solución.

8.2.4. Codificación

Según Pressman (2007), XP recomienda, que luego de establecer las historias de usuario y realizar el trabajo de diseño preliminar, no se debe comenzar la codificación, sino que se deben determinar casos de prueba que evalúen cada

una de las historias por cada iteración.

Durante la actividad de codificación, este método sugiere la programación en pareja, que consiste en dos personas en una misma estación de trabajo desarrollando el código de una historia de usuario. Esto ayuda a seguir los estándares de programación, lo cual es otro aspecto requerido por el método de Programación Extrema. Por último se deben realizar frecuentes integraciones de código entre los grupos de trabajo, permitiendo evitar problemas de compatibilidad e interfaz y ayudando al descubrimiento de errores en el sistema. (Pressman, 2007)

Para el desarrollo de este sistema, se adopta la programación en pareja con ciertas modificaciones, ya que siempre se trabaja sobre una iteración pero no necesariamente en la misma historia de usuario, esto depende de la dificultad del requerimiento. Se siguen estándares de codificación establecidos para el desarrollo de CONEST 2.0 (Grupo de Desarrolladores de CONEST, 2009), manteniendo la consistencia y legibilidad del código y facilitando la comprensión para los involucrados en el desarrollo del sistema. También se realiza una integración constante del código por medio del sistema de control de versiones (subversión), el cual es el repositorio utilizado a la hora de almacenar los cambios realizados.

8.2.5. Pruebas

XP establece que se deben codificar y automatizar las pruebas unitarias creadas en la fase de diseño, para cada historia de usuario; así como realizar pruebas de aceptación por cada iteración terminada, las cuales son especificadas por el cliente, enfocándose en las funcionalidades del sistema que son manejadas por él. (Pressman, 2007).

En el desarrollo de CONEST 2.0, se realizan, además las pruebas establecidas por la Programación Extrema, un conjunto de pruebas adicionales. Cada una de ellas es aplicada dependiendo de la historia de usuario que se esté verificando. Dichas pruebas son listadas a continuación:

- Pruebas de Consistencia e Integridad, aplicadas después de realizar la migración de los datos.

- Pruebas Unitarias de Ruby, centradas en los modelos y funciones más importantes de la aplicación Web.

- Pruebas Funcionales aplicadas con Cucumber. Estas herramientas permiten a los desarrolladores automatizar las pruebas, describiendo el comportamiento de un programa, en lenguaje natural escrito en texto plano. Estas pruebas son divididas por escenarios, los cuales corresponden a cada historia de usuario. El comportamiento del escenario se escribe con las palabras clave *"Dado que"*, *"Cuando"* y *"Entonces"*. *"Dado que"* especifica una precondición del escenario, *"Cuando"* las acciones a realizar y *"Entonces"* describe la respuesta esperada. Cada resultado es mostrado en colores verde y rojo, los cuales representan éxito y fallo respectivamente (Cucumber, *2008*).

- Pruebas de Usabilidad. Es realizada una adaptación del método Categorización de Contenido, para medir el nivel de aceptación que tiene el usuario sobre las interfaces y funcionalidades realizadas.

8.3. Implementación

En la siguiente sección es explicada la implementación de la metodología XP durante el desarrollo de la aplicación CONEST 2.0, especificando las 4 etapas correspondientes a cada iteración.

8.3.1. Iteración 0

El motivo de esta iteración es realizar la automatización de la migración de datos del modelo de CONEST al modelo de CONEST 2.0, permitiendo su ejecución en cualquier momento. Este proceso es realizado mediante la creación de scripts en código SQL y dependiendo de la dificultad de la migración, se desarrolla un programa que los genere.

- **Planificación**

En la siguiente la Tabla 2 se muestran las historias de usuario desarrolladas en esta iteración.

Fecha Inicio: 06/04/09 Fecha Fin: 24/04/09

Nro	Fecha	Descripción
1.0	06/04/09	Migración del modelo de datos CONEST a CONEST 2.0

Tabla 2 Planificación Iteración 0

- **Diseño**

Antes de realizar los diferentes scripts y programas para la migración, es necesario el estudio y comparación de los modelos de datos correspondientes a cada sistema, con el fin de localizar en CONEST, los atributos necesarios para la creación de las tablas en CONEST 2.0, un ejemplo de esto es presentado en la Figura 5.

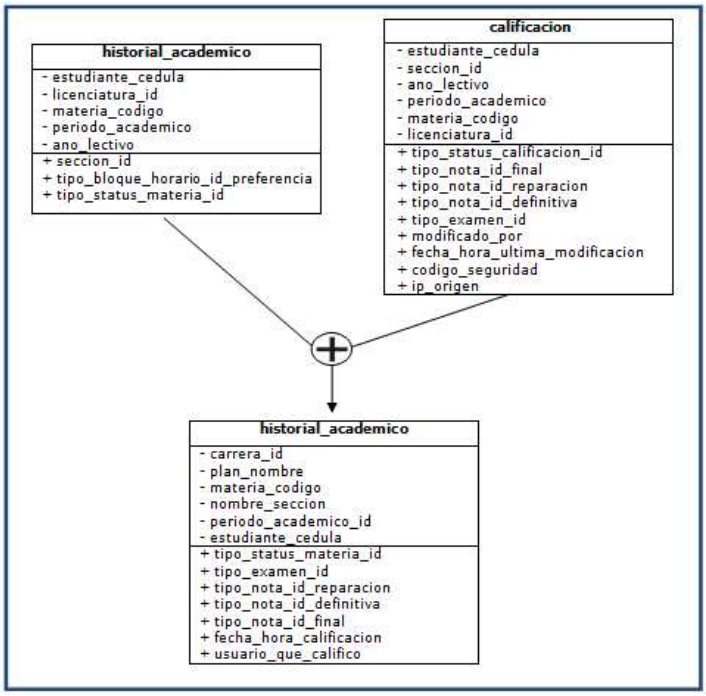

Figura 5 Creación de Historial Académico en CONEST 2.0.

También en la Figura 5 se puede observar que para la creación de Historial Académico de CONEST 2.0, es necesaria una composición de atributos de las tablas Historial Académico y Calificación de CONEST.

- **Codificación**

Para cumplir con los objetivos de la iteración, se crean scripts con código SQL, que realiza la migración automatizada de los datos del modelo de base de datos CONEST a una versión nueva de este modelo. En la Figura 6 se muestra un fragmento de código SQL que realiza parte de la migración de la tabla Historial Académico.

```
INSERT INTO conest2_development.historial_academico SELECT t1.licenciatura_id, 'PLAN COMPUTACION', t1.materia_codigo, nombre,
   CONCAT(t1.periodo_academico,'-',t1.ano_lectivo) AS periodo_academico_id, estudiante_cedula,tipo_status_materia_id,NULL,NULL,NULL,NULL,NULL,NULL
   FROM conest_development.historial_academico AS t1
   INNER JOIN conest_development.seccion AS t2 ON t2.id=t1.seccion_id
   INNER JOIN conest2_development.oferta_en_periodo ON t1.materia_codigo = conest2_development.oferta_en_periodo.materia_codigo
   AND t1.licenciatura_id = conest2_development.oferta_en_periodo.carrera_id
   AND CONCAT(t1.periodo_academico,'-',t1.ano_lectivo) = conest2_development.oferta_en_periodo.periodo_academico_id
   AND t1.licenciatura_id = 'C';

UPDATE conest2_development.historial_academico h2, conest_development.calificacion c1,conest_development.tipo_nota t1,
   conest_development.tipo_nota t2,conest_development.tipo_nota t3,conest_development.seccion s1
   SET h2.tipo_examen_id=c1.tipo_examen_id, h2.tipo_nota_id_reparacion=t1.nombre, h2.tipo_nota_id_definitiva=t2.nombre,
   h2.tipo_nota_id_final=t3.nombre, h2.usuario_que_califico=c1.modificado_por WHERE
   h2.estudiante_cedula=c1.estudiante_cedula AND
   h2.carrera_id=c1.licenciatura_id AND
   h2.periodo_academico_id = concat(c1.periodo_academico, '-', c1.ano_lectivo) AND
   h2.materia_codigo = c1.materia_codigo AND
   h2.nombre_seccion = s1.nombre AND
   c1.seccion_id = s1.id AND
   c1.tipo_nota_id_reparacion IS NOT NULL AND
   (c1.tipo_nota_id_reparacion = t1.id AND
   c1.tipo_nota_id_definitiva = t2.id AND
   c1.tipo_nota_id_final = t3.id) AND
   t1.nombre NOT IN ('RET','EQ') AND
   t2.nombre NOT IN ('RET','EQ') AND
   t3.nombre NOT IN ('RET','EQ');
```

Figura 6 Código SQL para migrar la tabla Historial Académico

En los casos en los que la lógica del modelo de datos de CONEST 2.0 no coincide con la de CONEST, se opta por realizar un programa en lenguaje Ruby que genere los scripts SQL necesarios. Por ejemplo, en la Figura 7 se tiene un segmento del código, que separa el atributo *nombre* de la tabla docente, que contiene su nombre completo en CONEST, para insertarlos en la tabla usuario de CONEST 2.0, en donde se tienen los atributos *primer_nombre*, *segundo_nombre*, *primer_apellido* y *segundo_apellido*.

```ruby
class Cambios
  def self.separar_nombre
    docente = Docente.find(:all)
    File.open("/home/usuarios.sql","w"){|f|
  #Separar nombre de docente para crear script para insertarlos en usuario
    docente.each do |doc|
      estudiante = Estudiante.find(:first, :conditions => ["cedula=?",doc.cedula])
      if (!estudiante)
        nombre_separado = doc.nombre.split(' ')
        if(nombre_separado.size>1)
          if (nombre_separado.size == 2)
            primer_apellido = nombre_separado[0]
            primer_nombre = nombre_separado[1]
            f.puts "INSERT INTO conest2_development.usuario VALUES ('#{doc.cedula}',NULL,NULL,NULL,
                   '#{primer_nombre}',NULL,'#{primer_apellido}',NULL,'#{doc.correo}','#{doc.correo_2}',
                   '#{doc.clave}','#{doc.telefono}','#{doc.telefono_celular}',NULL,
                   '#{doc.fecha_hora_ultimo_login}','#{doc.observaciones}');"
          else
          .
          .
          .
```

Figura 7 Fragmento de código Ruby para generar scripts

- **Pruebas**

Para verificar la correctitud de la migración de los datos, se toma una muestra aleatoria de 30 expedientes curriculares (cinco por escuela), comparando esta información con los datos en CONEST 2.0, en donde se observa la integridad y consistencia de estos.

