UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN
INGENIERÍA DE SOFTWARE Y SISTEMAS

Prototipo de un proceso para la extracción, transformación y visualización de índices académicos desde la base de datos de CONEST

Trabajo Especial de Grado presentado
ante la ilustre
Universidad Central de Venezuela por el Bachiller
Renny Alexander Hernández Gudiño
para optar al título de
Licenciado en Computación.

Prof. Nora Elena Montaño Fermín

Prof. Andrés Castro

Caracas, Venezuela

Mayo 2010

Dedicatoria

A mis padres Rosa y Valentín.

Agradecimiento

Agradezco a mis tutores, la profesora Nora Montaño y el profesor Andrés Castro por su apoyo y paciencia.

A los profesores Jossie Zambrano, Eugenio Scalise y Antonio Silva por todo su apoyo y colaboración.

Al grupo docente del Centro ISYS.

A mi amigo Oskar Raul Cahueñas por su apoyo y colaboración.

Al profesor José Páez por la asesoría prestada en el estudio de los índices académicos.

A mis padres y hermanos por su apoyo y paciencia.

Índice general

1. Introducción **1**

2. Marco Conceptual **7**

2.1. Visualización de Datos 7

2.1.1. Elementos de la visualización de datos 9

2.1.2. Terminología Básica en la Visualización 11

2.1.3. Ciclo de Vida de las visualizaciones de datos 11

2.1.4. Interfaces Web para la Visualización 17

2.2. Ruby On Rails 18

2.3. Rails y el Modelo MVC 21

2.3.1. Active Record: El ORM de Rails. 22

2.3.2. La Vista y el Controlador. 24

2.4. Servicios Web en Rails 25

2.4.1. Action Web Service 26

2.5. Desarrollo Rápido de Aplicaciones 29

2.5.1. Fases de RAD 31

2.5.2. Características del modelo RAD 32

3. Marco Aplicativo **37**

3.1. Aplicación del método de desarrollo RAD 37

3.1.1. Fases a implementar 37

3.1.2. Herramientas a utilizar 38

3.1.3. Revisiones con el usuario 38

3.2. Iteracíón 0 . . . 38
3.2.1. Requerimientos . . . 39
3.2.2. Diseño . . . 39
3.3. Iteración 1 . . . 40
3.3.1. Requerimientos . . . 40
3.3.2. Diseño . . . 42
3.3.3. Generación de la aplicación. . . . 43
3.4. Iteración 2 . . . 46
3.4.1. Planificación de Requerimientos . . . 47
3.4.2. Diseño . . . 47
3.4.3. Generación de la aplicación . . . 48
3.5. Iteración 3 . . . 53
3.5.1. Requerimientos . . . 53
3.5.2. Diseño . . . 54
3.5.3. Generación de la aplicación . . . 55
3.6. Iteración 4 . . . 57
3.6.1. Requerimientos . . . 57
3.6.2. Diseño . . . 58
3.6.3. Generación de la aplicación . . . 59
3.6.4. Implementación y Pruebas . . . 62
3.7. Iteración 5 . . . 63
3.7.1. Requerimientos . . . 63
3.7.2. Diseño . . . 63
3.7.3. Generación de la aplicación . . . 65
3.8. Iteración 6 . . . 66
3.8.1. Implementación y pruebas . . . 66

4. Conclusiones **67**

A. Generación de Código **71**
A.1. Actualizar datos . . . 71

A.2. Servicio Web REST 76
A.3. Función render_viz 80
A.4. Modelo E/R Final 81

Índice de figuras

1.1. Modelo de Negocios 3

2.1. Infográfico: Códigos www por país. 8
2.2. Infográfico: Modelo de Usuarios flickr.com 9
2.3. Estructura básica de la arquitectura MVC 22
2.4. Enfoque del modelo RAD 30
2.5. Relación entre las cajas de tiempo(*timeboxes*)y las revisiones del usuario. . . 34

3.1. Modelo de Negocio del sistema propuesto. 39
3.2. Modelo de Objetos del Dominio 41
3.3. Modelo de Clases 43
3.4. Estructura de la tabla datos. 44
3.5. Estructura de la tabla materias. 45
3.6. Bosquejo de formulario para extraer datos. 46
3.7. Datos de la Escuela de Computación desde el 2001 al 2009 47
3.8. Modelo E/R con tablas y relaciones añadidas 48
3.9. Prototipo de gráfico lineal 52
3.10. Modelo Entidad Relación Iteración 3 54
3.11. Tabla de detalle de indicadores 55
3.12. Tabla de datos de semestre seleccionado 56
3.13. Diagramación de página principal 58
3.14. Mapa de navegacion 60
3.15. Vista principal del Foro 61
3.16. Interfaz de entrada del foro con visualización 61

3.17. Flujo de actividades en la sincronización de datos 64

A.1. Modelo E/R actualizado 82

Capítulo 1

Introducción

Históricamente, el origen de las visualizaciones de datos se remonta al siglo XVI, cuando se utilizaban técnicas e instrumentos para la creación de mapas que ayudaban a la navegación y la exploración. Mas adelante, el nacimiento del pensamiento estadístico vino acompañado de una necesidad visual: Gráficos utilizados para la ilustración de demostraciones matemáticas y funciones, diagramas diseñados para demostrar propiedades numéricas, entre otras visualizaciones que facilitaban la comunicación de la información.

Más recientemente, los avances en la computación y de las tecnologías visuales han hecho posible la creación de herramientas para la visualización de datos impensables hace medio siglo. La visualización de datos se ha aprovechado de las ventajas de las nuevas tecnologías en internet. Según la revista especializada Smashing Magazine [3], las visualizaciones de datos actuales *"plantean la belleza, elegancia y descripción en su uso"*. Desde gráficos, histogramas y tablas clásicas, hasta formas mucho más novedosas y fascinantes como mapas mentales, visualización de estadísticas de aplicaciones web y redes sociales, así como la visualización de conexiones, herramientas y servicios de sistemas informáticos son ejemplo de los avances que se han logrado al utilizar tecnologías para aplicaciones web enriquecidas [1] junto con características de interfaces de usuario basadas en interfaces dinámicas.

[1] Aplicaciones Web Enriquecidas: Son aplicaciones web que poseen características de aplicaciones de escritorio, tipicamente se basan en estandares de plugins para buscadores o estandares independientes vía maquinas virtuales (Ejemplo: Curl, Adobe Flash/Adobe Flex/AIR, Java/JavaFX)

Planteamiento del Problema

Actualmente, la Comisión Curricular de la Escuela de Computación UCV realiza mediante procesos engorrosos la creación de visualizaciones de datos. Tal como se observa en el modelo de negocios en la figura 1.1: Los datos son extraidos de la base de datos de Control de Estudios, luego son exportados a un sistema procesador de hojas de calculo donde son ordenados y seleccionados. Finalmente, utilizando las herramientas de graficación se genera la visualización que puede ser exportada para su presentación. En estos procesos, se maneja una gran cantidad de datos, lo que aumenta la complejidad y el tiempo de creación de la visualización.

Objetivo General

Desarrollar un proceso automatizado para la extracción, transformación y visualización de índices académicos que apoyen la gestión de la comisión curricular de la Escuela de Computación UCV.

Objetivos Específicos

- Diseñar e implementar la conexión a la base de datos del sistema de Control de Estudios de la Facultad de Ciencias UCV **CONEST**.
- Implementar un proceso de modelo de los datos extraídos de la base de datos.
- Proveer una interfaz para la creación de gráficos lineales y de barras mediante parámetros configurables.
- Diseñar e implementar un sistema basado en servicios web para la actualización de la base de datos utilizada en la aplicación con información proveniente del sistema **CONEST**.

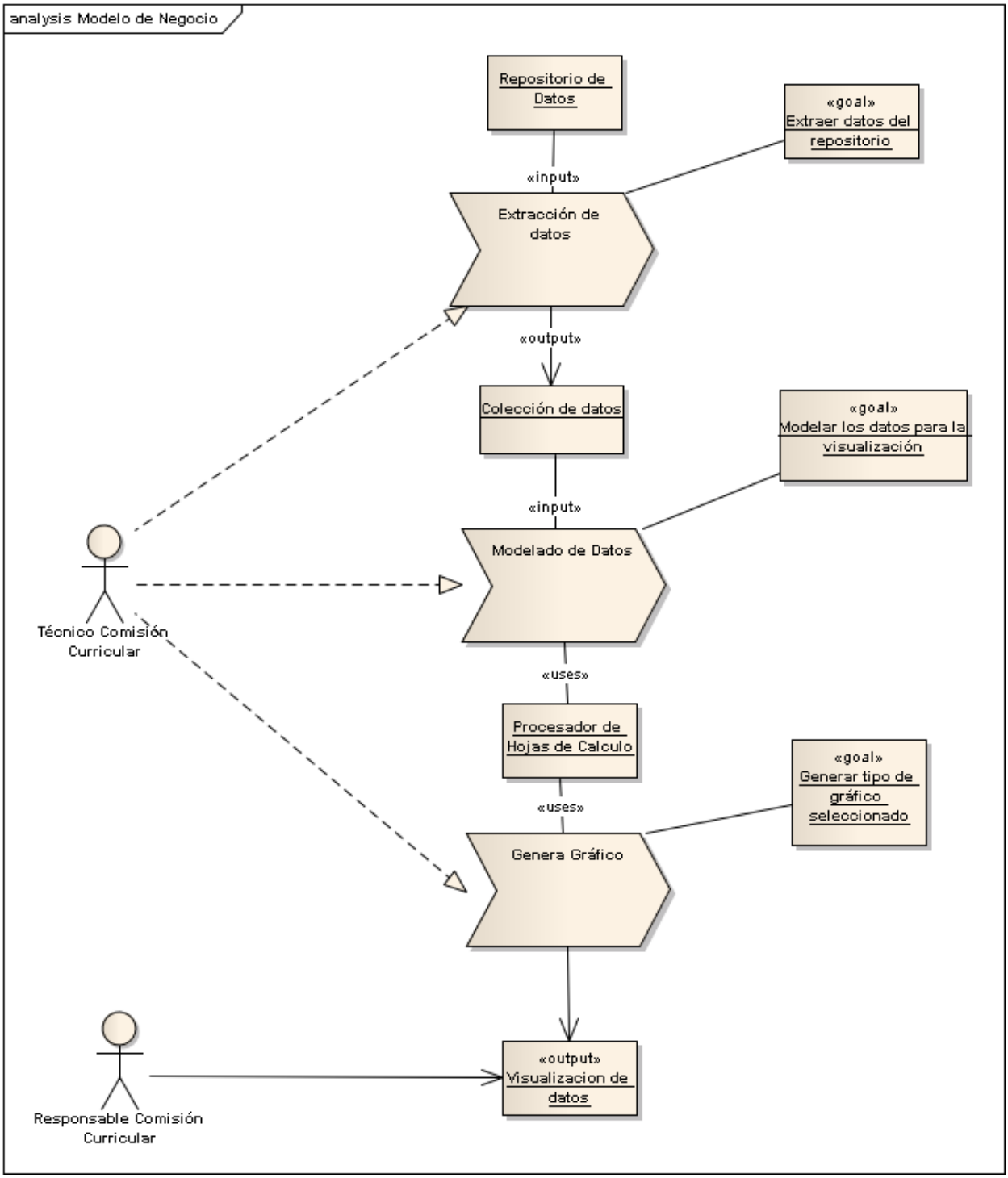


Figura 1.1: Modelo de Negocios

Justificación

La necesidad actual de las visualizaciones de datos para el apoyo en la toma de decisiones ha traído como consecuencia la creación de herramientas que facilitan el desarrollo de aplicaciones capaces de crear gráficos utilizando repositorios que manejan grandes cantidades de datos. Además, el desarrollo de estas aplicaciones requiere tomar consideraciones de diseño que permitan la integración con tecnologías implantadas, garanticen la integridad y confidencialidad de los datos (así como de los respositorios) en cualquier fase del ciclo de vida de una visualización de datos, cuidando aspectos relacionados a la interacción del usuario con la aplicación.

En el desarrollo de aplicaciones para la generación de visualizaciones de datos, se busca la reducción de costos y tiempos de entrega sin poner en riesgo la calidad del producto final. Para esto, se tienen metodologías rápidas de desarrollo que utilizan tecnologías y herramientas que promueven el reuso de componentes, creando aplicaciones de calidad con tiempos y costos reducidos.

La Comisión Curricular para disponer de los datos e índices académicos oportunamente, requiere una aplicación que automatice y simplifique dichos procesos. Esta aplicación debe comunicarse con los sistemas implantados, brindando una interfaz para la creación de visualizaciones utilizando tecnologías para el desarrollo de software y visualización web y el modelo de datos del sistema de gestión de control de estudios **CONEST**.

Alcance

El caso de estudio será la Comisión Curricular de la Escuela de Computación en la Facultad de Ciencias UCV. Las visualizaciones necesarias para el apoyo a la toma de decisiones en las reuniones de esta Comisión Curricular utilizan los datos almacenados por la División de Control de Estudios. Especificamente, se extraen la cantidad de inscritos, aprobados, retirados, repitientes y promedio de notas, así como los índices utilizados por el vicerrectorado Académico de la UCV: aprobados, aplazados, retirados y repitientes.

Además de los índices utilizados por Vicerrectorado Académico, se tienen la cantidad de inscritos, aprobados, aplazados, retirados, repitientes, equivalencias y el promedio de notas por curso.

La aplicación debe ser accedida por el grupo de usuarios seleccionado por el administrador o super usuario. En este caso el rol de administrador le pertenece al responsable de la Comisión Curricular de la Escuela de Computación UCV.

Aunque la base de datos contiene registros de los cursos dictados por las escuelas de la Facultad de Ciencias desde desde el año 1965 disponibles en la base de datos **CONEST**, por fines prácticos de la Comisión Curricular, se permite la extracción desde los datos correspondientes a los cursos dictados en el año 2001.

Estructura del Documento

Este trabajo expone el desarrollo de una aplicación escalable para la visualización de datos e índices académicos propuestos por la Comisión Curricular de la Escuela de Computación UCV. Está estructurado de la siguiente manera:

En el Capítulo 2 se recopilan los conceptos básico de herramientas de visualización de datos, su clasificación y puntos a tratar al ser diseñados. Además, se estudian las tecnologías web disponibles para el desarrollo rápido de aplicaciones, el framework para el desarrollo Web Ruby on Rails tomando en cuenta sus componentes y su implementación del patrón MVC, así como el Desarrollo Rápido de Aplicaciones RAD como método de desarrollo de software, sus fases y características.

En el capítulo 4 se define la metodología de trabajo aplicada y se describen las iteraciones realizadas en el proceso.

Capítulo 2

Marco Conceptual

2.1. Visualización de Datos

La visualización de datos es la ciencia que estudia la representación visual de los datos que han sido abstraidos de una forma esquemática como atributos o variables de unidades de información. El ser humano busca estructuras, características, patrones, tendencias, anomalías y relaciones en los datos y la visualización de datos soporta esto presentandolos en varias formas con distintas interacciones. Una visualización provee una vista cualitativa de un grupo de datos complejo, resumir datos y asistir la identificacion de regiones de interes y parámetros de interes para un analisis cuantitativo más detallado. En un sistema ideal, la visualización extiende las capacidades perceptuales del sistema visual humano[5].

La meta principal de la visualización radica en su habilidad de mostrar datos y comunicar información clara y efectivamente. Esto no significa que la visualización de datos deba estar sobrecargada para ser funcional o extremadamente sofisticada para ser atractiva. Para transmitir ideas efectivamente, la estética y la funcionalidad necesitar tomarse de la mano, brindando entendimiento entre datos dispersos y complejos y comunicando sus aspectos claves de una forma mas intuitiva. Aunque muchos diseñadores tiendan a descartar el balance entre diseño y función, creando visualizaciones muy vistosas que fallen al cumplir su propósito de transmitir información.

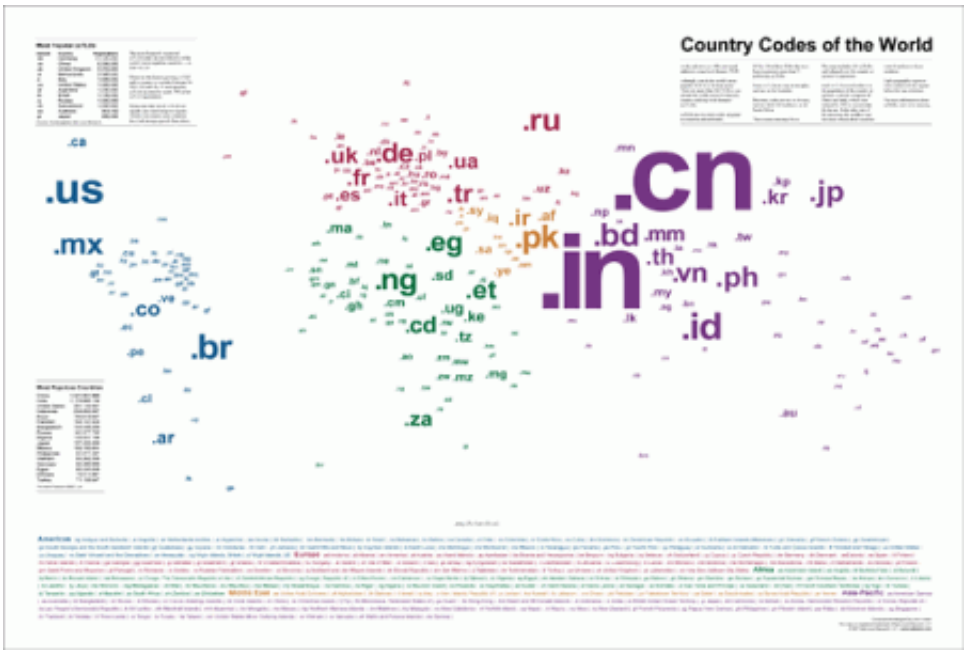


Figura 2.1: Infográfico: Códigos www por país.

Tanto el diseño impreso o web de infográficos[1], son comunmente usados para el soporte de información, fortaleciéndola y presentándola dentro de un contexto provocativo y sensible. En la figuras 2.1 y 2.2 se muestran dos ejemplos visualización de datos e infográficos[4].

Según Michael Friendly [5], los enfoques principales de la visualización de datos son: los gráficos estadísticos y la cartografía temática. Estos dos tipos de visualización de datos se enfocan en la representación cuantitativa y categórica de los datos, pero con diferentes objetivos en su representación. La visualización cartográfica, su principal preocupación es la representación restringida a un dominio espacial. Los gráficos estadísticos se aplican a cualquier dominio en el cual los metodos gráficos se emplean para el servicio del análisis estadístico.

Además, los gráficos estadísticos y cartográficos comparten objetivos comunes como la representación visual para la exploración y el descubrimiento. Esto abarca desde un simple mapeo de locaciones a distribuciones espaciales de características geofráficas (especies, mortalidad, ecosistemas) y la amplia variedad de gráficos utilizados para

[1]Representaciones visuales de información, datos o conocimiento

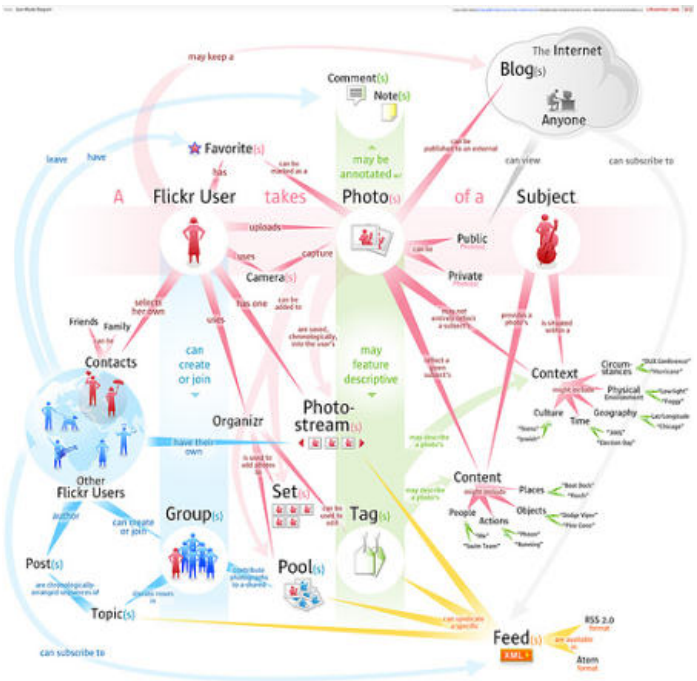


Figura 2.2: Infográfico: Modelo de Usuarios flickr.com

visualizar patrones, tendencias e indicadores.

2.1.1. Elementos de la visualización de datos

Las visualizaciones de datos se pueden caracterizar según las distintas técnicas, componentes e interacciones utilizadas en su generación. A continuación, se describen brevemente estas cualidades y se define un glosario de términos utilizados en el ámbito de la generación de visualizaciones de datos:

Técnicas de Visualización

Las técnicas de visualización pueden ser clasificadas de distintas formas. La visualización basada en tarea o en la dimensión de la presentación.

Las visualizaciones pueden ser utilizadas para explorar datos, confirmar una hipótesis o para persuadir un cliente, tal como se hace con los folletos de mercadeo. En las visualizaciones exploratorias, el usuario no necesariamente debe conocer que se espera

de la visualización. Esto crea un escenario dinámico en el cual la interacción es un factor crítico. El usuario busca identificar estructuras o tendencias e intenta llegar a alguna hipótesis. En una visualización confirmatoria el usuario tiene una hipótesis que necesita ser probada. Este escenario es bastante estable y predecible. Los parametros de sistema normalmente son predefinidos y se necesitan herramientas analíticas que permitan confirmar o refutar dicha hipótesis[2]. También se encuentran las visualizaciones de producción, donde el usuario ha validado una hipótesis y también conoce exactamente que le será presentado, por lo que enfoca y refina la visualización para optimizar la presentación (Este escenario es común en las campañas de mercadeo). Esta es la mas estable y predecible de las visualizaciones. Muchas veces, no es necesario que el autor este presente.

Las visualizaciones también pueden ser clasificadas por su tipo de dato, espacial o no espacial o por si la visualización se presenta en dos o tres dimensiones.

Componentes de la Visualización

Los datos pueden ser estáticos o dinámicos. Un sismógrafo manipula datos dinámicos sobre el tiempo. Además, una visualización puede ser estacionaria,animada o interactiva. Una resonancia magnetica, una simulación de un analisis de elementos finitos en el tiempo y la representación en tiempo real del flujo de viento sobre un automóvil son ejemplos de de visualizaciones estacionarias, animadas y dinámicas respectivamente. Así mismo, el procesamiento de datos en la visualización por parte del sistema puede ser por lotes o interactivo. Un procesamiento por lotes puede ser el análisis de un conjunto de imágenes, mientras que el análisis previo a una cirugía utiliza un procesamiento de datos interactivo.

Interacciones en la Visualización

El usuario puede interactuar con los datos de muchas formas. Esto incluye la búsqueda, la ampliación para visualizar un escenario completo, la reduccion de tamaños de dato mediante muestreos. Con una interacción dirigida en el caso de las consultas “ad-hoc” y asociativa cuando se accede a datos relacionados.

2.1.2. Terminología Básica en la Visualización

Visualización: La comunicación gráfica de información (Lo opuesto a textual o verbal).

Interacción: Un componente fundamental de la visualización que permite modificaciones específicadas por el usuario a los parametros de una visualización.

Modelo de Datos: Una representación de los datos que puede incluir la estructura, los atributos, relaciones comportamiento y semántica. Así como un repositorio de valores de datos.

Atributos Gráficos Aspectos controlables por el usuario del proceso de generación de la imagen, incluyendo la posición, el tamaño, el color, la forma, orientación, velocidad, textura y transparencia de entidades gráficas.

Mapeo Del inglés Map (Trazar un mapa o cartograma). Una asociación entre valores de datos y sus atributos y entidades gráficas y sus atributos.

Renderizar (Del inglés Render: Dibujar). Dibujar una imagen o gráfico.

Campo Una malla de puntos de datos que pueden ser uniformes o no uniformes.

Escalar Un valor de datos numérico.

Vector Una lista de valores asociados con un solo tipo de datos

Glifo Una forma o imagen generada por mapear componentes de datos en atributos gráficos (También llamado ícono).

La creación de visualizaciones involucra un proceso interactivo que parte de la extracción de los datos desde un respositorio mediante subconjuntos de datos hasta finalizar con la generación de la visualización. Este ciclo de vida se describe a continuación.

2.1.3. Ciclo de Vida de las visualizaciones de datos

Bases de Datos: Uno o mas repositorios de datos, posiblemente distribuidos, que contienen los datos primitivos y los metadatos relevantes para la tarea de exploración.

Subconjunto de bases de datos: Por medio de un mecanismo de consultas, se puede obtener un subconjunto de los datos. Estas consultas pueden basarse en rangos por atributos de un dato en particular, o por metadatos asociados al subconjunto requerido.

Mapeos de Representación Los atributos de datos requieren transformaciones para ser mapeados a atributos gráficos.

Configuración de la visualización Mapas de colores, configuración de las luces y restricciones geométricas deben ser especificadas para que la vista sea renderizada.

Vista de datos Es la visualización que se presenta al usuario.

Existen muchas decisiones críticas que deben tomarse a medida que los datos pasan de una fase a otra en el ciclo de vida, tal como seleccionar la mejor visualización para un dominio o una tarea en particular, decidir qué tipo de interacción debe proveerse y qué tanto deben integrarse las herramientas analíticas en la visualización.

Seleccionar una visualización

Existen factores importantes que deben ser considerados al seleccionar una visualización apropiada. Algunos se describen a continuación.

- La tecnología actual soporta gráficos de millones de pixeles, así como un conjunto de datos puede contener gigabytes o terabytes de información. Se debe estudiar la forma de utilizar estos recursos mas efectivamente o si es necesario recurrir a atributos como sonidos y control de tiempo para suplementar las limitaciones de la resolución de pantalla.