En la Tabla 3 se muestran la cantidad de registros que poseen algunas de las tablas de importancia en CONEST y CONEST 2.0, las cuales se utilizan para verificar la integridad de los datos migrados.

Tabla	CONEST	CONEST 2.0
Estudiante	36.256	36.256
Docente	648	648
Calificador Externo	377	377
Usuario	14	36.653
Historial Académico	891.592	890.844
Calificación	890.588	N/A

Tabla 3 Comparación de la cantidad de registros CONEST y CONEST 2.0

Como se aprecia en la tabla anterior, existen similitudes y diferencias significativas en los datos migrados. En el caso de *estudiante, docente* y *calificador externo,* se tiene una concordancia exacta en la cantidad de registros. En CONEST 2.0, la tabla *usuario* contiene los registros correspondientes a estudiante, docente, calificador externo y usuario (personal administrativo) de CONEST, eliminando las redundancias existentes entre estas tablas.

Referente a historial académico se observa una discrepancia en la cantidad de registros migrados, esto debido a inconsistencias presentes en CONEST, donde esta tabla y calificación deben coincidir, sin embargo no es el caso.

8.3.2. Iteración 1

En la iteración 1 se realizan diferentes modificaciones para optimizar el modelo de datos de CONEST 2.0. Para hacer una migración efectiva de estas modificaciones se debe realizar una depuración de los diferentes scripts que se crearon y generaron anteriormente.

- **Planificación**

En la Tabla 4 se especifican las historias de usuario a desarrollar en esta iteración.

Fecha Inicio: 27/04/09 Fecha Fin: 07/05/09

Nro	Fecha	Descripción
2.0	27/04/09	Modificaciones en el modelo de datos de CONEST 2.0.
1.1	28/04/09	Depurar y modificar scripts para realizar migración del modelo de datos CONEST a CONEST 2.0.
1.2	29/04/09	Depurar y modificar programa que genera scripts para realizar migración del modelo de datos CONEST a CONEST 2.0.

Tabla 4 Planificación Iteración 1

- **Codificación**

Debido a cambios en el modelo de datos de CONEST 2.0, es necesario efectuar modificaciones tanto en los scripts como en el programa, para que la migración se realice de forma correcta. Un ejemplo de ello, es la supresión del atributo tipo_status_sistema_id de las tablas docente y estudiante, agregándose a la tabla usuario, de donde es heredado. El código para este cambio es mostrado a continuación en la Figura 8.

```
INSERT INTO conest2_development.usuario
    SELECT cedula, tipo_status_sistema_id, tipo_estado_civil_id, tipo_sexo_id, tipo_nacionalidad_id,
           primer_nombre, segundo_nombre, primer_apellido, segundo_apellido, correo, correo_2, clave,
           telefono_actual, telefono_celular, fecha_nacimiento,fecha_hora_ultimo_login, observaciones
    FROM conest_development.estudiante;

INSERT INTO conest2_development.estudiante
    SELECT cedula,municipio_id_actual,estado_id_origen,direccion_actual,telefono_origen,
           direccion_origen, cohorte
    FROM conest_development.estudiante;
```

Figura 8 Código SQL para migrar las tablas usuario y estudiante

- **Pruebas**

Se realizan las mismas pruebas que en la iteración anterior, en donde la Tabla 5 tiene la cantidad de registros que poseen algunas de las tablas de importancia en CONEST y CONEST 2.0, las cuales se utilizan para verificar la integridad de los datos migrados.

Tabla	CONEST	CONEST 2.0
Estudiante	36.256	36.256
Docente	648	984
Calificador Externo	377	N/A
Usuario	14	36.653
Historial Académico	891.592	890.844
Calificación	890.588	N/A

Tabla 5 Comparación de la cantidad de registros CONEST y CONEST 2.0

Como se aprecia en la tabla anterior, existen similitudes y diferencias significativas en los datos migrados. En el caso de *estudiante,* se tiene una concordancia exacta en la cantidad de registros, sin embargo, en el caso de *docente*, se aprecia una diferencia relevante ya que los registros de *calificador externo* forman parte de la tabla *docente* en CONEST 2.0, eliminando redundancias en los datos ya que en CONEST se puede encontrar un mismo docente en ambas tablas.

8.3.3. Iteración 2

El objetivo de esta iteración es realizar las funciones administrativas con respecto a la entidad docente. Entre estas tenemos agregar y consultar un docente, así como consultar su historial en la facultad. Es importante destacar que al consultar un docente, se pueden realizar cambios sobre algunos de sus datos.

- **Planificación**

Las historias de usuario para esta iteración son presentadas a continuación en la Tabla 6.

Fecha Inicio: 18/05/09 Fecha Fin: 05/06/09

Nro	Fecha	Descripción
3.0	18/05/09	Agregar docente.
4.0	25/05/09	Agregar docente externo.
5.0	29/05/09	Consultar/ Modificar docente.
6.0	03/06/09	Consultar historial docente.

Tabla 6 Planificación Iteración 2

- **Diseño**

Para resolver esta iteración se propone el uso de un *autocomplete* (que complete tanto por nombres como por número de cédula), como primer paso para

la creación y consulta de un docente. De esta forma se pueden realizar diferentes verificaciones, tales como saber si ya el usuario existe en el sistema o si el número de cédula es válido, además reduce la carga cognitiva del administrador en el caso que el usuario ya exista y no tenga sus datos completos (Ver Figura 9).

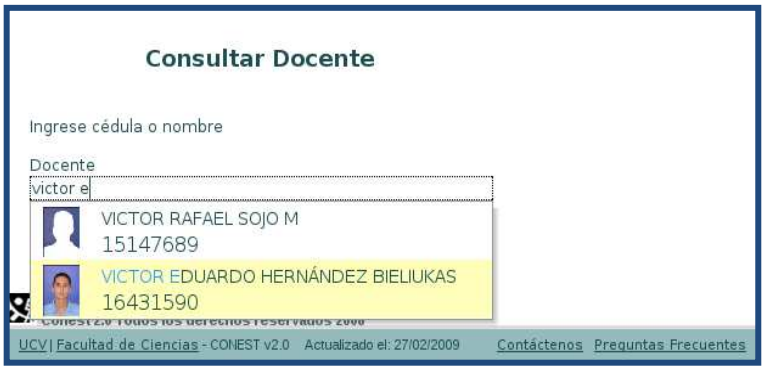

Figura 9 Autocomplete para buscar docente

Buscando reducir los pasos para crear un docente, tanto interno como externo, según el diseño de CONEST, se propone realizar un mismo formulario en donde se agreguen datos personales, datos laborales y cargos administrativos.

La consulta o modificación de los datos de un docente, se puede realizar desde una misma interfaz, en donde se muestre toda la información asociada al mismo, eliminando la necesidad de navegar entre diferentes páginas.

Para consultar el historial de un docente, se realizan vistas en donde esta información se despliegue en tablas.

- **Codificación**

Al momento de agregar un docente, existen dos posibilidades, agregar un docente interno o uno externo, para ello, se realizan dos funciones en el controlador encargadas de crear el registro para cada caso en la tabla docente.

Para consultar cualquier docente, se realiza un *autocomplete* para facilitar su búsqueda, luego una función se encarga de realizar la clasificación entre

docente externo e interno (ver Figura 10), direccionando a las acciones correspondientes encargadas de recopilar la información específica que es mostrada en las vistas de cada caso.

```
# Accion que verifica si un docente es
# externo o no y redirecciona a la funcion para
# consultar al correspondiente
#
# Parametros que necesita
# cedula del docente
#
def clasificar_docente
  session[:cedula] = params[:docente][:cedula]
  @docente = Docente.find(:first, :conditions => ['cedula = ?',session[:cedula]])
  if @docente
    if @docente.es_externo?
      bitacora("consultó su usuario docente externo",session[:cedula])
      redirect_to :action=>'consultar_externo'
    else
      bitacora("consultó su usuario de docente",session[:cedula])
      redirect_to :action=>'consultar_interno'
    end
  else
    flash[:mensaje]='El docente no existe'
    redirect_to :action => 'consultar'
  end
end
```