- La densidad o difusión de los datos, en conjunto con el mapeo de visualización seleccionado, puede requerir interpolaciones o muestreos de los datos antes de mostrarlos. Esto puede ocasionar inconvenientes en la presentación final. Por ejemplo, los datos pueden aparecer difusos y el muestreo puede terminar en pérdida de características.

- Los datos pueden presentarse en su tamaño original o reducirse via subconjuntos o proyecciones. Existen convenciones tales como la oclusión[2] a la hora de redimensionar la visualización resultante.

- La tarea que se va a realizar(Ej. Detección y medición de patrones o anomalías) y el propósito de la visualización(exploración, confirmación o presentación) puede determinar cual visualización puede ser mas efectiva en relación a otras. Una visualización exitosa es aquella que resalta la información de interés y la presenta en una resolución suficiente para cumplir su función.

Luego de seleccionar una visualización, se deben tomar en cuenta las operaciones en las que interactúa el usuario con la aplicación. Por ejemplo, el usuario debe contar con una herramienta de acercamiento para detallar áreas específicas de la visualización. Estas interacciones se conocen como interacciones para el soporte de la exploración.

Interacciones en el soporte de la exploración

Existen muchas clases de operaciones de interacción que pueden y deben ser integradas en el proceso de visualización. Estas incluyen:

Operaciones de seleccion de datos: Para obtener un conjunto de interes en una visualización en particular.

Operaciones de manipulación de datos: Para suavizar, filtrar intepolar y manipular los datos primitivos proveniente de la fuente de datos.

Operaciones de representación: Para configurar y modificar el mapeo de atributos que se estan utilizando, tales como cambiar la combinación de colores.

Operaciones de orientación/vista de imágenes: Para aumentar, reducir y modificar la ventana de visualización o viewport.

[2]Oclusión: En computación gráfica, se usa para describir la manera como un objeto puede cerrarse en relación a su vista original.

Operaciones de Interacción con la Visualización Para navegar o realizar una selección al manipular directamente los elementos que están siendo visualizados.

Una de las operaciones de interacción más frecuentes se conoce con el nombre de muestras (*probes* en inglés). Estas permitan al usuario aislar una sección de los datos que se va a visualizar y la presenta en una visualización secundaria, normalmente de dimensiones reducidas. A pesar de que crear una visualización para un conjunto de datos particulares es relativamente directo, crear una visualización efectiva que comunique la información acertadamente de una forma que ayude al usuario a realizar su tarea, puede resultar bastante una actividad bastante compleja. Es improbable que siempre exista un procedimiento a seguir para que la visualización resultante sea una visualización adecuada. Sin embargo, existen una serie de reglas de facto que se pueden tomar para evitar problemas comunes e incrementar la usabilidad de la visualización. Algunas de estas reglas pueden ser[12]:

- Incluir una leyenda y clave para los simbolos y/o colores del mapeo, así como etiquetas para los ejes. Estas son elementos esenciales para una interpretación adecuada.

- Usar mapeos intuitivos en lo posible. (Ej. Espacial a espacial, temperatura a colores). Muchas veces un mapeo intuitivo puede revelar características interesantes.

- Utilizar los colores cuidadosamente. Se debe tener en cuenta las expectativas de un contexto sensible al color, así como las limitaciones asociadas a la percepción. Proveer accesos para alternar la combinación de colores y la personalización del usuario y considerar controlar el color al mapear datos redundantes cambiando a otro atributo gráfico diferente.

- Proveer metodos sencillos para la modificación y selección de vistas, tales como acercamiento, manipulación de la panoramica y rotacion.

- Evitar imagenes recargadas. Proveer oportunidades al usuario para habilitar o deshabilitar elementos o componentes de la visualización para mejorar su vista.

- Evitar la distorsión en los datos. Las diferencias y similitudes en la representación gráfica de los datos deben ser comparables con las relaciones que tienen dichos datos.

- Escalar los datos cuidadosamente, y transmitir la escala por medio de una clave. La redimensión puede revelar elementos esenciales del conjunto de datos, pero también puede ser malinterpretado facilmente.

- Evitar las relaciones innecesarias. Por ejemplo, graficar una correlación entre manchas solares y el mercado bursátil. Se asumen que los datos mostrados en una misma visualización tiene alguna relación semántica entre sí.

- Ser conciso. Evitar el uso de artilugios (gráficos 3D para datos unidimensionales). A veces se tiende a usar todas las funcionalidades que una herramienta de visualización provee innecesariamente. La clave esta en mantener la simplicidad.

- Evitar suposiciones sobre la comprensión de la representación. Es mas confiable tener en cuenta una comprensión relativa. La habilidad del ser humano para hacer mediciones absolutas basadas en atributos gráficos es limitada [2]

- Diferenciar los datos originales de los derivados (interpolados, suavizados). Muchas veces se pueden descartar los datos derivados de aquellos que se intentan transmitir con la visualización. A pesar de que esto puede ser muy útil y efectivo, es importante informar al usuario que lo que se esta viendo ha sido modificado del original.

- No debe dejarse a un lado la estética (Las visualizaciones deben ser atractivas a la vista). Si necesidad de que se tenga conocimientos artísticos, se puede buscar el balance, la simplicidad y las combinaciones de colores agradables mientras se evita el uso de elementos llamativos, variaciones excesivas de texturas, destellos que causen distracciones, bips y cualquier otro detalle innecesario ajeno a la comunicación de información.

Es importante acotar que la implementación de estas operaciones no garantiza un entendimiento pleno del mensaje que se desea comunicar con la visualización. El análisis por parte del usuario es fundamental para cumplir este objetivo y para soportarla se han desarrollado distintas herramientas que integran el análisis con la creación de visualizaciones.

Integración del análisis con la visualización

La visualización no es un sustituto del análisis cuantitativo. En cambio, es un recurso cualitativo que soporta al enfoque analítico y ayuda a seleccionar los parámetros más apropiados para el uso de técnicas cuantitativas. Existen paquetes como Matlab y Mathematica que integran herramientas estadísticas con visualizaciones para permitir a los usuarios formular interactivamente un modelo para describir los datos que estan siendo analizados. Otros paquetes, tales como Minesite e Intelligent Miner de IBM combinan las operaciones de minería de datos tales como reglas de asociación, generadores, y clasificadores de arboles de decision con visualizaciones que pueden ser utilizadas para el aprendizaje de resultados (confirmación de hipótesis) y ajustar los parametros de las operaciones que soporta. Las herramientas analíticas también pueden ser utilizadas para guiar el proceso de visualización. Por ejemplo, XGobi utiliza algoritmos de búsqueda de proyecciones para localizar vistas en datos que potencialmente pueden contener alto contenido de información. La clave de una integración exitosa radica en un modelo de datos compartido, una navegación intuitiva y en el uso de herramientas de selección que facilitan la creación, refinamiento iterativo y la validación de hipótesis relacionadas a las estructuras contenidas en los conjuntos de datos.

En la actualidad, se tienen paquetes de aplicaciones web desarrolladas tomando en cuenta las características mencionadas anteriormente en cuanto a selección, interacción e integración con el análisis que aprovechan las ventajas que nos brinda internet. A continuación se describen estas soluciones.

2.1.4. Interfaces Web para la Visualización

Los sistemas de visualización que hemos tratado poseen interfaces propias de tipo escritorio. En la actualidad, la tendencia es de utilizar las ventajas de las tecnologías en internet y los buscadores para crear interfaces de sistemas de visualización. Esto tiene sentido, ya que al enfrentarse con datos que están distribuidos, es favorable utilizar los mecanismos de búsqueda y de recolección, además de brindar los siguientes beneficios:

- Aprovechar el acceso a internet y los repositorios de datos distribuidos en ella.
- Proveer una interfaz visual a información multimedia distribuida.
- Proveer un acceso transparente a la información en una red.
- Proveer un extenso uso de gráficos, video y sonido.
- Aprovechar el reuso de librerías portables para la generación de representaciones visuales.
- Aprovechar las tecnologías **Web 2.0** para generar interfaces usables y elegantes.

En los enfoques modernos de la visualización de datos, para transmitir un mensaje efectivamente a un usuario de nuevas tecnologías como internet, se necesita algo mas que gráficas convencionales (tablas, histogramas, gráficos de barras y de tipo torta). Para ello se utilizan técnicas permiten implementar formas creativas, profundas y fascinantes de visualizar de datos como mapas mentales, presentación de noticias, presentación de datos, vistas de páginas web, artículos y otros recursos, herramientas y servicios[2].

Por ello, las visualizaciones de datos actualmente se toman muy en cuenta en áreas relacionadas al diseño gráfico y a la representación de información, que además del análisis estadístico y cartográfico mencionados anteriormente, utilizan estas técnicas para obtener conocimiento de datos, patrones y relaciones mediante imágenes generadas por un computador. Debido a esta preocupación por la visualización de datos en ambientes web, es común encontrar herramientas e implementaciones que permiten la creación de gráficos.

Una de las soluciones utilizadas para la creación de aplicaciones web es el framework Ruby On Rails. Debido a ventajas como su escalabilidad y flexibilidad, entre otras, ha hecho que cada vez más desarrolladores lo adopten para la creación rápida de proyectos.

A continuación, se define el framework para el desarrollo de aplicaciones web Ruby on Rails, así como sus características y ventajas.

2.2. Ruby On Rails

Ruby on Rails (muchas veces llamado Rails o RoR), es un framework de aplicaciones web de código abierto escrito por David Hansson, liberado al público por primera vez en julio de 2004 que ha venido tomando auge hasta convertirse en una de las herramientas de referencia al crear aplicaciones web robustas y eficientes bajo procesos de desarrollo ágiles. Martin Fowler en su blog personal escribió sobre Rails

> En la recién formada comunidad Rails, la palabra *Empresarial* se ha convertido en una mala palabra. Para mucha gente el framework Rails, con su agresiva simplicidad es la antítesis de los sobre complejos y *empresarializados* frameworks.

El término *Framework* surge en la industria de la construcción, donde se refiere a una construcción o casa parcialmente construida. Una vez que esa construcción alcanza la etapa de framework, gran parte del trabajo está hecho, aunque luzca exactamente igual a otras construcciones que estén en esa etapa, es sólo después de que se está en esa etapa cuando los arquitectos y diseñadores comienzan a trabajar para que esa construcción sea distinta a otras. Los frameworks en aplicaciones informáticas se mantienen luego de que son instaladas y se pueden utilizar para muchas otras aplicaciones. La reusabilidad nos permite que nunca se tengan que escribir las partes en una aplicación que ya están escritas previamente como parte del framework.

David Hansson y el grupo de desarrollo del proyecto Rails han aprendido de los errores de otros frameworks de aplicaciones. En lugar de de proveer una plataforma extremadamente compleja que pueda solucionar casi cualquier problema, Rails resuelve

un problema simple extremadamente bien, y con esa solución de lado del desarrollador, es posible atacar problemas mucho más complejos de manera más fácil. Es muchos casos es más sencillo solucionar un problema complejo con Rails que entender otra plataforma para la solución.

A continuación, se citan algunas de las razones del éxito de Ruby on Rails según Sam Ruby[14]:

Convención sobre Configuración

Debido al uso de convenciones, en las aplicaciones Rails no es necesario configurar todos sus aspectos para un correcto funcionamiento. Si se pueden seguir estas convenciones, se puede lograr mucho más de lo que se puede con todos los archivos de configuraciones y código extra. Esta filosofía ha sido la base de muchos otros frameworks escritos en diferentes lenguajes como Django en Python o CakePHP en el mismo PHP, entre otros.

Uso libre de la generación de código

Rails puede escribir una gran cantidad de código por el desarrollador. El ejemplo clásico es la creación de una clase para representar una tabla existente en la base de datos en Rails, no es necesario escribir la mayoría de los métodos de esa clase sino que son generadas por el framework. Se puede además, utilizar extensiones para agregar comportamientos y funciones especiales, y si realmente se necesita, es posible agregar métodos propios. Esto hace que se escriba con Rails una fracción de código menor al que se escribe con los demás frameworks.

No te repitas (en inglés DRY: Don't Repeat Yourself)

DRY, junto a la convención sobre configuración es la característica más referida al hablar de Rails. Esta característica hace que en el framework Ruby on Rails casi nunca se tiene que repetir código, lo que lleva a escribir menos y a cometer menos confusiones a la hora de modificar un trozo de código similar a otro.

Además, Rails cuenta con características que lo hacen mucho más productivo de cara al problema de la construcción de interfaces de usuario web para el manejo de bases de datos relacionales. Entre estas características, se pueden mencionar:

Metaprogramación

La metaprogramación consiste en escribir programas que escriben o modifican otros programas o a sí mismos. Ruby es uno de los mejores lenguajes para la metaprogramación, y Rails utiliza estas técnicas para usar programas que escriban programas a diferencia de otros frameworks que brindan un pequeño incremento no generalizado en la productividad al utilizar scripts que permitan al desarrollador la personalización de código en sólo algunos puntos seleccionados.