Figura 10 Código de la función para clasificar docentes

En el caso de consultar historial de docente, se desarrollan funciones en el modelo que realizan las búsquedas de las materias que ha dictado y coordinado, así como los seminarios y tesis en los que ha sido tutor y jurado. En la Figura 11 se muestra el código de la función *materias_periodo*, la cual busca todas las materias dictadas o coordinadas por un docente en un período específico.

```
# Metodo que devuelve la lista de materias que ha
# dado en un periodo academico
# Recibe como parametros
# * <tt>:periodo</tt> codigo de la materia
# ==== Retorno
# Arreglo de jurado examinador
#
def materias_periodo(periodo)
  jurado_examinador = JuradoExaminador.find(:all,
    :conditions => ['usuario_cedula = ? and periodo_academico_id = ? and tipo_jurado_id = ?',
    self.cedula, periodo,TipoJurado::PRINCIPAL])
  historiales = []

  jurado_examinador.each do |je|
    materia = MateriaEnPlan.find(:first,
      :conditions => ['materia_codigo = ? and tipo_materia_id not in (?)',
      je.materia_codigo, [TipoMateria::TESIS,TipoMateria::SEMINARIO]])

    if materia
      historial = HistorialAcademico.find(:first,
        :conditions => ['nombre_seccion =? and materia_codigo = ? and periodo_academico_id =?',
        je.nombre_seccion,je.materia_codigo,je.periodo_academico_id])
      if historial
        historiales << historial
      end
    end
  end
  return historiales
end
```

Figura 11 Código de función que busca materias dictadas por un docente

- **Pruebas**

En esta iteración se realizan pruebas funcionales para las todas las historias de usuario. Los escenarios realizados por cada historia de usuario son los listados a continuación:

- Para agregar docente:
  - Escenario 1: Proporcionar cédula por el *autocomplete*.
  - Escenario 2: Agregar datos del docente.
- Para agregar docente externo:
  - Escenario 1: Proporcionar cédula por el *autocomplete*.
  - Escenario 2: Agregar datos del docente externo.
- Para consultar docente:
  - Escenario 1: Proporcionar cédula por el *autocomplete*.
  - Escenario 2: Consultar datos del docente.
  - Escenario 3: Modificar datos del docente.
- Para consultar historial del docente:
  - Escenario 1: Proporcionar cédula por el *autocomplete*.
  - Escenario 2: Seleccionar tipo de consulta T.E.G.
  - Escenario 3: Seleccionar tipo de consulta Seminarios.
  - Escenario 4: Seleccionar tipo de consulta Materias.
  - Escenario 5: Consultar T.E.G como tutor.

- Escenario 6: Consultar T.E.G como jurado.
- Escenario 7: Consultar Seminarios como tutor.
- Escenario 8: Consultar Seminarios como jurado.
- Escenario 9: Consultar materias dictadas.

En la Figura 12 se muestra un ejemplo de una de las pruebas ejecutadas para crear docente, donde se puede observar tanto el escenario de éxito como el de fallo al momento de crear un docente, obteniendo, como el color verde lo indica, resultados exitosos en ambos casos.

```
Característica: Creacion de docente

  Escenario: Exito paso 1 de creacion (autocomplete)
    Dado que soy personal voy al controlador "docente/docente_nuevo" y accion "nuevo"
    Cuando lleno el campo "usuario[cedula]" con el valor "110011"
    Y presiono el boton "Continuar"
    Entonces veo la cadena "110011" en el contenido de la pagina

  Escenario: Fallo paso 1 creacion (Ya existe)
    Dado que soy personal voy al controlador "docente/docente_nuevo" y accion "nuevo"
    Cuando lleno el campo "usuario[cedula]" con el valor "13736933"
    Y presiono el boton "Continuar"
    Entonces veo la cadena "El docente ya existe" en el contenido de la pagina
```

Figura 12 Escenario 1 de creación de docente

Asimismo, como en el escenario mostrado en el ejemplo anterior, los resultados de las pruebas realizadas son exitosos.

8.3.4. Iteración 3

La iteración 3 se basa en cambios a la iteración anterior, en cuanto a la creación de un docente y a agregar nuevas acciones a la hora de consultarlo o modificarlo. Al momento de crear un docente, se dividen en dos fases, una desde el módulo administrativo, en donde sólo se colocan los datos primordiales y otra al momento de la autenticación del docente en el sistema, en donde debe proporcionar el resto de sus datos.

En cuanto a las acciones agregadas al momento de consultar un docente, se tiene el reiniciar la contraseña de un usuario, poder cambiar su estatus en el sistema, cambiar el tipo de docente y reenviar clave de calificación.

- **Planificación**

En la Tabla 7 se presenta la planificación por historia de usuario de esta iteración.

Fecha Inicio: 08/06/09 Fecha Fin: 26/06/09

Nro	Fecha	Descripción
7.0	08/06/09	Agregar docente desde módulo administrativo.
8.0	10/06/09	Agregar docente desde módulo docente.
9.0	12/06/09	Cambiar clave del sistema para un usuario.
10.0	15/06/09	Activar/desactivar estatus en sistema de un usuario.
11.0	16/06/09	Reenviar clave de calificación.
12.0	18/06/09	Reiniciar clave del sistema.
13.0	18/06/09	Cambiar tipo de docente (interno/externo).

Tabla 7 Planificación Iteración 3

- **Diseño**

Para agregar un docente desde el módulo administrativo sólo son necesarios datos específicos del mismo (nombre, apellido, correo y organización a la que pertenecerá), sin embargo, con el modelo actual de la base de datos no son suficientes estos datos para registrar un docente. Por lo tanto, se agregan tablas intermedias al modelo (ver Figura 13), que sirven de puente entre el agregar los datos desde el módulo administrativo y la creación del docente al momento que este agregue sus datos.

Las funcionalidades agregadas al consultar docente son enlaces que permiten realizar la acción correspondiente.

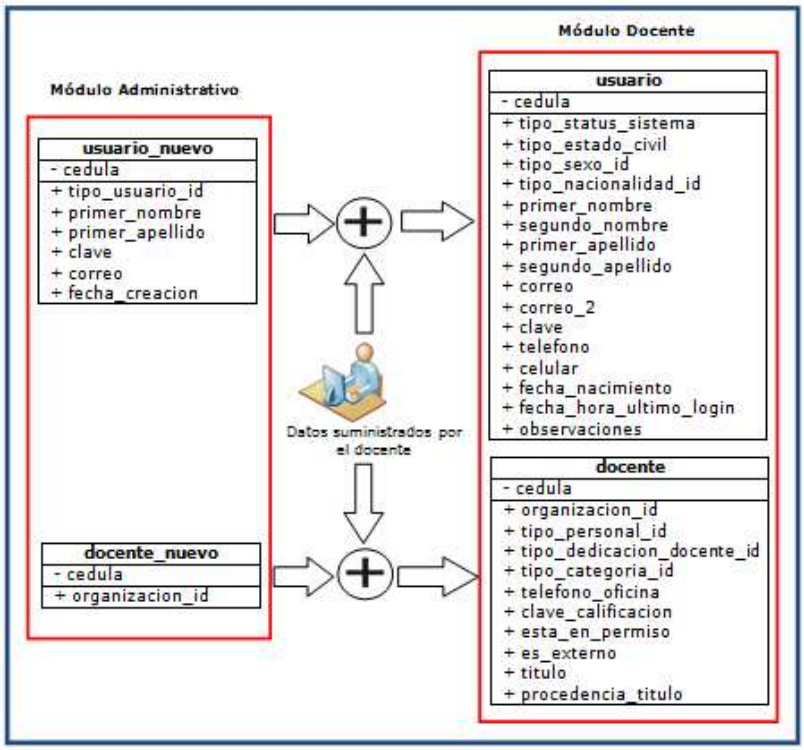

Figura 13 Diagrama para crear docente

Con respecto al cambiar clave, se realiza un formulario en donde se pide la clave anterior que tiene en el sistema, la clave nueva y su confirmación.

- **Codificación**

Para agregar un docente desde el módulo administrativo, se crea una función que registra los datos básicos (cédula, nombre, apellido, correo y organización) en tablas intermedias, y envía la clave de inicio de sesión al docente mediante un correo electrónico confirmando así su registro parcial en el sistema. Parte del código que realiza la creación del docente se presenta en la Figura 14.

```
def crear
  @usuario_nuevo = UsuarioNuevo.new
  @docente_nuevo = DocenteNuevo.new

  clave = session[:cedula].conest_cifrado

  DocenteNuevo.transaction do
    begin
      @usuario_nuevo.asignar_valores(session[:cedula],TipoUsuario::DOC,clave,params[:usuario_nuevo])
      @docente_nuevo.asignar_valores(session[:cedula],params[:docente_nuevo])
      @usuario_nuevo.save!
      @docente_nuevo.save!

      @titulo = "Información Importante - CONEST Facultad de Ciencias"
      @mensaje = "Bienvenido al Sistema CONEST de la Facultad de Ciencias, usted ahora es parte
      del grupo docente. Ingrese lo antes posible en la página www.conest.com para completar su registro.
      La forma de ingreso es con su número de cédula en usuario y clave"
      dest = @usuario_nuevo.correo
      EnviarCorreos.deliver_correo(dest,@titulo,@mensaje)
    rescue
      @usuario_nuevo.destroy
      @organizacion = Organizacion.find(:all, :conditions => ["id != ?",Organizacion::EXTERNA])
      flash[:mensaje] = "Error al guardar el docente"
      bitacora_error("error al guardar el docente",session[:cedula])
      render :action=>'agregar_datos'
      return
    end
  end
  flash[:mensaje] = "Docente guardado exitosamente"
  bitacora_alerta("Creó el docente",session[:cedula])
  redirect_to :action=>'nuevo'
end
```

Figura 14 Código para crear docente nuevo

Como siguiente paso para completar el registro, el docente debe ingresar en el sistema con la clave proporcionada. La función *crear,* se encarga de guardar los datos proporcionados por el docente, además de los que se encuentran almacenados en las tablas intermedias. De completarse con éxito el registro del docente, los datos en las tablas *docente_nuevo* y *usuario_nuevo* son eliminados.