Active Record

Basado en el patrón Active Record catalagado por Martin Fowler, Rails utiliza un framework para el Active Record, el cual guarda objetos en la base de datos. Esta versión del Active Record extrae las columnas de una tabla en la base de datos y las agrega a un objeto de dominio utilizando la metaprogramación.

***Scaffolding* o Andamiaje**

Muchas veces se utiliza código temporal en el comienzo del desarrollo para ayudar a tener una aplicación en menor tiempo y observar como los componentes trabajan entre sí. Rails genera el esqueleto o andamiaje que se va a utilizar.

Pruebas Automatizadas

Rails brinda el soporte para implementar pruebas automatizadas que pueden ser extendidas, además de proveer soporte de código Ruby que hace que los casos de prueba sean más fáciles de escribir y ejecutar. Luego de que se tienen todas las pruebas definidas, el framework puede ejecutarlas en lote.

Ambientes de desarrollo, pruebas y producción

Rails, al igual que otros frameworks nos brinda tres ambientes por defecto: Desarrollo, pruebas y producción. Cada uno es independiente del otro, y tiene un ligero cambio en el comportamiento que nos facilita el proceso de desarrollo. Por ejemplo, en el ambiente de prueba, el framework crea una base de datos de prueba para cada corrida.

Rails posee otras características como el soporte AJAX para interfaces de usuario enriquecidas, vistas parciales y helpers que ayuda en el reuso de código, facilidad en el uso de cache, frameworks de correos y de servicios web. Además, el framework implementa el patrón MVC para estructurar las aplicaciones generadas. A continuación se describe la manera en la que Rails realiza esa implementación.

2.3. Rails y el Modelo MVC

A mediados de los años 70, la estrategia MVC (Modelo - Vista - Controlador) evolucionó en la comunidad Smalltalk para reducir el acoplamiento entre la lógica de negocios y la lógica de presentación. En MVC, se centra la lógica de negocios dentro de objetos de dominio separados y se abstrae la lógica de presentación en vistas, que muestran los datos provenientes de los objetos del dominio. El controlador maneja la navegación entre vistas, procesa la entrada de datos del usuario y asegura el correcto flujo entre la vista y el modelo, y éste a su vez encapsula el almacenamiento de datos y la implementación de las reglas de negocio. Muchos desarrolladores han utilizado MVC desde ese entonces, implementando aplicaciones MVC usando frameworks escritos en diferentes lenguajes. En la figura 2.3 se observa en términos generales como está estructurada la arquitectura MVC descrita anteriormente.

Como se mencionó anteriormente, siendo Rails un framework que utiliza el modelo MVC, se deben seguir algunas reglas para estructurar una aplicación Web. Por ejemplo, al desarrollar con Rails, se crean los modelos, las vistas y controladores como partes funcionales separadas que luego se enlazan al ejecutarse el programa. Una de las ventajas que posee Rails, es que durante este proceso de unión utiliza métodos predeterminados

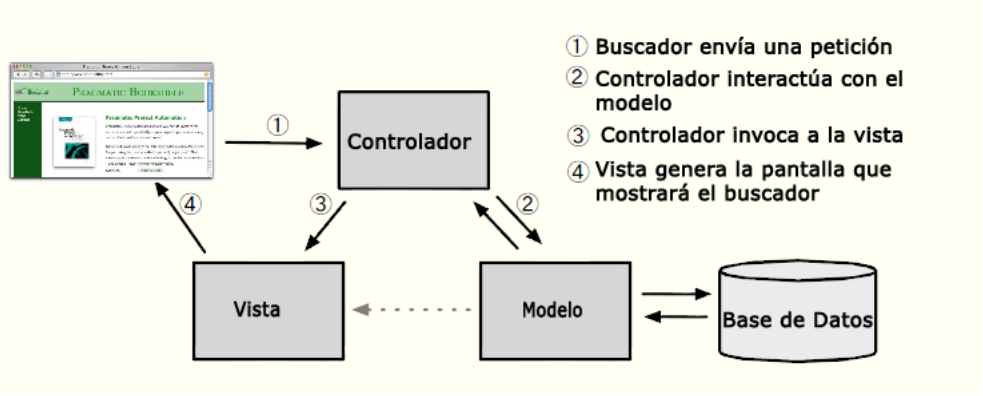


Figura 2.3: Estructura básica de la arquitectura MVC

inteligentes, por lo que no es necesario escribir código u otra configuración externa, siguiendo la filosofía de convención sobre configuración.

Tambien en Rails, las peticiones entrantes son enviadas primero al enrutador o *router*, el cual resuelve en que parte de la aplicación estas peticiones son enviadas y cómo deben ser traducidas. Luego de esto, se identifica el método en particular contenida en el controlador. Esta acción puede extraer datos que están en la petición, interactuar con el modelo e invocar a otros métodos. Eventualmente, una acción puede preparar la vista, la cual muestra visualizaciones al usuario. El modelo en una aplicación Rails (y en la arquitectura MVC) contiene toda la carga lógica de la aplicación.

Así es como Rails cumple con la arquitectura MVC, siguiendo una serie de convenciones y particionando las funcionalidades apropiadamente. Este procedimiento lo realiza el framework, ocultando los detalles y abstrayéndolos del desarrollador, lo que hace que este se enfoque en el núcleo de funcionalidades de la aplicación.

2.3.1. Active Record: El ORM de Rails.

Anteriormente al trabajar con bases de datos, el desarrollador incrustraba las consultas SQL en el código directamente mediante variables de tipo *String* o utilizando algún preprocesador que traducía estas consultas en llamadas de bajo nivel al motor de base de datos. Este enfoque todavía sigue siendo muy utilizado y muchas veces, puede con-

stituir una solución en aplicaciones pequeñas. El problema que surge cuando se mezclan la lógica de negocios y el acceso a la base de datos, es que hace a la aplicación mucho más difícil de mantener en un futuro. Por lo que se diseñó una solución a esto utilizando la orientación a objetos, encapsulando el acceso de base de datos en una capa, abstrayéndolo así del código de la aplicación y viceversa.

Bajo ese enfoque, la implementación en una aplicación puede llegar a ser más difícil de lo que parece. En la práctica, las tablas en una base de datos pueden estar interconectadas, y se quiere plasmar esto en nuestros objetos (Por ejemplo, una Orden puede tener muchos artículos asociados, por lo que en el objeto orden debería contenerse una lista de objetos artículos).

Lo anteriormente descrito, aumenta la complejidad en la aplicación y conlleva problemas de consistencia, navegación en objetos y desempeño. Por todo esto, se creó el mapeo de objetos relacionales u ORM (por sus siglas en inglés Object-Relational Mapping). Rails utiliza Active Record como ORM.

ORM mapea tablas en clases. Por ejemplo, si se tiene una tabla llamada ordenes, el programa tendrá una clase llamada Orden, las filas de la base de datos corresponden a un objeto de esa clase, con atributos para obtener o guardar las columnas de esa fila y métodos para ejecutar toda la lógica de negocios relacionada. En resumen, una capa ORM mapea tablas a clases, filas a objetos y sus columnas a atributos de esos objetos. Los métodos de esa clase se usan para realizar operaciones a nivel de tablas así como los métodos de una instancia realizan operaciones a nivel de filas individuales.

Una librería ORM típica, necesita de datos de configuración para el mapeo entre entidades de la base de datos y entidades de programa. Los desarrolladores con estas herramientas, se enfrentan a trabajar con una carga importante de archivos de configuración en formato XML. Rails mitiga este problema encapsulando cuidadosamente el estándar de un modelo ORM: Tablas mapeadas en clases, filas a objetos, y columnas a atributos de objeto. Su diferencia principal entre muchos otros ORM es la forma en la que se configura. Apoyandose en la filosofía de convención sobre configuración y comenzando con valores por defecto, Active Record minimiza la cantidad de configuraciones que los desarrolladores realizan.

Active Record es la base del modelo MVC en Rails, por lo que se ha dedicado una gran relevancia en su estudio tanto en la documentación como bibliografías dedicadas. Junto a él, se tienen las clases *Action Controller* y *Action View* que definen la base para el manejo de la capa controlador y vista respectívamente

2.3.2. La Vista y el Controlador.

Anteriormente, se describió a fondo el funcionamiento de una aplicación con una arquitectura MVC, y se pudo observar que la capa de Vista y el Controlador están íntimamente relacionados. El controlador provee datos a la vista y recibe eventos desde las páginas generadas por ésta. El componente de Rails encargado de la capa Vista se conoce como *Action View*, su funcionamiento se estudiará en detalle a continuación.

La Vista

La vista es la responsable de la presentación de la respuesta que la aplicación enviará al usuario, esta puede ser una página,un archivo XML o un correo electrónico, etc. Específicamente en Rails, la vista genera contenido por medio de tres plantillas. El esquema de plantillas más utilizado es el ERb(Embedded Ruby o Ruby Embebido, similar a esquemas utilizados bajo PHP o JSP), que consiste en código Ruby incrustrado en un documento de vista utilizando una herramienta con el mismo nombre. Este enfoque resulta ser flexible, aunque en muchas discusiones se ha visto como una violación a la arquitectura MVC, ya que se puede correr el riesgo de agregar lógica a la vista que debería estar en el controlador o en el modelo. Sin embargo, desde el MVC original, la vista puede contener código activo y depende del programador mantener la separación de funciones tal como lo describe la arquitectura.

Por último, la clase encargada de la capa controlador se conoce como *ActionController*

El Controlador

El Controlador es el centro lógico de una aplicación MVC. Desde esta capa se coordina interaccion entre el usuario, las vistas y el modelo. Rails maneja la mayor parte de

esta interacción, permitiendo que el código escrito se enfoque en las funcionalidades a nivel de aplicación, haciendo las aplicaciones mas fáciles de desarrollar.

Además, el controlador brinda los siguientes servicios:

- Es el responsable del enrutamiento de peticiones externas a acciones internas.
- Maneja el uso del cache, que mejora el desempeño en las aplicaciones.
- Maneja los módulos *helpers*, extendiendo las capacidades de las plantillas de la vista sin sobrecargarlas de código.
- Maneja el uso de sesiones, dando la sensación de una interacción continua a las aplicaciones.

Actualmente, surge en las aplicaciones web la necesidad del uso de recursos brindados por otras aplicaciones. Estos recursos se conocen como servicios web y Rails implementa el soporte a éstos tal como se describe a continuación.

2.4. Servicios Web en Rails

Un Servicio Web o Web Service, es un sistema identificado por un URI (Uniform Resource Identifier o Identificador Uniforme de Recurso), cuyas interfaces pública y relaciones son definidas y descritas utilizando XML. Su definición puede ser descubierta por otros sistemas de software que interactúan con el servicio web según su definición, usando mensajes basados en XML transportados mediante protocolos de Internet[1] En terminos generales, un web service se divide en dos categorías: Cliente y Servidor. Ambos se comunican a través de tres protocolos principales: Representational State Transfer (REST), Simple Object Access Protocol (SOAP) y Extensible Markup Language - Remote Procedure Call (XML-RPC). Es común encontrar web services que permitan conectarse a través de más de un protocolo. Por ejemplo, muchas aplicaciones que utilizan el Action-WebService de Rails ofrecen acceso vía SOAP y XML-RPC, esto permite que los clientes puedan implementar la arquitectura que más convenga según la aplicación. Así mismo, un servicio web no depende del lenguaje en el que se implementa. Un WebService en

un servidor implementado en Rails puede ser utilizado por clientes con Rails además de otros desarrollados los lenguajes de programación y scripting PHP, Java, ASP, entre otros.

Cuando se habla de un servicio web de tipo cliente o servidor, se debe tener en cuenta que la mayoría de los clientes del servicio web no son programas de escritorio o de ejecución por lotes. De hecho, la mayoría se utilizan en servidores que recolectan datos de otro servicio web y lo empaquetan para el mismo propósito. A diferencia de otros cuyo propósito es generar una aplicación web, que a su vez puede servir como otro servicio web.

Las implementaciones de Rails con versiones anteriores a la 2.0 definen el uso de servicios web mediante la clase Action Web Service, la que se define a continuación.

2.4.1. Action Web Service

En Rails, el soporte para la creación de Web Services SOAP o XML-RPC lo provee el módulo ActionWebService (AWS). Este convierte las peticiones entrantes de invocaciones de métodos en llamadas a métodos en nuestros web service y se encarga de enviar las respuestas. Lo que permite enfocar el trabajo en escribir los métodos específicos en la aplicación para servir a esas peticiones.

AWS no implementa toda las especificación de SOAP y WSDL (Web Service Definition Language) propuesta por W3C, así como tampoco provee todas las características de XML-RPC. En lugar de eso, se enfoca en la funcionalidad que razonablemente se espera en el uso con regularidad de un web service. Brindando características como[14]:

- Tipos de estructuras anidados de manera arbitraria.
- Uso de variables tipificadas de tipo arreglo.
- Envío de información y trazas al registrarse una excepción.

Action Web Service permite algunas libertades en el formato de llamadas remotas entrantes, y a su vez tiene ciertas restricciones al momento de emitir una respuesta, forzando una asignación correcta de los tipos de datos en los valores de entrada y salida.

Además, mediante del uso de Action Web Service, se puede:

- Utilizar funcionalidades de aplicaciones web como blogs, redes sociales mediante interfaces de programación (API) en una aplicación Rails. Por ejemplo, las redes sociales Facebook, Blogger y Twitter son aplicaciones web que brindan soporte a un API mediante servicios web.

- Implementar una interfaz propia y permitir a desarrolladores generar clases partiendo del WSDL generado por el Action Web Service.

- Brindar soporte a los protocolos de transporte SOAP y XML-RPC utilizando el mismo código.

Al crear un Servicio Web en una aplicación Rails utilizando Action Web Service, lo primero que se debe hacer es decidir la funcionalidad que se desea brindar a los clientes remotos, así como la cantidad de información que se les envía. En un sistema ideal, sería suficiente con solo escribir una clase implementando la funcionalidad y hacerla disponible para su invocación, pero esto causa problemas de interoperabilidad al comunicarse con lenguajes que no posean el mismo dinamismo de Ruby. Lo que nos puede llevar a un fallo en el momento en el que un cliente remoto recibe una respuesta inválida.

Debido a esto, AWS realiza forzado de asignación de tipos. Si un parámetro entrante o un valor de retorno no es del tipo correcto, Action Web Service trata de convertirlo. Esto abstrae al cliente de este problema y evita que se tenga que manipular la aplicación para asignar tipos correctos a los valores de entrada cada vez que se reciban valores erróneos, como por ejemplo recibir una variable de tipo cadena en vez de un valor entero esperado. Además, debido a que Ruby no puede usar definiciones de métodos para determinar los parámetros de entrada y los valores de retorno de una función, es necesario crear una clase de definición API que no contiene codigo de implementación y no pueden ser instanciadas, estas solo describen la API.

Como se describió anteriormente, los clientes de un servicio web envían peticiones a una dirección URL, que luego son procesadas por AWS, mapeándolas a los métodos que implementan el servicio. Esto se conoce como despacho o *dispatch*. El despacho por

defecto se conoce como despacho directo, el cual no requiere configuración adicional para realizarse y se relaciona la definición directamente con el controlador. Es decir, las implementaciones de los métodos definidos en la API son colocados en el controlador como metodos públicos.

Además del despacho directo, se tienen el despacho por capas y el despacho delegado. El despacho por capas permite relacionar multiples definiciones API asociados a un URL con un sólo controlador. Por ultimo, el despacho delegado es similar al despacho por capas, sólo que para cada definición API se tiene un URL único.

Para evitar duplicar el mismo código en múltiples métodos, AWS permite realizar la intercepción de invocaciones, que permite registrar llamadas antes y/o después de cada petición al web service. La intercepción en AWS trabaja de manera similar a los filtros implementados en el Action Pack, pero incluye información adicional acerca de la petición web service, como los son el nombre del método y sus parámetros decodificados.

Las versiones de Rails posteriores a la 2.0 no incluyen el paquete ActionWebService, por lo que sólo se implementa el protocolo REST. Si se desea el soporte para SOAP o XML-RPC, se puede obtener descargando el gem.

Al comenzar un proyecto de desarrollo de software, se debe tomar en cuenta una metodología de trabajo que define los pasos para cumplir dicho desarrollo. En la actualidad, al abordar un proyecto de este tipo se busca la reducción de costos y tiempos de entrega sin poner en riesgo la calidad del software. Por ello, el uso de procesos pesados con una gran carga de artefactos y entregables se ha reducido, y a su vez, han surgido distintas metodologías rápidas que responden a estas necesidades.

El Desarrollo Rápido de Aplicaciones, es una de las primeras metodologías documentadas basadas en la construcción de prototipos y el uso de utilidades que aumenten la velocidad de desarrollo de software. A continuación, se describe el Desarrollo Rápido de Aplicaciones.

2.5. Desarrollo Rápido de Aplicaciones

La metodología RAD ó Rapid Application Development, surgió como resultado de enfoques anteriores basados en prototipaje rapido y fue formalizada en 1991 por James Martin en su libro *Rapid Application Development* [10]. En este libro, Martin escribe:

> Rapid Application Development (RAD) es un ciclo de desarrollo diseñado para dar resultados de desarrollo y alta calidad mucho mas rapido que los obtenidos con los otros metodos tradicionales. Esta diseñado para tomar mayor ventaja de los poderosos software de desarrollo que han emergido recientemente.

RAD tiene como objetivos clave la creación de sistemas funcionales de alta calidad,el cumplimiento de fechas de entrega cortos entre 60 y 90 días, así como la reducción de los costos utilizando grupos de desarrollo reducidos. A estos objetivos se les suma la necesidad comercial de entregar aplicaciones de negocio en una menor escala de tiempo e inversión.

Tal como se mencionó anteriormente, RAD es el resultado de metodologías basadas en prototipos. Específicamente, es la evolución de una metodología formulada por Boehm y Gilb llamada RIPP: Rapid Iterative Production Prototyping (Producción de Prototipaje

Rápido e Iterativo) y de comprimir metodologías convencionales, transformandolas en procesos iterativos como se puede observar en la figura2.4. El enfoque incluye el desarrollo y refinamiento de modelos de datos, modelos de proceso y prototipaje en paralelo utilizando un proceso iterativo. Los requerimientos de usuario son refinados, una solución es diseñada, prototipada y revisada. Luego, se provee una entrada de usuario y el proceso comienza de nuevo.

Figura 2.4: Enfoque del modelo RAD

Los desafíos que enfrentan las organizaciones de desarrollo de software pueden ser resumidos como la siguiente frase: “más, mejor y mas rápido”. El Desarrollo RAD aborda estos puntos brindando recursos para desarrollar sistemas mas rapido, mientras se reducen los costos y se incrementa la calidad. La metodología RAD consiste fundamentalmente en:

- Combinar las mejores técnicas disponibles y especificar la secuencia de tareas que hacen a esas técnicas más efectivas.
- Usar prototipos evolutivos que gradualmente son transformados en el producto final.
- Realizar talleres para recopilar requerimientos y revisar diseños.

- Seleccionar un conjunto de herramientas CASE[3] para el soporte en el modelado, prototipaje y la reusabilidad de código.
- Implementar desarrollo por cajas de tiempo que permite al equipo construir el núcleo del sistema más rápidamente e implementar refinamientos en entregas posteriores.
- Proveer directrices para el éxito y describir dificultades a evitar.

Al involucrar activamente al usuario en el ciclo de vida RAD asegura que los requerimientos de negocio y las expectativas del usuario sean claramente comprendidas. RAD aprovecha las ventajas de poderosas herramientas para el desarrollo de aplicaciones para crear aplicaciones de alta calidad mucho mas rapidamente. El prototipaje se usa para visualizar y realizar cambios en el sistema mientras que esta siendo construido, permitiendo a las aplicaciones evolucionar iterativamente. Las técnicas RAD tienen mucho éxito al afrontar requerimientos de negocio inestables, o bien cuando se desarrollan sistemas no tradicionales.

El ciclo de vida RAD consta de las fases descritas a continuación.

2.5.1. Fases de RAD

El modelo RAD puede verse como una adaptación del modelo lineal secuencial con la construcción basada en componentes, si los requerimientos y el enfoque del proyecto están bien definidos, el proceso RAD permite a un equipo de desarrollo crear un sistema funcional en períodos cortos de tiempo normalmente entre 60 y 90 días[13].

La estructura del ciclo de vida en el desarrollo RAD esta diseñada para asegurar que los desarrolladores construyan sistemas que el usuario realmente necesita. Este desarrollo, consta de las siguientes cuatro fases, donde se incluyen todas las actividades y tareas requeridas para enfocar y definir requerimientos de negocio y así diseñar, desarrollar e implementar un sistema que satisfaga esos requerimientos.

[3]Herramientas CASE (Acrónimo de Computer Aided Software Engineering). Se refiere a un conjunto de herramientas para el apoyo de las actividades en la ingeniería del software.

Fase de Planificación de Requerimientos

Llamada también Fase de Definición de Conceptos, define las funciones de negocios, el dominio en el que se encuentran determina el enfoque del sistema y los datos que soportará.

Fase de Diseño de usuario

También conocido como fase de diseño funcional, en esta fase se realizan talleres para modelar los datos y procesos del sistema y se construye un prototipo con los componentes críticos del sistema.

Fase de Construcción

Se conoce también como Fase de desarrollo. En esta fase, se completa la construcción del sistema físico de la aplicación, se desarrollan ayudas de usuario, así como planes de trabajo para la implementación.

Fase de Implementación

También conocido como fase de despliegue. En esta fase se incluyen las pruebas finales del usuario y el entrenamiento. También se realiza la conversión de datos y la implementación del sistema de aplicación. En el ciclo de vida de los proyectos RAD, se deben tomar en cuenta sus características como el uso de prototipaje, los talleres de desarrollo de aplicación, entre otras que son descritas con más detalle a continuación.

2.5.2. Características del modelo RAD

En distintos enfoques RAD discutidos en la literatura, resaltan las siguientes características:

Desarrollo Conjunto de Aplicaciones

RAD se implementa en grupos de desarrollo pequeños de hasta 8 personas. Estos grupos se componen de desarrolladores y usuarios con potestad de tomar decisiones en el transcurso del ciclo de desarrollo. Esto conlleva que todos los miembros del grupo deben poseer ciertas habilidades sociales y de negociación, así como usuarios con conocimiento detallado del dominio de la aplicación y desarroladores con habilidades en el uso de herramientas avanzadas. Por eso, actividades de construcción del equipo o *Team-Building*, tales como almuerzos grupales, son vistas como una parte importante de un proyecto RAD[8]. Muchos enfoques evidencian el uso de talleres de Desarrollo Conjunto de Aplicación o Joint Application Development (JAD) en varios puntos del proceso de desarrollo, particularmente en la fase de levantamiento de requerimientos. En estos talleres, los usuarios clave, el cliente, los desarrolladores y un encargado de la transcripción de los hechos producen un enfoque del sistema y requerimientos de negocio bajo la tutela de un facilitador. Los equipos de desarrollo esperan obtener requerimientos altamente documentados en un período de tres a cinco días. Estos requerimientos pueden especificar una serie de entregables con fechas de culminación fijados dentro de un período de tiempo. Mas adelante. Además, los talleres pueden ser establecidos durante el ciclo de vida del proyecto para desarrollar conjuntamente los entregables.

Rapidez de Desarrollo

Típicamente, los proyectos bajo RAD son a pequeña escala y de corta duración. De hecho, se considera una duración de dos a seis meses como la duración normal de un proyecto. La razón fundamental de que cualquier proyecto tarde mas de seis meses en completarse es comunmente que existan muchas negociaciones implicadas en el proceso. En definitiva, es recomendable no dedicar más de seis años hombre en el desarrollo de cualquier proyecto RAD. Por ello, existen proyectos en los que se pide un período de culminación menor al tiempo esperado.

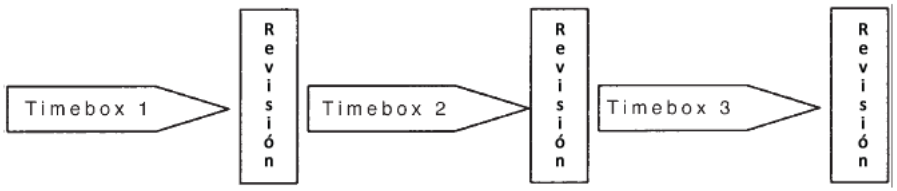


Figura 2.5: Relación entre las cajas de tiempo(*timeboxes*)y las revisiones del usuario.

Cajas de tiempo

El control en proyectos RAD involucra enfocar el proyecto a prioritizar el desarrollo y definir las fechas tope o cajas de tiempo (*timebox*). Si los proyectos comienzan a descuidarse,los esfuerzos se enfocan en reducir los requerimientos para ajustarse al *timebox*, no en alargar la fecha tope. En la figura 2.5 se muestra la relación entre el uso de cajas de tiempo y las revisiones del producto por parte del equipo de usuarios.

Prototipaje Incremental

En RAD, los términos de prototipaje incremental y entregables con fechas fijadas son ampliamente discutidos. El prototipaje es escencialmente el proceso de construir un sistema de una manera interactiva. Los desarrolladores, despues de investigaciones iniciales, construyen un modelo funcional que luego es mostrado a un grupo representativo de usuarios. Junto con los usuarios, los desarrolladores discuten sobre este prototipo, tratando sus correcciones e incrementos. Este ciclo de inspección-discusión-corrección usualmente se repite al menos tres veces en un proyecto RAD, hasta que el usuario está satisfecho con el sistema [7]. En RAD, el prototipaje puede ser utilizado en cualquier fase del ciclo de desarrollo: Elicitación de requerimientos, diseño de la aplicaciónn, construcción de la aplicación, pruebas y entrega. Además, se conocen dos tipos de incrementos por prototipaje: El prototipaje horizontal, que muestra incrementos de la aplicación en interacción o presentación y el prototipaje vertical, que a su vez, refleja incrementos funcionales [13].