- **Pruebas**

En esta iteración se realizan pruebas funcionales para las historias de usuario agregar docente nuevo desde el módulo administrativo, agregar docente desde el módulo docente y cambiar clave del sistema para un usuario. Los escenarios realizados por cada historia de usuario son los listados a continuación:

- Para agregar docente nuevo desde módulo administrativo:

  - Escenario 1: Proporcionar cédula por el *autocomplete*.

  - Escenario 2: Agregar datos del docente.

- Para agregar docente nuevo desde módulo docente:
  - Escenario 1: Autenticación de docente nuevo en el sistema.
  - Escenario 2: Agregar datos del docente.
- Para cambiar clave del sistema para un usuario:
  - Escenario 1: Autenticación de usuario en el sistema.
  - Escenario 2: Proporcionar claves.

En la Figura 15 se muestra un ejemplo de una de las pruebas ejecutadas para cambiar clave de un docente en el sistema, teniendo como resultado éxito en la prueba.

```
Característica: Cambio de clave

  Escenario: Exito como docente
    Dado que me autentique como "docente" con cedula "13736933" y clave "123123"
    Y estoy en el controlador "principal/principal" y accion "cambiar_clave"
    Cuando lleno el campo "clave_anterior" con el valor "123123"
    Y lleno el campo "clave" con el valor "111222333"
    Y lleno el campo "clave_confirmation" con el valor "111222333"
    Y presiono el boton "Cambiar"
    Entonces veo la cadena "Clave cambiada exitosamente" en el contenido de la pagina
```

Figura 15 Escenario 2 de cambiar clave

8.3.5. Iteración 4

El objetivo de esta iteración es realizar las funciones administrativas con respecto a la entidad estudiante. Entre estas tenemos agregar un estudiante nuevo desde el módulo administrativo, agregar un estudiante desde el módulo estudiante, agregar un estudiante completo desde el módulo administrativo y poder consultarlo. De igual forma que en la entidad docente, al consultar un estudiante, se pueden realizar modificaciones en ciertos datos.

- **Planificación**

En la siguiente la Tabla 8 se muestran las historias de usuario desarrolladas en esta iteración.

Fecha Inicio: 05/07/09 Fecha Fin: 17/07/09

Nro	Fecha	Descripción
14.0	05/07/09	Agregar estudiante nuevo desde módulo administrativo.
15.0	07/07/09	Agregar estudiante desde módulo estudiante.
16.0	10/07/09	Agregar estudiante completo desde módulo administrativo.
17.0	14/07/09	Consultar/ Modificar estudiante.

Tabla 8 Planificación Iteración 4

- **Diseño**

Para resolver esta iteración, al igual que en la iteración 2, se propone el uso de un *autocomplete* (que complete tanto por nombres como por número de cédula), como primer paso para la creación y consulta de un estudiante. De esta forma se pueden realizar diferentes verificaciones, tales como saber si ya el usuario existe en el sistema o si el número de cédula es válido, además reduce la carga cognitiva del administrador en el caso que el usuario ya exista y no posea sus datos completos.

Para agregar un estudiante nuevo desde el módulo administrativo, sólo son necesarios datos específicos del mismo (nombre, apellido, correo y organización a la que pertenecerá), por lo tanto se utiliza las tablas intermedias *estudiante_nuevo* y *usuario_nuevo* del modelo (ver Figura 16), que sirve de puente entre el agregar los datos desde el módulo administrativo y la creación del estudiante al momento que este agregue sus datos.

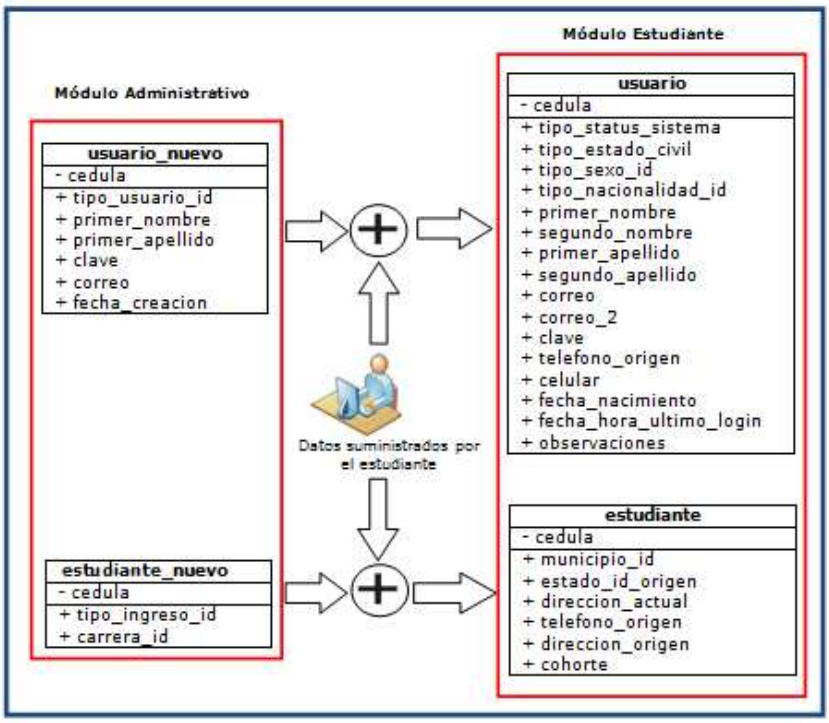

Figura 16 Diagrama para crear estudiante

Con respecto al agregar estudiante completo desde el módulo administrativo, se propone realizar un mismo formulario en donde se agreguen datos personales y datos académicos, esto en el caso que el estudiante deba ser registrado desde la DCE.

La consulta o modificación de los datos de un estudiante, se puede realizar desde una misma interfaz, en donde se muestre toda la información asociada al mismo, eliminando la necesidad de navegar entre diferentes páginas.

- **Codificación**

Para agregar un estudiante desde el módulo administrativo, se crea una función que registra los datos básicos (cédula, nombre, apellido, correo, tipo de ingreso y carrera) en tablas intermedias, y envía la clave de inicio de sesión al estudiante mediante un correo electrónico confirmando así su registro parcial en el sistema. Parte del código que realiza la creación del estudiante se presenta en la Figura 17.

```
def crear
  @usuario_nuevo = UsuarioNuevo.new
  @estudiante_nuevo = EstudianteNuevo.new

  clave = session[:cedula].conest_cifrado

  EstudianteNuevo.transaction do
    begin
      @usuario_nuevo.asignar_valores(session[:cedula],TipoUsuario::EST,clave,
      params[:usuario_nuevo])
      @estudiante_nuevo.asignar_valores(session[:cedula],params[:estudiante_nuevo])
      @usuario_nuevo.save!
      @estudiante_nuevo.save!
      @titulo = "Información Importante - CONEST Facultad de Ciencias"
      @mensaje = "Bienvenido al Sistema CONEST de la Facultad de Ciencias,
      usted ahora es parte del grupo estudiante. Ingrese lo antes posible
      en la página www.conest.com para completar su registro.
      La forma de ingreso es con su número de cédula en usuario y clave"
      dest = @usuario_nuevo.correo
      EnviarCorreos.deliver_correo(dest,@titulo,@mensaje)
    rescue
      @usuario_nuevo.destroy
      @carrera = Carrera.find(:all, :conditions => ["id != ?",Carrera::COOR])
      @tipo_ingreso = TipoIngreso.all
      flash[:mensaje] = "Error al guardar el estudiante"
      bitacora_error("error al guardar el estudiante",session[:cedula])
      render :action=>'agregar_datos'
      return
    end
  end
  flash[:mensaje] = "Estudiante guardado exitosamente"
  bitacora_alerta("Creó el estudiante",session[:cedula])
  redirect_to :action=>'nuevo'
end
```

Figura 17 Código crear estudiante nuevo

Como siguiente paso para completar el registro, el estudiante debe ingresar en el sistema con la clave proporcionada. La función *crear,* se encarga de guardar los datos proporcionados, además de los que se encuentran almacenados en las tablas intermedias. De completarse con éxito el registro, los datos en las tablas *estudiante_nuevo* y *usuario_nuevo* son eliminados.

Para el caso de registrar un estudiante completo desde el módulo administrativo, se tiene una función que se encarga de almacenar todos los datos del estudiante en las tablas correspondientes.

Asociado al consultar/modificar estudiante, se tiene una función encargada de la búsqueda de los datos que son mostrados al usuario. Debido a la gran cantidad de información mostrada en la página, y la necesidad de realizar cambios en los datos, se desarrolla una modularización de las funciones encargadas de guardar los datos personales, académicos (inscripción y calificación) y las deudas.

- **Pruebas**

En esta iteración se realiza una variante de la prueba de usabilidad Categorización de Contenido (Card Sorting), el cual permite tener una aproximación a como los usuarios esperan encontrar el contenido de la página.

Para esta prueba, se les proporciona un conjunto de tarjetas sin ningún orden, a cada usuario del sistema de la División de Control de Estudios según su rol, las cuales representaban cada sección de la funcionalidad consultar estudiante. La finalidad de esto, es establecer un orden en las tarjetas, según la prioridad y frecuencia de uso, para facilitar las anotaciones de los resultados, cada tarjeta fue identificada con una figura específica.

Esta prueba fue registrada en un formato, en donde se especifica el usuario, el orden de las figuras escogidas, las cuales tienen asociadas las funcionalidades datos personales, datos académicos, deudas, generar constancias, horario, resumen datos académicos, bitácora, verificar requisitos, datos egreso, información laboral, datos inscripción y datos preinscripción, así como el nivel de aceptación en un orden del 1 al 5 y diversas sugerencias. (Ver Anexos).

Los resultados son satisfactorios, ya que los siete usuarios del sistema a quienes se les realiza la prueba, catalogan la nueva forma de consultar estudiante, con un nivel 5 de aceptación. Además se logra establecer el orden de las secciones en esta funcionalidad, así como también se obtiene un conjunto de sugerencias que ayudan a mejorar la interacción entre el usuario y el sistema.

8.3.6. Iteración 5

En esta iteración se agregan funcionalidades en el requerimiento consultar estudiante, realizando tareas de los procesos, que permitan la comunicación entre la entidad estudiante y el proceso de inscripción.

- **Planificación**

En la Tabla 9 se presenta la planificación por historia de usuario de esta iteración.

Fecha Inicio: 27/07/09 Fecha Fin: 31/07/09

Nro	Fecha	Descripción
17.1	27/07/09	Modificar interfaz de consultar/modificar estudiante
18.0	27/07/09	Generar constancias y planillas, como el expediente curricular (kardex), la constancia de inscripción, planilla de notas y constancia de estudios.
19.0	27/07/09	Eliminar todas las materias inscritas de un estudiante en el período actual.
20.0	29/07/09	Modificar inscripción del período actual al estudiante consultado.
21.0	31/07/09	Mostrar materias preinscritas del estudiante consultado.

Tabla 9 Planificación Iteración 5

- **Diseño**

Para cumplir con los nuevos requerimientos planteados, se moldea una sección en la interfaz de consultar estudiante, para mostrar los diferentes enlaces que permiten generar las planillas y constancias del mismo.

En el caso de las tareas del proceso de inscripción (Ver Figura 18), se muestra una sección con la información de los diferentes períodos académicos que ha cursado el estudiante, mostrando un enlace para eliminar o reiniciar la inscripción sólo en el caso que el período mostrado sea el actual. Las modificaciones referentes a la inscripción (agregar materias nuevas, eliminar y retirar materias inscritas y cambiar secciones de materias), son realizadas en la

misma sección de inscripción. Asimismo, se realiza otra sección para mostrar los datos de las materias preinscritas y el estatus que éstas tienen.

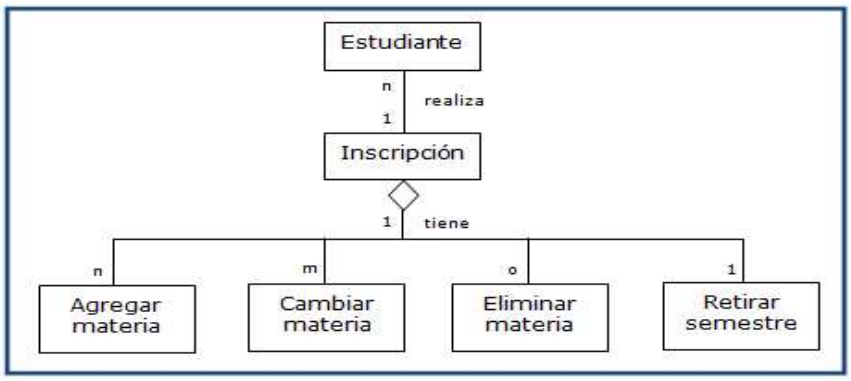

Figura 18 Diagrama de Objetos del Dominio entre estudiante e inscripción

Tomando en cuenta las sugerencias proporcionadas por los usuarios de la DCE, se incluye la información laboral y los datos de ingreso, en la sección de datos personales, así como también se modifica la interfaz, agregando un *acordeón* para mostrar la información del estudiante por secciones, como se muestre en la figura 19. El orden de las secciones mostradas, es producto de la prueba antes realizada al personal de la DCE.

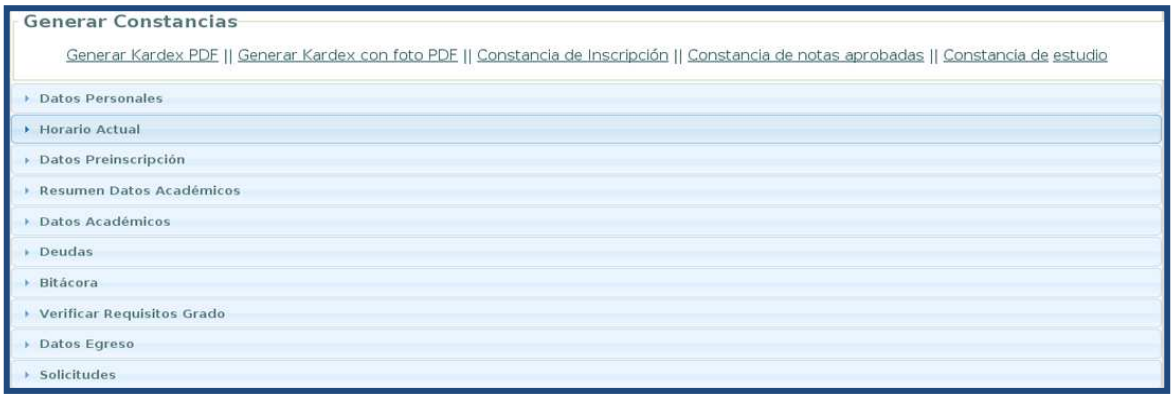

Figura 19 Diseño de interfaz con *acordeón*

- **Codificación**

Para cumplir con las historias de usuario referentes a la inscripción se desarrolla una función por cada acción que se puede realizar. Todas estas funciones se comunican con un método específico del proceso *inscripcion,* por ejemplo, en la Figura 20 se muestra el código que cambia la sección de una materia para un estudiante.

```
secciones.each do |seccion|
  if seccion != ""
    proceso = proceso_inscripcion
    proceso = proceso.cambiar_seccion({
      :carrera_id => session[:estudiante_inscripcion][i].carrera_id,
      :plan_nombre => session[:estudiante_inscripcion][i].plan_nombre,
      :nombre_seccion => seccion,
      :materia_codigo => session[:estudiante_inscripcion][i].materia_codigo,
      :periodo_academico_id => session[:estudiante_inscripcion][i].periodo_academico_id,
      :estudiante_cedula => session[:cedula]
    })
  end
  i += 1
end
```

Figura 20 Código para cambiar sección de una materia

Por cada sección a ser modificada, se crea una instancia del proceso *proceso_inscripcion* para luego llamar a su función *cambiar_seccion* proporcionando una lista con los parámetros necesarios.

- **Pruebas**

Para comprobar la eficacia de la comunicación entre el proceso *inscripción* y las diferentes funciones desarrolladas para esta iteración, se realizan a través de la interfaz de usuario un conjunto de pruebas aleatorias, cambiando, agregando y eliminando materias del período académico actual, para luego constatar dichos cambios en base de datos.

8.3.7. Iteración 6

En esta iteración se sigue ampliando y modificando la iteración 4 en cuanto a la consulta y modificación del estudiante. Se agregan acciones administrativas en

cuanto a las deudas que un estudiante posee, su tipo de ingreso, su horario actual y el enviar correos personalizados.

- **Planificación**

En la siguiente la Tabla 10 se muestran las historias de usuario desarrolladas en esta iteración.

Fecha Inicio: 03/08/09 Fecha Fin: 17/08/09

Nro	Fecha	Descripción
22.0	03/08/09	Modificar tipo de ingreso en estudiante al consultarlo.
23.0	05/08/09	Agregar estudiante a la opción docente.
24.0	06/08/09	Enviar correo personalizado al estudiante consultado.
25.0	07/08/09	Administrar deudas del estudiante consultado.
26.0	12/08/09	Mostrar horario actual del estudiante consultado.

Tabla 10 Planificación Iteración 6

- **Diseño**

Para agregar un estudiante a la opción docente se propone mostrar un enlace a los estudiantes que cumplan con los requisitos y que pertenezcan a las carreras permitidas en este convenio (Biología, Física, Matemática y Química).

Para enviar un correo personalizado al estudiante, se proporciona un enlace que despliega una ventana en donde se escribe el asunto y cuerpo del mensaje a enviar.

En el caso de administrar las deudas de un estudiante, se realiza una sección en donde se muestran las deudas que posee, así como la posibilidad de eliminarlas o agregar nuevas deudas bien sean administrativas, biblioteca, bolsa del libro, etc. (Ver Figura 21).

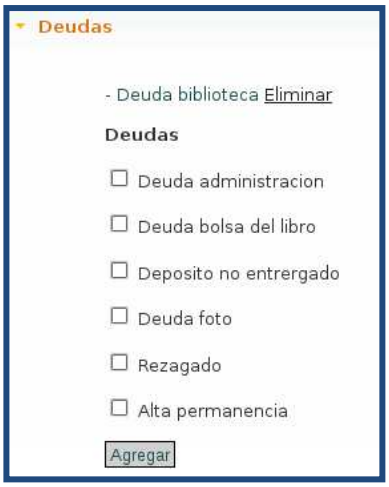

Figura 21 Diseño de la interfaz de la sección de deudas

El horario es mostrado en una sección de la interfaz de usuario, en forma de calendario semanal, indicando el día y hora de las materias inscritas por el estudiante en el semestre actual.

- **Codificación**

Se desarrolla una función que realiza la búsqueda del horario actual de un estudiante según la carrera en la que esté inscrito, la cual retorna un arreglo de objetos del tipo *HorarioAsignado.* Por cada elemento de este arreglo, se invoca a otra a otra función encargada de dar el formato necesario para ser mostrado en la vista. El código de esta función es mostrado en la Figura 22.

```
def horario_actual(carrera)
  horario_actual=[]
  historial_actual = HistorialAcademico.find(:all,
    :conditions=>['estudiante_cedula = ? and periodo_academico_id=? and carrera_id=?',
    cedula,PeriodoAcademico.periodo_academico_actual.id,carrera.carrera_id ])
  historial_actual.each do |historial|
    horario_actual = horario_actual + historial.materia.horario_actual(historial.nombre_seccion)
  end
  return horario_actual
end
```

Figura 22 Código de la función horario actual

- **Pruebas**

Se realizan pruebas unitarias a la función del modelo estudiante *horario_actual,* verificando que los datos obtenidos por esta función sean los correctos, obteniendo resultados satisfactorios.