Herramientas para el Desarrollo Rápido

Es común encontrar en los enfoques modernos de RAD, el uso de herramientas que soportan el desarrollo. Esto normalmente implica combinaciones de lenguajes de programación última generación, herramientas para la construcción de interfaces gráficas, sistemas manejadores de bases de datos y herramientas CASE para el apoyo en la ingeniería de software. Al usar estas herramientas, las correcciones discutidas en las reuniones usuario-desarrollador pueden ser realizadas inmediatamente. Kerr y Hunter[7], describen un proyecto que utiliza la metodología RAD de Martin en el desarrollo de de un sistema financiero para un banco en los Estados Unidos. En esta descripción se observa un gran uso de las herramientas CASE, así como la mitigación de riesgos y problemas, tales como el *developer-burnout*[4].

[4]Developer Burnout: Estado de depresión que sufren los desarrolladores de un proyecto causado por situaciones de estrés.

Capítulo 3

Marco Aplicativo

3.1. Aplicación del método de desarrollo RAD

Para comenzar con el desarrollo de un prototipo para la visualización de datos de la Comisión Curricular de la Escuela de Computación, se procede a definir las fases del modelo RAD aplicadas a este escenario particular.

3.1.1. Fases a implementar

Planificación de Requerimientos

En esta fase se definen los requerimientos de la iteración basados en las revisiones del usuario. Estos requerimientos pueden soportarse con la creación de un glosario de términos, modelado de negocios, de la misma forma como se describe en fases implementadas en metodologías ágiles como el Proceso Unificado Ágil o AUP [9].

Diseño Funcional

En esta fase se parte del modelo de objetos de dominio para definir las entidades conceptuales. Utilizando la notación UML, el Modelo de Dominio es un conjunto de diagramas de clases en las que no se definen operaciones. Es importante destacar que los objetos que se toman en las primeras revisiones son abstracciones tomadas del negocio

y no de la implementación de la aplicación [9]. A medida que el desarrollo avanza, se definen los datos y procesos que se construyen en la siguiente fase.

Generación de Aplicación

La construcción de la aplicación se realiza mediante framework Ruby on Rails brindando una arquitectura MVC integrada al Sistema Manejador de Bases de Datos libre MYSQL. También se utilizará el framework Javascript JQuery para el desarrollo de las vistas junto a la librería gráfica **JQPlot**.

Implementación y pruebas

Para la etapa de implementación y pruebas, se estarán probando los componentes mediante pruebas unitarias y funcionales.

3.1.2. Herramientas a utilizar

Se tienen como herramientas CASE, el sistema propietario Enterprise Architect para la modelación y UML junto con el sistema de uso libre MySQL Workbench para el manejo y modelado de la base de datos.

3.1.3. Revisiones con el usuario

Las mesas de revisón se harán con una frecuencia de 7 a 15 días, según lo acordado con el grupo de usuarios. En estas reuniones se obtiene la retroalimentación por medio de prototipos funcionales.

3.2. Iteracíón 0

En esta iteración se estudia el negocio de la aplicación. Se discuten el funcionamiento sistema utilizado actualmente para la generación de visualizaciones y los conceptos de datos e índices de rendimiento académico.

3.2.1. Requerimientos

La aplicación brinda una interfaz para la extracción de datos tomando en cuenta parámetros de entrada. Éstos pueden ser cronológicos o por organización curricular de la escuela de Computación. Los parámetros cronológicos definen el año y el período lectivo del curso en específico. Los parámetros por organización currícular, varían desde mostrar los datos por licenciatura, hasta filtrar por materia, pasando por los distintos componentes, menciones y semestres asignados en el pensum registrado en el modelo de datos **CONEST**.

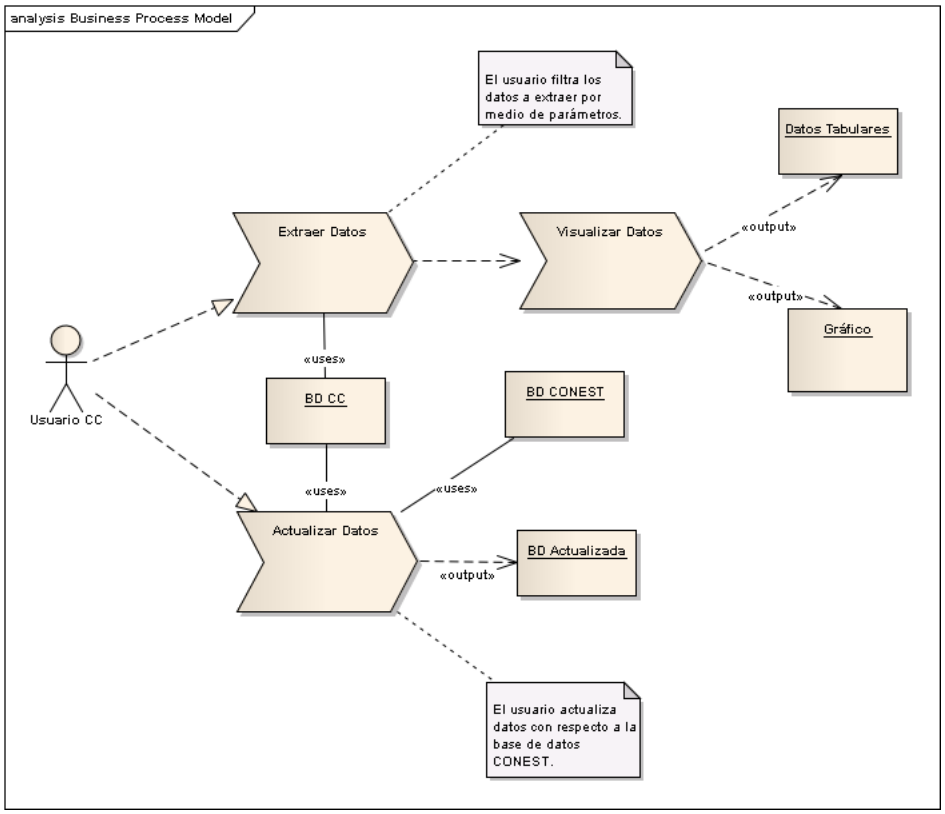


Figura 3.1: Modelo de Negocio del sistema propuesto.

3.2.2. Diseño

El Modelo de Objetos del Dominio (Fig. 3.2) corresponde a la organización de la Escuela de Computación, la cual posee un pensum de estudio actual en el que las materias

son clasificadas por componentes: básico, instrumental, profesional, complementario y práctica profesional. Ordenados dentro de los 10 semestres de carrera. El pensum actual es utilizado en la escuela desde el 2004, siendo éste un refinamiento del pensum 2000 que contiene modificaciones notables con respecto a su predecesor, utilizado desde el año 1985. Los cursos son las materias dictadas en un período académico, y se dividen en secciones. La cantidad de secciones depende del número de inscritos y y de profesores disponible en ese período lectivo.

Los estudiantes pertenecientes a una Escuela, pueden inscribirse en los cursos que estén disponible sí y solo sí hayan aprobado las materias que tenga como prelaciones. Cada estudiante mantiene un historial académico que registra las calificaciones obtenidas en los cursos inscritos.

La Comisión Curricular de la Escuela de Computación tiene como función asesorar al Consejo de Escuela en la elaboración de estrategias y políticas curriculares basados en estudios cualitativos y cuantitativos previamente establecidos. El estudio cuantitativo del rendimiento académico puede hacerse mediante el número de inscritos, aprobados, aplazados, retirados, repitientes y el promedio de notas de los estudiantes de la escuela.

Se desean incluir los índices de aprobados, aplazados, retirados, repitientes, eficiencia y promedio, aprobados por el Vicerrectorado Académico de la Universidad Central de Venezuela.

3.3. Iteración 1

En esta iteración se realiza la extracción de índices y datos requeridos en la aplicación.

3.3.1. Requerimientos

Los requerimientos funcionales de la iteración son:

- Obtener la cantidad de inscritos, aprobados, aplazados, retirados, repitientes y promedio por licenciatura, período lectivo y materia.

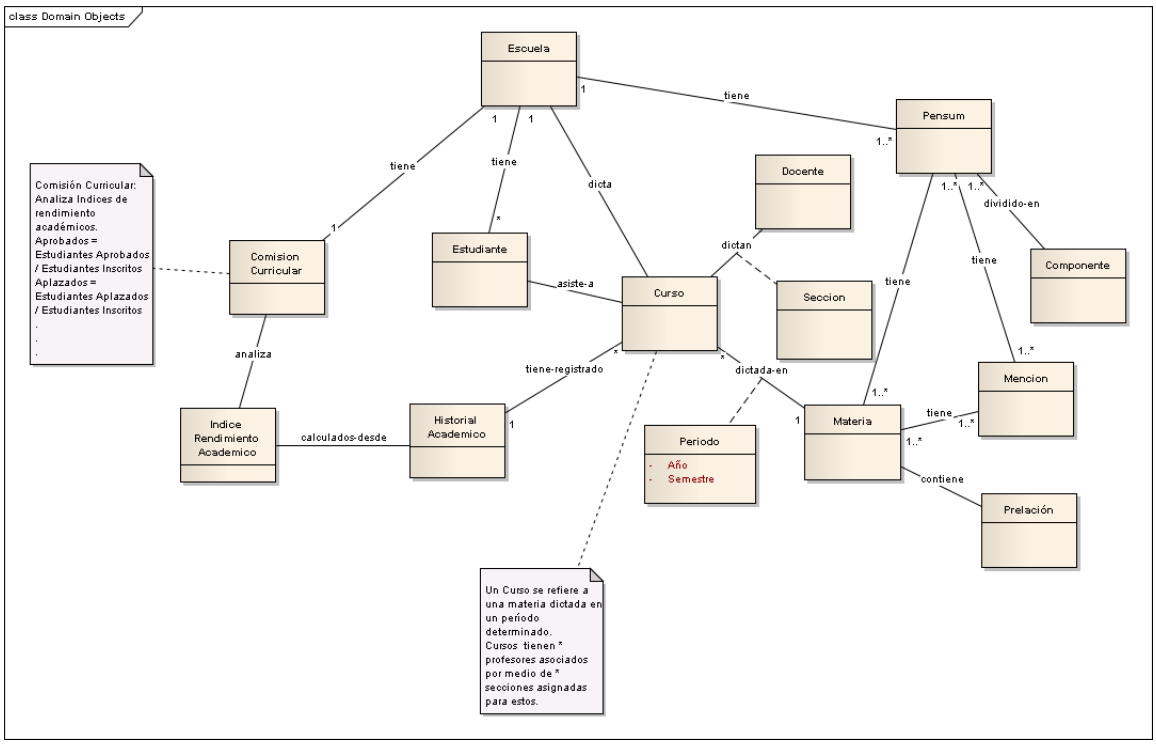


Figura 3.2: Modelo de Objetos del Dominio

- Calcular índices de aprobados, aplazados, retirados, repitientes, promedio y eficiencia por licenciatura, período lectivo y materia.

Se realizó el siguiente glosario de términos:

Período lectivo. Período comprendido por 16 semanas de actividades académicas.

Datos gruesos. Término relacionado a los datos extraídos de un repositorio y no han sido sujetos a algún cambio o cualquier otra manipulación.

Cohorte. Conjunto de estudiantes agrupados por período lectivo de ingreso.

Promoción. Conjunto de estudiantes agrupados por período lectivo de graduación.

Promedio de notas. Valor equivalente a la suma de todas las calificaciones de un estudiante divididas entre el total de materias calificadas.

Índice de Aprobados. Proporción de alumnos que aprueban una asignatura, curso o

semestre con relación al número de inscripciones definitivas realizadas en el período lectivo

Índice de aplazados. Proporción de alumnos que no aprueban una asignatura, curso o semestre, habiendo asistido como mínimo al 75 % de las actividades programadas, respecto al número de inscripciones definitivas realizadas en el período lectivo.

Índice de eficiencia académica. . Relación que se establece entre la sumatoria de los créditos aprobados por todos los estudiantes de un período y la sumatoria de créditos dictados por la institución en el mismo período lectivo.

Luego de definir los términos básicos, comienza la fase de diseño del proceso para la extracción de datos y cálculo de índices académicos.

3.3.2. Diseño

El Modelo de Clases resultado del refinamiento del modelo conceptual de datos se muestra en la figura 3.3.

El usuario utiliza la aplicación para la extracción de datos. Esta se realiza seleccionando el rango de años a mostrar (los datos requeridos se pueden elegir desde el año 2001), datos e índices a mostrar. Los datos pueden mostrarse por período de tiempo, mostrando el acumulado por Escuela, componente en pensum, por un grupo de materias o una materia en particular. Cuando se elige el nivel de vista por grupo de materias, el conjunto seleccionado puede ser por semestre asignado en el pensum, mención o por componente.

La generación de gráficos se ejecuta luego de que se tienen los parámetros definidos y dibuja una tabla con los datos resultados. Los datos pueden ser exportados como hoja de cálculo si el usuario lo requiere.

Los datos almacenados en la base de datos se actualizan mediante un procedimiento automático ejecutado cada final de semestre. El usuario tiene la posibilidad de chequear la integridad de los datos con respecto a los registros que se encuentran en la base de datos del sistema de control de estudios Conest.