8.3.8. Iteración 7

En esta iteración se agregan funcionalidades en el requerimiento consultar estudiante, realizando tareas de los procesos, que permiten la comunicación entre la entidad estudiante y el proceso de calificación. También se realiza la tarea del proceso calificación entre la entidad docente y la calificación.

- **Planificación**

Las historias de usuario para esta iteración son presentadas a continuación en la Tabla 11.

Fecha Inicio: 18/08/09 Fecha Fin: 28/08/09

Nro	Fecha	Descripción
27.0	18/08/09	Modificar nota de un estudiante en cualquier período académico.
28.0	20/08/09	Retirar una materia específica de un estudiante.
29.0	21/08/09	Agregar materias por equivalencias a un estudiante en cualquier período académico.
30.0	24/08/09	Retirar materias en cualquier período académico.
31.0	25/08/09	Verificar requisitos de grado de un estudiante.
32.0	25/08/09	Calificar una sección de un docente.

Tabla 11 Planificación Iteración 7

- **Diseño**

Debido a que la interfaz requerida para esta iteración es similar a la que se utiliza en los datos de inscripción, se propone hacer una combinación de esta sección, en donde se muestran tanto los datos de inscripción como los de calificación por cada período académico que sea seleccionado. También se agrega un enlace que permite agregar materias por equivalencia. (Ver Figura 23).

Figura 23 Diseño de interfaz para inscripción y calificación

Por cada materia que se muestra, se agregan dos columnas para colocar las modificaciones de las notas del estudiante, una para la nota final y otra para la nota de reparación. Asimismo, se coloca otra columna que indica si se desea retirar una materia en particular.

En cuanto a la verificación de requisitos del estudiante, se agrega otra sección que muestra la relación de la cantidad de materias y número de créditos que han sido aprobados por el estudiante y los necesarios para poder graduarse según la carrera y mención elegidas, así como también un enlace que despliega una ventana mostrando un listado de las materias obligatorias que aún le falta aprobar, ejemplo de esto es mostrado en la Figura 24.

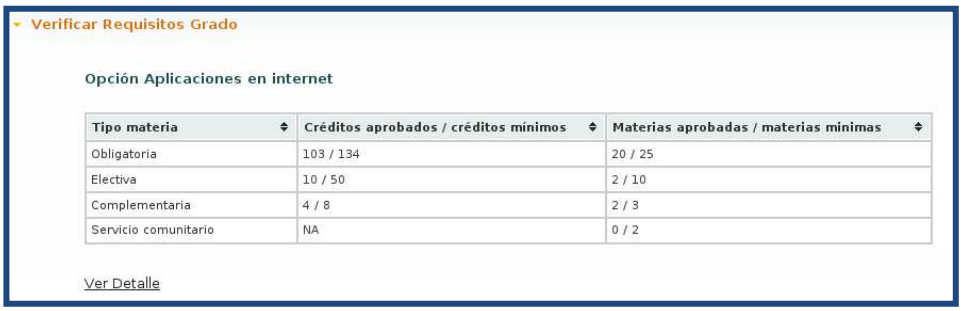

Figura 24 Diseño de interfaz de Verificar Requisitos

Desde el consultar docente en el módulo administrativo, se permite calificar y modificar la calificación para una materia determinada. Para esto, se toma como referencia la interfaz utilizada en CONEST, mostrando los datos de los estudiantes de esa sección y dos columnas para colocar las notas correspondientes, una para su nota final y otra para la de reparación. Se toma en consideración la posibilidad de guardar temporalmente las notas o calificar de forma definitiva la sección.

- **Codificación**

Para cumplir con las historias de usuario referentes a la calificación se desarrolla una función por cada acción que se puede realizar. Todas estas funciones se comunican con un método específico del proceso *calificacion,* por ejemplo, en la Figura 25 se muestra el código que modifica la calificación una materia para un estudiante.

```
materias=[]
notas=[]
finales_anterior.each do |final|
  if (final != nil and finales[i]!=final)or(final == nil and finales[i]!="")or (reparaciones_anterior[i] != nil
    if validar_notas?(session[:estudiante_inscripcion][i].materia_en_plan,finales[i],reparaciones[i])
      materias << {
        :plan_nombre=> session[:estudiante_inscripcion][i].plan_nombre,
        :carrera_id => session[:estudiante_inscripcion][i].carrera_id,
        :periodo_academico_id => session[:estudiante_inscripcion][i].periodo_academico_id,
        :estudiante_cedula => session[:cedula],
        :materia_codigo => session[:estudiante_inscripcion][i].materia_codigo,
        :nombre_seccion => session[:estudiante_inscripcion][i].nombre_seccion}
      notas << {:tipo_nota_id_final => finales[i],:tipo_nota_id_reparacion => reparaciones[i]}
    else

    end
  end
  i+=1
end
proceso = proceso_calificacion
proceso = proceso.calificar(materias,notas)
```

Figura 25 Código para cambiar calificación

Para modificar las notas, se crea una única instancia del proceso *proceso_calificacion* para luego llamar a su función *calificar,* proporcionando como parámetros dos arreglos de listas, uno que contiene las materias y otro con las nuevas notas.

En cuanto a la historia de usuario verificar requisitos de grado, se desarrolla una función que busca la cantidad de créditos y materias aprobadas de un estudiante, esta información es utilizada al momento de mostrar la relación entre la cantidad aprobada y la cantidad requisito.

Para ver en detalle las materias obligatorias no aprobadas, se crea una función en la que se busca las materias obligatorias en el plan activo de la carrera cursada. Por cada obligatoria, se comprueba si no ha sido aprobada por el estudiante, para agregarla a un arreglo que es recorrido y mostrado en la vista. Parte de este código es mostrado en la Figura 26.

```
def detalles_requisitos
  @estudiante = Estudiante.find(:first,
    :conditions => ['cedula = ?', session[:cedula]])
  @opcion_carrera = OpcionCarrera.find(:first,
    :conditions => ['id = ?', params[:opcion]])

  @obligatorias_por_aprobar = []
  plan_activo = PlanActivo.find(:first, :conditions => ['carrera_id = ?',
    @opcion_carrera.carrera_id])
  materias_opcion = @opcion_carrera.materias_opcion_carrera(@opcion_carrera.carrera_id)
  obligatorias = plan_activo.plan.obligatorias_plan(@opcion_carrera.carrera_id)

  #Se suman las obligatorias del plan con las obligatorias de la mencion
  obligatorias = obligatorias + materias_opcion
  obligatorias = obligatorias.uniq

  proceso = proceso_inscripcion
  obligatorias.each do |obligatoria|
    if !proceso.estudiante_aprobo_materia?({
      :estudiante_cedula => session[:cedula],
      :carrera_id => @opcion_carrera.carrera_id,
      :plan_nombre => plan_activo.plan_nombre,
      :materia_codigo => obligatoria.codigo})
      @obligatorias_por_aprobar << obligatoria
    end
  end
  render :layout => false
end
```

Figura 26 Código para buscar las materias obligatorias por aprobar

- **Pruebas**

Se realizan pruebas unitarias a la función del modelo estudiante *tipo_creditos_aprobados*, como también la del modelo plan *obligatorias_plan*, verificando que los datos obtenidos por estas funciones sean los correctos, obteniendo resultados satisfactorios.

Además para comprobar la eficacia de la comunicación entre el proceso calificación y las diferentes funciones desarrolladas para esta iteración, se realizan a través de la interfaz de usuario, un conjunto de pruebas aleatorias, calificando, retirando y agregando materias del período académico actual, para luego constatar dichos cambios en base de datos.

Conclusiones

El objetivo de este Trabajo Especial de Grado se cumplió satisfactoriamente, ya que se logró realizar la automatización de la migración de datos del sistema CONEST al sistema CONEST 2.0 manteniendo la integridad y consistencia de los datos. También se aplicaron estándares de programación y se implementó el patrón de diseño MVC; como consecuencia, se obtuvo un código legible y ordenado, que facilitará a futuros programadores continuar con el mantenimiento y evolución del sistema.

Tomando como referencia las acciones del sistema CONEST asociadas a los estudiantes y docentes, se desarrollaron funcionalidades, que cumplen con la gestión de estas entidades de una forma más optimizada, gracias a que se tomó en consideración la experiencia anterior en cuanto a la programación, base de datos, estándares, código y diseño de la interfaz de usuario.

Se utilizó como método de desarrollo una adaptación de Programación Extrema, la cual se ajustó a las condiciones de trabajo, ya que los requerimientos del sistema cambiaron constantemente en el tiempo y por cada entrega realizada. El trabajo en pareja es una de las ventajas ofrecidas por XP, la cual permite una mejor comprensión y solución del problema y un desarrollo rápido del código, además de realizar constantemente revisiones al trabajo que se está realizando, pudiendo incorporar mejoras. Otra de las ventajas, es la constante integración de código con el fin de evitar posibles fallas o conflictos. Para esto, se utiliza la herramienta Subversion (SVN), la cual de una manera sencilla, permite hacer actualizaciones e integraciones del código entre varios grupos de trabajo.

Además se utiliza la librería de *javascript*, denominada *jQuery*, la cual apoya la interacción de la aplicación con nuevos efectos y funciones que persiguen los beneficios de la web 2.0. El uso de esta librería, sumada al manejo de *Ajax*, mejora la usabilidad del sistema al reducir la carga cognitiva, prevenir errores,

proveerle seguridad y confianza al usuario, además de reducir el tiempo de respuesta de la aplicación.

A pesar de que las versiones de CONEST realizan la misma gestión administrativa, debido al cambio en la lógica de negocio en CONEST 2.0, no se pudo reutilizar el código existente, sin embargo, se tomó como ejemplo en varias oportunidades durante el desarrollo del sistema.

La implementación de las pruebas funcionales automatizadas mediante el uso de la herramienta Cucumber, son desarrolladas por primera vez en el sistema CONEST en este trabajo de investigación, sirviendo de aporte para futuros programadores, ya que es un proceso sencillo, en donde se realizan pruebas automáticamente de todos los posibles casos y funciones, ahorrando tiempo y esfuerzo.

Este Trabajo Especial de Grado contribuye a través de las TIC con la gestión administrativa de la División de Control de Estudios, haciendo que las actividades diarias sean más sencillas, con una reducción de tiempo y esfuerzo. Esta investigación es el inicio para continuar con el desarrollo de las otras entidades y procesos necesarios en el sistema y así seguir proponiendo soluciones a la gestión académica de nuestra Facultad.

Anexos

Desde la Figura 27 hasta la Figura 38, son mostradas las tarjetas elaboradas para la prueba de usabilidad Categorización de Contenido, las cuales representan las secciones en las que se encuentra dividida la funcionalidad consultar estudiante.

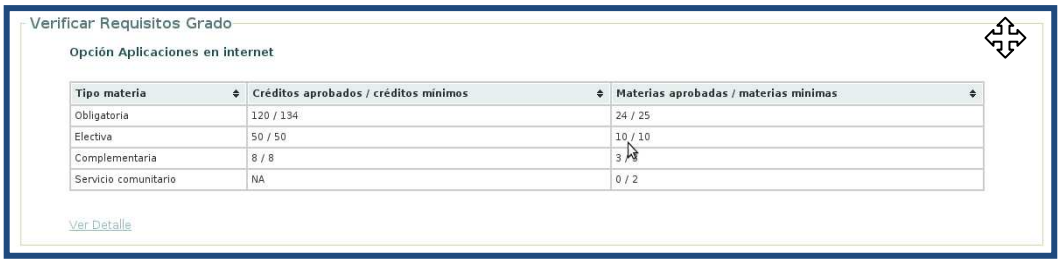

Figura 27 Verificar Requisitos Grado CONEST 2.0

Figura 28 Datos Ingreso CONEST 2.0

Figura 29 Deudas CONEST 2.0

Figura 30 Bitácora CONEST 2.0

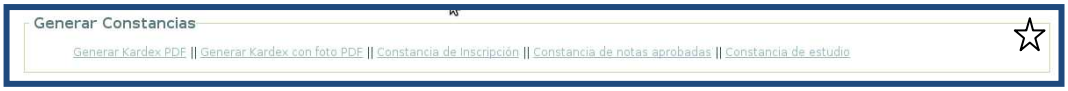

Figura 31 Generar Constancias CONEST 2.0

Figura 32 Datos Preinscripción CONEST 2.0

Figura 33 Información Laboral CONEST 2.0

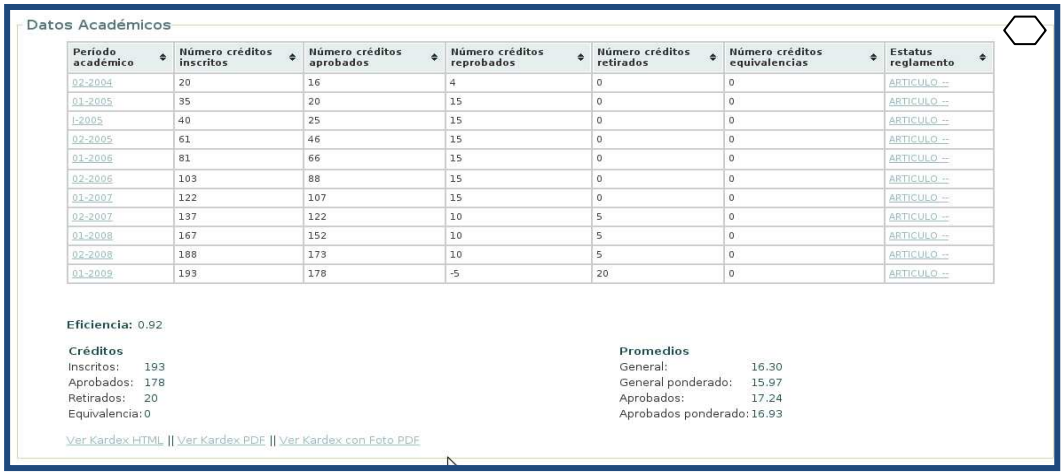

Figura 34 Datos Académicos CONEST 2.0

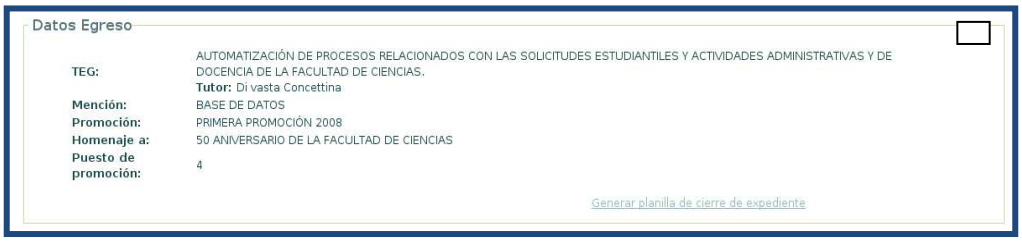

Figura 35 Datos Egreso CONEST 2.0

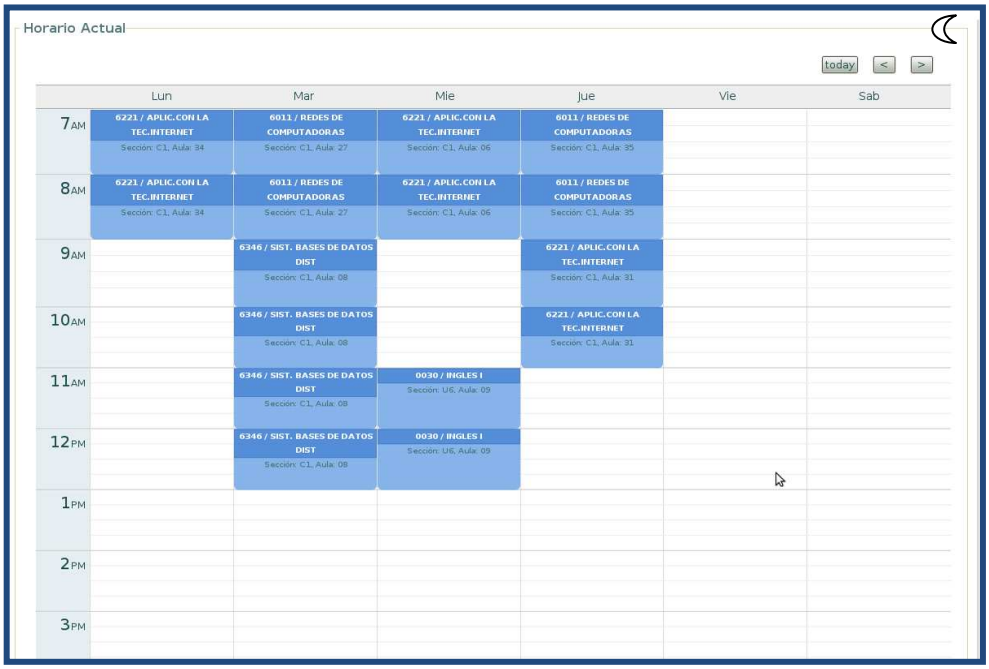

Figura 36 Horario Actual CONEST 2.0

Figura 37 Resumen Datos Académicos

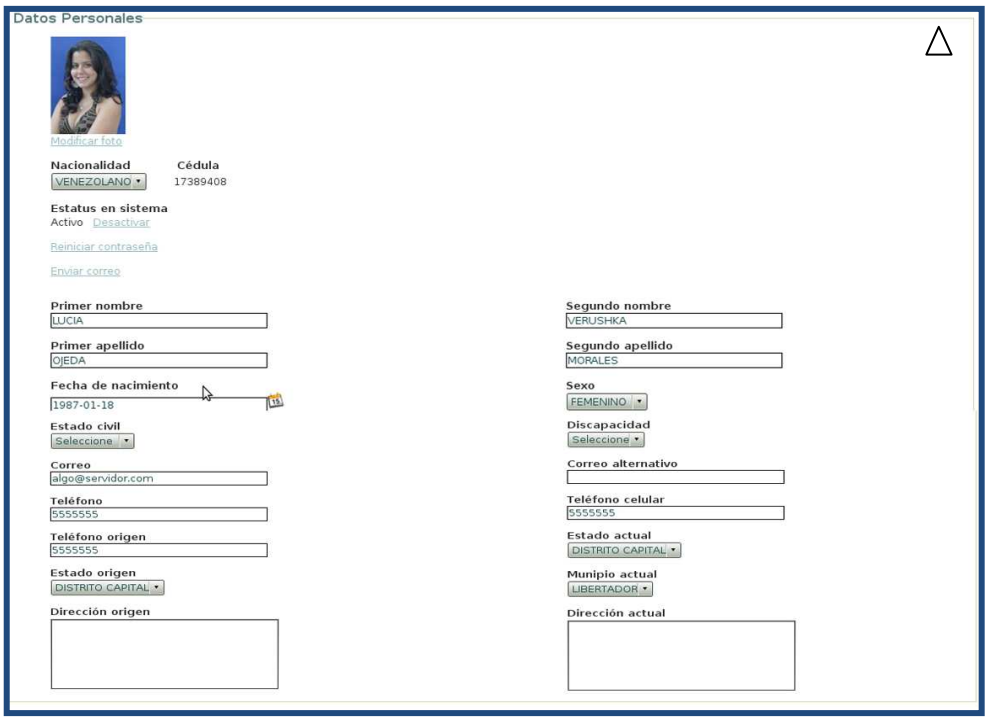

Figura 38 Datos Personales CONEST 2.0

En la Tabla 12 se tienen los datos obtenidos de la prueba de Categorización de Contenido realizada a parte del personal de la DCE, en donde se identifica el orden o prioridad escogida por cada uno de los usuarios del sistema, así como su nivel de aceptación, el tiempo empleado por cada uno de ellos y diferentes sugerencias que son tomadas en cuenta para la mejora del sistema.