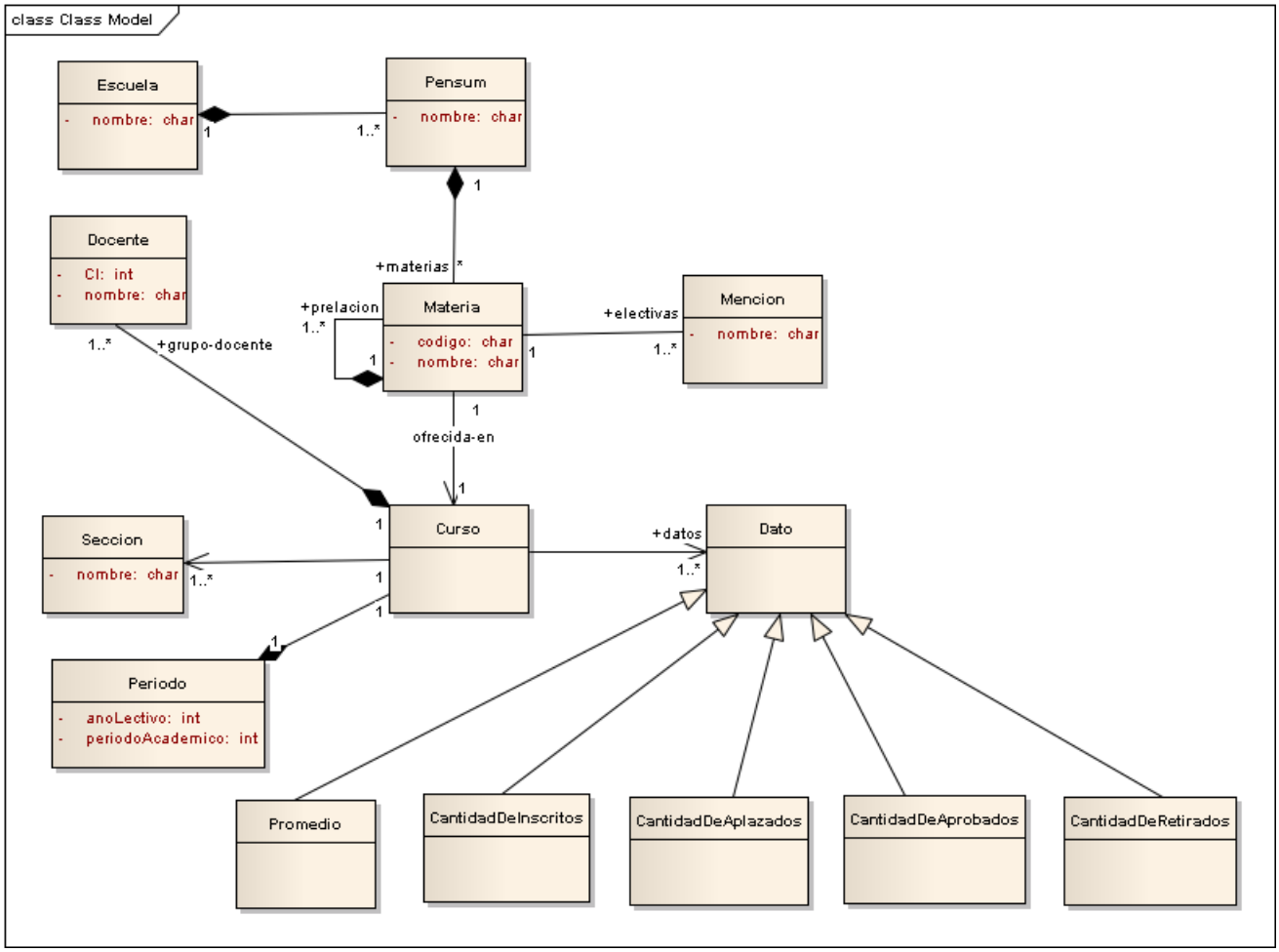


Figura 3.3: Modelo de Clases

En la fase de Generación de la aplicación se procede a crear un formulario simple para la selección de los datos que serán mostrados en formato tabular.

3.3.3. Generación de la aplicación.

Para cumplir con los requerimientos de la iteración, se crea la primera tabla en la base de datos de aplicación, que tendrá la estructura mostrada en la figura 3.4. En esta tabla se tendrán almacenados la cantidad de inscritos, aprobados, aplazados, retirados, eficiencia, repitientes y promedio de notas en cada curso registrado por el sistema **CONEST**. Ya que se tiene la aplicación conectada a la base de datos, se necesita un mecanismo de extracción de los datos requeridos. Para esta iteración, no es necesaria la interacción en

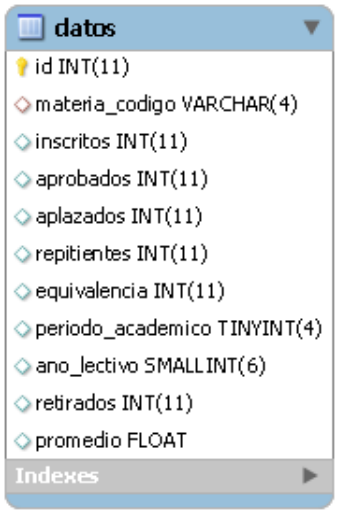


Figura 3.4: Estructura de la tabla datos.

este proceso por parte del usuario, por lo que se puede utilizar el procedimiento descrito en el apéndice A.1 en la base de datos **CONEST**:

Para mantener la integridad en la tabla `datos`, el procedimiento se invoca manualmente por la cónsola MySQL con los parámetros 1965 y 2009. Por lo que, en principio, se tiene una tabla con datos académicos que van desde el año 1965 hasta el año 2009.

El procedimiento `CC\_extraer\_datos()` inserta directamente a la tabla `datos`. Posteriormente, éste podrá reutilizarse para la implementación de la conexión con la aplicación CONEST.

Tal como aparece en la estructura de la tabla *datos*(Fig. 3.4), cada dato está relacionado a una materia por medio del campo *materia_codigo*. En la aplicación debe crearse la tabla `materias`. Luego de creada, los valores son importados de la base de datos **CONEST** según la estructura definida en la figura 3.5

El cálculo de los Índices de Rendimiento Académico se realiza mediante la aplicación a partir de la cantidad de inscritos, aprobados, aplazados, retirados y repitientes extraídos de la base de datos **CONEST** y almacenados en la base de datos del sistema. Aprovechando la estructura que brinda Rails, los datos pueden calcularse desde el Modelo `Curso`. Para ello, se modifica su contenido con el siguiente código:

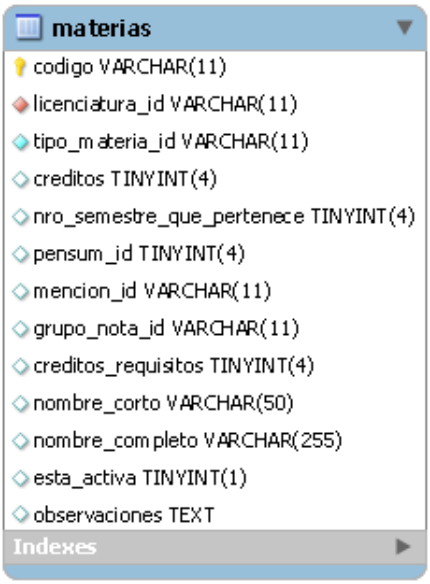


Figura 3.5: Estructura de la tabla materias.

```
class Curso < ActiveRecord::Base
def indice_eficiencia
(self.materia.creditos * self.aprobados).to_f /
(self.materia.creditos * (self.inscritos-self.equivalencia))
unless (self.materia.creditos * (self.inscritos-self.equivalencia)) == 0
end
def indice_aprobados
(self.aprobados).to_f /
(self.inscritos-self.equivalencia) unless (self.inscritos-self.equivalencia) == 0
end
def indice_aplazados
(self.aplazados).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
def indice_retirados
```

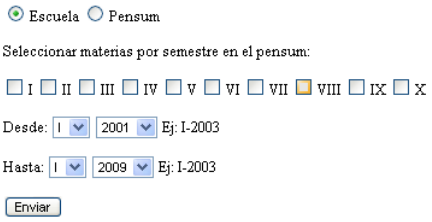


Figura 3.6: Bosquejo de formulario para extraer datos.

```
(self.retirados).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
def indice_repitientes
(self.repitientes).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
end
```

Para tener una vista de los datos, se crea el controlador `datos`, que en esta iteración mostrará un formulario para la selección de los parámetros y una vista en la que se muestra la tabla con datos extraídos.

Se crea un formulario simple para seleccionar los datos(Fig. 3.6) y una tabla que contiene los datos extraidos (Fig. 3.7).

3.4. Iteración 2

En esta iteración se crean los gráficos y tablas de datos por escuela que sirven de apoyo a la visualización generada.

Datos Escuela

Parametros elegidos:

Datos mostrados por: **Escuela**
Desde el año **2001** hasta el año **2009**

Indice	1-2001	2-2001	1-2002	2-2002	1-2003	2-2003	1-2004	2-2004	1-2005	2-2005	1-2006	2-2006	1-2007	2-2007	1-2008	2-2008	1-2009	2-2009	Total
Indice Aprobados	67.24%	60.64%	63.47%	67.72%	68.03%	65.02%	72.82%	64.90%	65.31%	67.35%	62.11%	59.26%	67.50%	59.76%	67.62%	60.30%	64.02%	0.00%	63.86%
Indice Aplazados	20.30%	28.88%	23.30%	12.87%	21.33%	22.38%	19.36%	21.03%	19.34%	18.11%	22.56%	28.72%	20.99%	23.51%	22.13%	22.85%	27.14%	0.00%	20.23%
Indice Retirados	12.46%	10.48%	13.22%	19.40%	10.63%	12.60%	7.82%	14.07%	15.35%	14.54%	15.33%	12.02%	11.50%	16.73%	10.25%	16.85%	8.84%	100.00%	15.90%
Indice Repitientes	24.30%	14.19%	28.92%	13.35%	20.57%	14.83%	23.69%	15.88%	28.93%	18.93%	29.71%	17.58%	29.34%	14.58%	27.72%	17.07%	24.67%	11.44%	19.87%
Cantidad Inscritos	1777	2125	1811	2816	2460	2841	2422	2663	2289	2554	2159	2603	1991	2471	1974	2684	2057	62	41054
Cantidad Aprobados	965	1111	979	1593	1392	1587	1362	1461	1218	1460	1164	1289	1066	1157	1048	1263	992	0	22264
Cantidad Aplazados	554	730	522	488	724	890	762	819	605	676	604	975	588	994	630	976	777	0	12377
Cantidad Retirados	554	730	522	488	724	890	762	819	605	676	604	975	588	994	630	976	777	0	6413

Figura 3.7: Datos de la Escuela de Computación desde el 2001 al 2009

3.4.1. Planificación de Requerimientos

Esta iteración satisface los siguientes requerimientos:

- Visualizar las tablas de datos correspondientes las materias de la Escuela de Computación por semestres asignados en el Pensum actual, entre el año 2001 y 2009.
- Generar una visualización por linea de tiempo de los datos extraídos por Escuela y por Materia.

3.4.2. Diseño

Se define el modelo entidad/relación (Figura 3.8). La implementación de la base de datos contiene las tabla `cursos`, `materias` y `licenciaturas`. La tabla `licenciaturas` puede almacenar más de una licenciatura en los registros.

La tabla `materias` almacena las asignaturas registradas en la base de datos del sistema **CONEST**. Las licenciaturas contienen materias que se dictan a traves de cursos registrados por periodo lectivo en la tabla `cursos`.

La tabla `cursos` almacena el numero de inscritos, aprobados, retirados, repitientes,

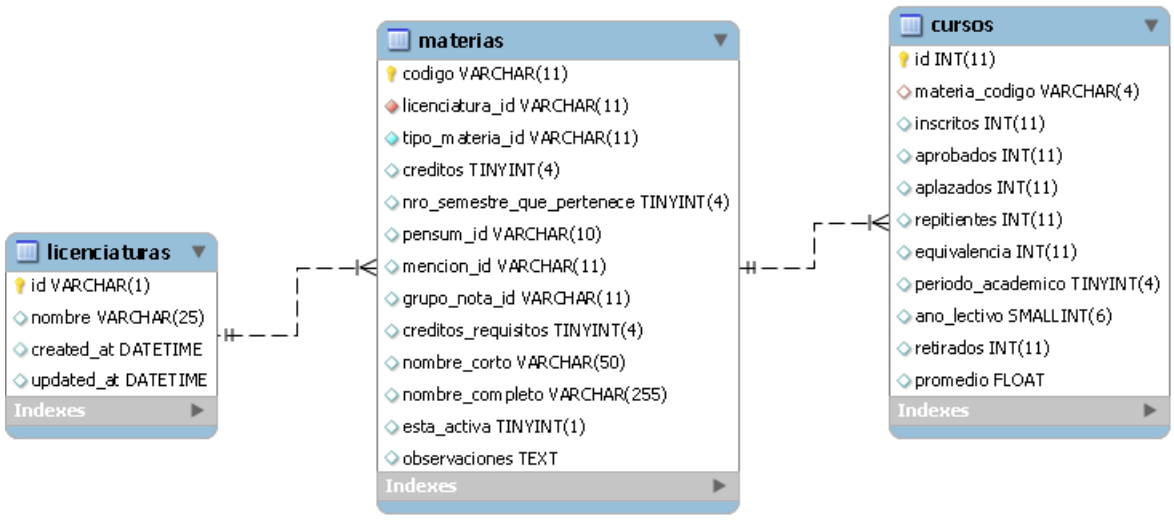


Figura 3.8: Modelo E/R con tablas y relaciones añadidas

promedio y equivalencia registrados en un curso dictado.