Entrevistado		Prioridad			Tiempo	Aceptación	Sugerencias
Usuario 1	1	△	⇦	⇧	12 min	5	Mostrar datos personales solo si es necesario
	2	☆	☾	⇨			
	3	⧖	⬡	⇩			
	4	○	✥	□			
Usuario 2	1	△	⇦	⇧	12 min	5	Dividir información con pestañas
	2	☆	☾	⇨			
	3	⧖	⬡	⇩			
	4	○	✥	□			
Usuario 3	1	△	⇦	⇧	12 min	5	Agregar a datos personales los datos de ingreso
	2	☆	☾	⇨			
	3	⧖	⬡	⇩			
	4	○	✥	□			
Usuario 4	1	△	⇦	⇧	20 min	5	Agregar a datos personales los datos laborales
	2	☆	☾	⬡			
	3	⧖	⇨	○			
	4	⇩	✥	□			
Usuario 5	1	☆	△	☾	5 min	5	Horario desplegable
	2	⇩					
	3						
	4						
Usuario 6	1	☆	△	☾	5 min	5	
	2	⇩					
	3						
	4						
Usuario 7	1	△	☆	✥	5 min	5	
	2	□					
	3						
	4						

Tabla 12 Datos de la Prueba Categorización de Contenido

Referencias Bibliográficas

- (Acerca de Ruby, s.f.). Acerca de Ruby. Consultado el 24 de septiembre de 2008 en http://www.ruby-lang.org/es/about/

- (ALEGSA, 2009). Diccionario de definiciones. Revisado el día 15 de septiembre de 2009 en http://www.alegsa.com.ar/Dic/dom.php

- (Antelo, E., 2004). Diseño de Aplicaciones en Tres Capas. Consultado el 21 de septiembre de 2008 en http://www.geocities.com/trescapas/TresCapas.htm

- (Aplicaciones Web, 2007). Consultado el 06 de septiembre de 2008 en http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_Web

- (Application Programming Interface, 2008). Consultado el 24 de septiembre de 2008 en http://es.wikipedia.org/wiki/API

- (Arquitectura Cliente/Servidor, s.f). Consultado el 09 de septiembre de 2008 en http://www.csi.map.es/csi/silice/Global71.html

- (Brando, 2008). Ruby on Rails 2.1. ¿Qué hay de nuevo?. Consultado el 01 de diciembre de 2008 en http://www.improveit.com.br/en/company/tapajos

- (Código abierto, 2009). Consultado el día 27 de enero de 2009 en http://www.eseguridad.gob.mx/wb2/eMex/eMex_Glosario_de_terminos_Seguridad?page=6

- (Complemento, 2008). Consultado el 24 de septiembre de 2008 en http://es.wikipedia.org/wiki/Add-on

- (Cucumber, 2008). Cucumber, Making BDD fun. Consultado el día 07 de octubre de 2009 en http://www.cukes.info/

- (Di Campoy, 1999). Tutorial de Base de Datos I. Consultado el 10 de septiembre de 2008 en http://sistemas.itlp.edu.mx/tutoriales/basedat1/tema1_9.htm

- (DIMAGIN Web Development, 2007). Principales ventajas de las aplicaciones Web. Consultado el 09 de septiembre de 2008 en http://www.dimagin.net/es/contenido.php?t_id=6

- (Grupo de desarrolladores CONEST, 2009). Estándares de Código de CONEST. Revisión 1.0.

- (Eguiluz, J, 2009 a). Introducción a XHTML. Consultado el 12 de septiembre de 2008 en http://www.librosWeb.es

- (Eguiluz, J, 2009 b). Introducción a CSS. Consultado el 13 de septiembre de 2008 en http://www.librosWeb.es

- (Eguiluz, J, 2009 c). Introducción a JavaScript. Consultado el 12 de septiembre de 2008 en http://www.librosWeb.es

- (Figueroa, 2008). Frameworks MVC de desarrollo Web. Consultado el 22 de septiembre de 2008 en http://blog.buhoz.net/blog1.php/2008/03/06/frameworks-mvc-de-desarrollo-web

- (Froufe, A., 1997). Tutorial de Java - Arquitectura Modelo/Vista/Controlador. Consultado el 03 de septiembre de 2008 en http://www.cica.es/formacion/JavaTut/Apendice/arq_mvc.html

- (Internet Didáctica, 2007). Glosario de Internet. Revisado el día 15 de septiembre de 2009 en http://www.internet-didactica.es/glosario_internet/glosario_internet_s.php

- (Jeffries, 2001). Xtreme Programming an Agile Software Development Resource. Consultado el día 07 de octubre de 2009 en http://xprogramming.com/xpmag/whatisxp

- (Kioskea, 2008). Redes- Arquitectura Cliente/Servidor. Consultado el 09 de septiembre de 2008 en http://es.kioskea.net/cs/csintro.php3

- (Menéndez, 2000). Servidores de Base de Datos. Consultado el día 27 de enero de 2009 en http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/sgbd.html

- (Metaprogramación, 2009). Metaprogramación. Consultado el día 8 de octubre de 2009 en http://es.wikipedia.org/wiki/Metaprogramaci%C3%B3n

- (Modelo Vista Controlador, 2007). Modelo Vista Controlador. Consultado el 08 de septiembre de 2008 en http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

- (O' Reilly, 2005). ¿Qué es la Web 2.0? Patrones del diseño y modelos de negocio para la siguiente generación de software. Consultado el día 20 de septiembre en http://oreilly.com/web2/archive/what-is-web-20.html

- (Palasí, V., 2003). Buenas Prácticas en J2EE. Segunda Parte. Consultado el 10 de septiembre de 2008 en http://espanol.aurumsol.com/articulos/art7/art7-8.html

- (Pecos, D., 2002). PostGreSQL vs. MySQL. Consultado el 16 de septiembre de 2008 en http://www.netpecos.org/docs/mysql_postgres/x108.html

- (Pressman, 2007). Pressman, R. Ingeniería del Software. Un enfoque práctico. Sexta Edición. Editorial Mc Graw Hill.

- (The Pragmatic Programmers LLC, 2007). Agile Web Development with Rails. Consultado el 16 de septiembre de 2008 en http://media.pragprog.com/titles/rails2/DepotAjax.pdf

- (Rails Documentation, 2008). Consultado el 27 de enero de 2009 en http://api.rubyonrails.org/

- (Rivera, 2009). Consultado el 04 de febrero de 2009 en http://www.asesoriainformatica.com/definiciones_c.htm

- (Ruby on Rails, 2008). Consultado el 20 de septiembre de 2008 en http://es.wikipedia.org/wiki/Ruby_on_Rails

- (Seguridad Informática, s. f.). Consultado el 27 de enero de 2009 en http://www.seguridadinformatica.dcyc.ipn.mx/glosario.html

- (Servidor HTTP Apache, 2007). Consultado el 20 de septiembre de 2008 en http://es.wikipedia.org/wiki/Servidor_HTTP_Apache

- (Sobre Rails, s.f.). Consultado el 24 de septiembre de 2008 en http://sobrerailes.com/articles/2006/01/29/extensiones-para-rails-plugins-componentes-y-engines

- (SQL, 2009). Consultado el día 27 de enero de 2009 en http://es.wikipedia.org/wiki/Celda_activa

- (Summerville, 2005). Ingeniería del Software. Séptima edición. Editorial Pearson Educación S.A.

- (Tutorial jQuery, 2007). Tutorial de jQuery. Consultado el día 15 de septiembre de 2009 en http://www.cristalab.com/tutoriales/tutorial-de-jquery-c214l/

- (UCV, 2009). Plan Estratégico de la Universidad Central de Venezuela. Consultado el 6 de octubre de 2009 en http://www.ucv.ve/sobre-la-ucv/resena-organizacional/plan-estrategico-ucv.html

- (Van Der Henst, 2005). ¿Qué es la Web 2.0? Consultado el día 20 de septiembre de 2009 en http://www.maestrosdelweb.com/editorial/web2/

- (W3C, 2008 a). Sobre el W3C. Consultado el día 26 de febrero de 2009 en http://www.w3c.es/Consorcio/

- (W3C, 2008 b). Guía breve de XHTML. Consultado el 21 de septiembre de 2008 en http://www.w3c.es/Divulgacion/Guiasbreves/XHTML

- (W3C, 2008 c). El W3C de la A a la Z. Consultado el 26 de febrero de 2009 en http://www.w3c.es/divulgacion/a-z/#xml