3.4.3. Generación de la aplicación

Para mostrar la tabla con los datos extraídos, el sistema debe tener como entradas los parámetros. Luego de realizar la búsqueda de los datos. Muestra la tabla con las materias. Se crea la accion `extraer` dentro del controlador llamado `CursosController` que muestra un formulario simple para la extraccion de los datos a mostrar (Fig. 3.6). El modelo `Curso` contiene el codigo suficiente para calcular los indices de aprobados, indices de aplazados, indices de retirados e indices de repitientes. El codigo fuente del modelo `Curso` se muestra a continuacion:

```
class Curso < ActiveRecord::Base
belongs_to :materia, :foreign_key=> :materia_codigo
def indice_eficiencia
(self.materia.creditos * self.aprobados).to_f /
(self.materia.creditos * (self.inscritos-self.equivalencia))
unless (self.materia.creditos * (self.inscritos-self.equivalencia)) == 0
end
```

```
def indice_aprobados
(self.aprobados).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
def indice_aplazados
(self.aplazados).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
def indice_retirados
(self.retirados).to_f /
(self.inscritos-self.equivalencia)
unless (self.inscritos-self.equivalencia) == 0
end
def indice_repitientes
(self.repitientes).to_f /
(self.inscritos-self.equivalencia) unless (self.inscritos-self.equivalencia) == 0
end
end
```

Se tienen las materias de una licenciatura, y estas a su vez contienen registrados cursos dictados mediante los distintos periodos lectivos. Para cumplir con este escenario, se asocian mediante la clave foranea `materia_codigo` los modelos `Curso` y `Materia` utilizando la funciones `belongs_to` y `has_many` respectivamente de la siguiente manera:

```
#/models/Curso.rb
class Curso < ActiveRecord::Base
belongs_to :materia, :foreign_key=> :materia_codigo
  ...
end
```

```
#/models/Licenciatura.rb
class Licenciatura < ActiveRecord::Base
has_many :materias
has_many :cursos, :through => :materias
end

#/models/Materia.rb
class Materia < ActiveRecord::Base

set_primary_key 'codigo'
has_many :datos, :foreign_key => :materia_codigo
belongs_to :licenciatura

end
```

Para calcular los índices y promedios acumulados por período lectivo, se crea una función llamada `All` dentro del módulo `Calculos`. Los filtros para la extracción de los datos son recibidos por el controlador `DatosController` que crea un arreglo con los objetos `Materia` y `Curso` e invoca a invoca a la función `all`. Luego se redirecciona a la acción `mostrar` que genera la vista con la tabla mostrada en la figura 3.7

La generación del gráfico de lineas se realiza utilizando las librerías Javascript JQuery y JQPlot. JQuery es un *framework* JavaScript ligero de úso público que contiene funciones para la manipulación de los elementos DOM, eventos, CSS y Ajax. Una de las ventajas de JQuery es su extensibilidad por medio de plugins de fácil instalación que agregan funciones extras como las funciones de graficación brindadas por JQPlot.

JQPlot es un plugin para JQuery que agrega funciones de graficación y permite la generación de visualizaciones de datos como gráficos lineales, gráficos de barra y gráficos de torta. Además, posee funciones para la manipulación del campo de la visualización. Tanto JQuery como JQPlot son librerías probadas en los buscadores Firefox, Chrome, Opera e Internet Explorer.

Para crear la visualización se utiliza el controlador `GraficoController` para modelar los datos y hacer la redirección a la vista apropiada. El controlador `GraficoController` contiene la acción mostrar con el siguiente código:

```
  def mostrar
    @datos = Licenciatura.find("C").get_all_datos(:ano_min=>params[:desde_ano],
       :ano_max=>params[:hasta_ano],
        :menciones=>[],
        :componentes=>[],
        :semestres=>[ params[:semestre1],
                  params[:semestre2],
                  params[:semestre3],
                  params[:semestre4],
                  params[:semestre5],
                  params[:semestre6]])
    @aprobados = @datos[:indices_aprobados].map{|k,v|[k,v[:indice]*100]}
    @repitientes = @datos[:indices_repitientes].map{|k,v|[k,v[:indice]*100]}
    @aplazados = @datos[:indices_aplazados].map{|k,v|[k,v[:indice]*100]}
    @retirados = @datos[:indices_retirados].map{|k,v|[k,v[:indice]*100]}
    @indices = [@aprobados,@aplazados,@retirados,@repitientes]
    @indices = @indices.to_json
    @labels_for_indices=["Indice Aprobados",
                         "Indice de Aplazados",
                         "Indice de Retirados",
                         "Indice de Repitientes"]
   end
end
```

El código anterior muestra la extracción de datos utilizando el modelo `Licenciatura` y la variable `params` que encapsula los parámetros provenientes de la vista del formulario. La vista `mostrar` invoca al helper `render_viz`. La función `render_viz` genera el

código Javascript suficiente para generar la visualización y renderizarla. La imagen generada se muestra en la figura 3.4.3

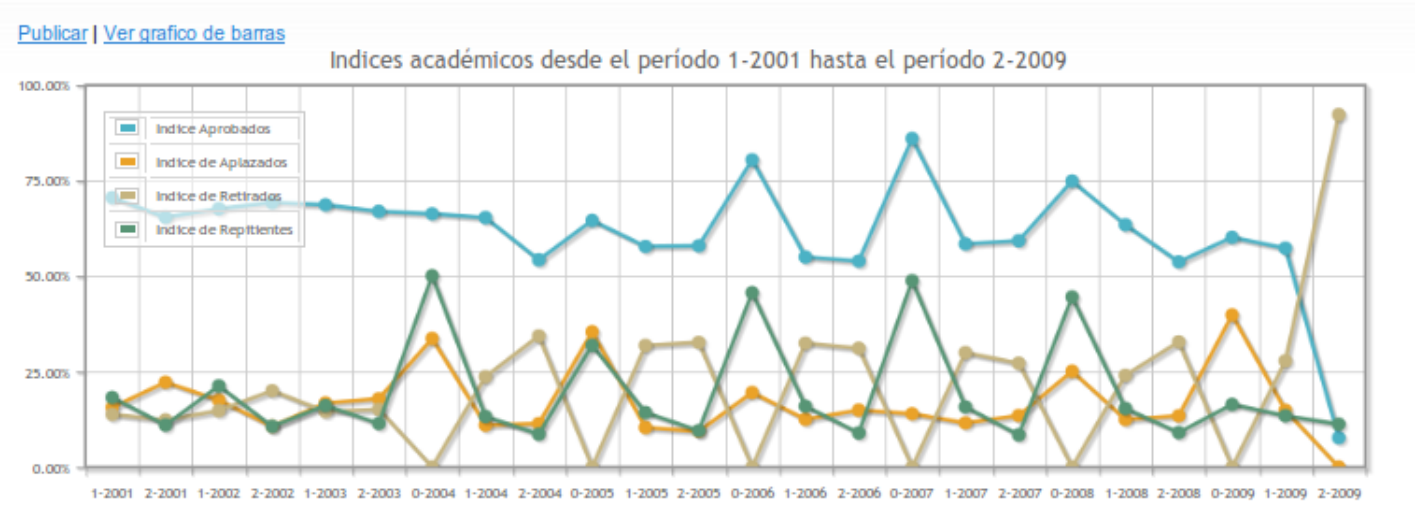


Figura 3.9: Prototipo de gráfico lineal

El código de la función `render_viz` se muestra a continuación:

```
 # Metodos agregados son disponibles en cualquier controlador desde
la vista.
module ApplicationHelper
def render_viz(labels,data,title)
    series=String.new
    series += labels.map{ |l| "{label:'#{l}'}," }.to_s
    logger.debug "SERIES #{series}"
    <<EOF
          <script type="text/javascript" language="javascript">
                jQuery(document).ready(function(){
              plot3 = jQuery.jqplot('chart1', #{data}, {
                  legend: {show:true,location:'nw'},
                  grid:{background:"#FFF;"},
                  title: "#{title}",
```

```
                    series:[#{series}],
                    axes:{
                          xaxis: {renderer:$.jqplot.CategoryAxisRenderer},
                          yaxis: {max:100,
                                  min:0,
                                  tickOptions:{formatString:'%.0i%'} }
                          },
                    cursor:{zoom:true, showTooltip:false, clickReset:true}


                  });
                  });
                </script>
EOF
  end
end
```

3.5. Iteración 3

3.5.1. Requerimientos

- Visualizar tabla de datos por asignaturas.
- Filtrar asignaturas por componentes.
- Visualizar tabla de datos por materias.
- Visualizar gráficos lineales para datos extraídos.
- Visualizar gráficos de barra para datos extraídos.
- Exportar datos como hojas de calculos.

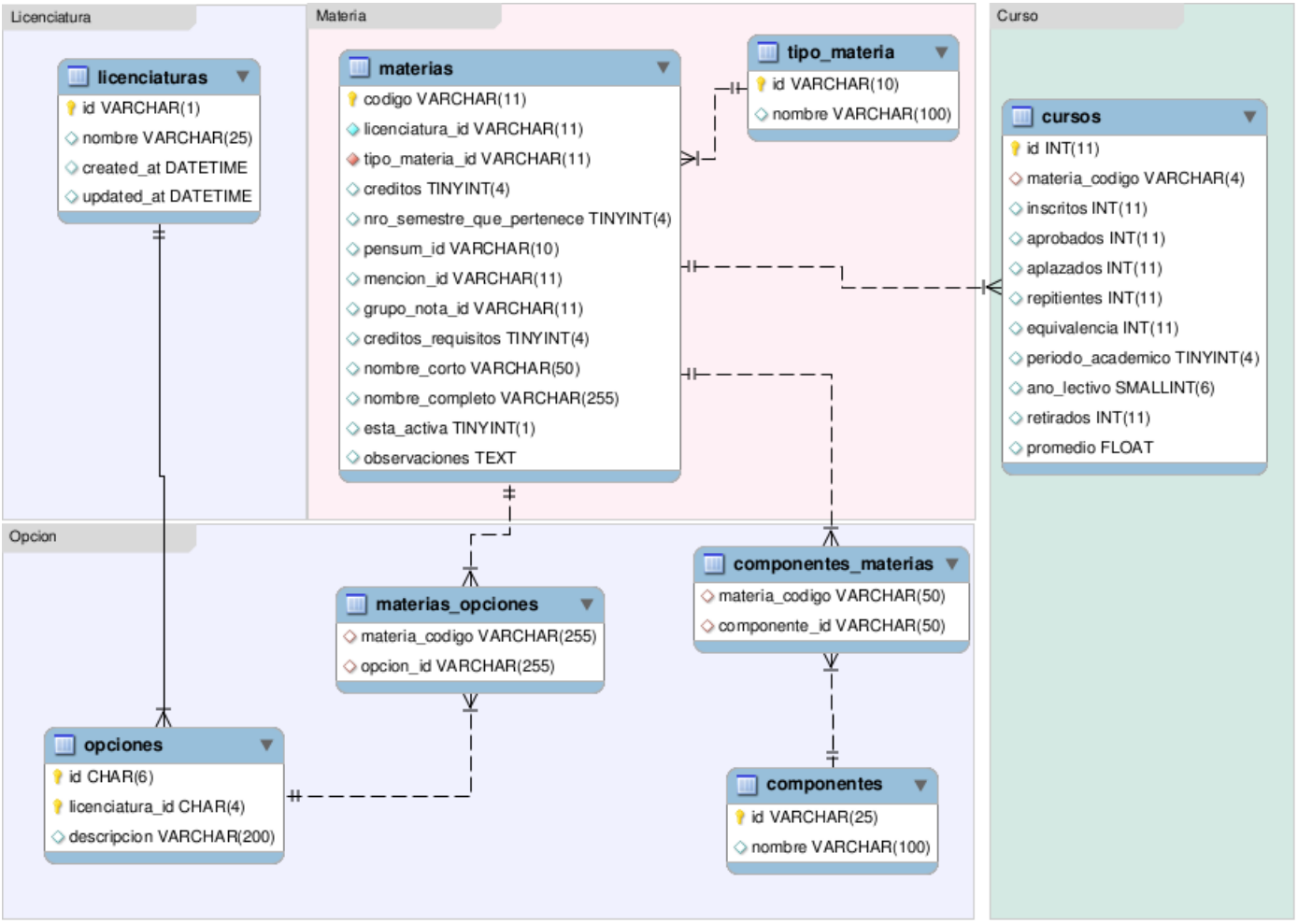
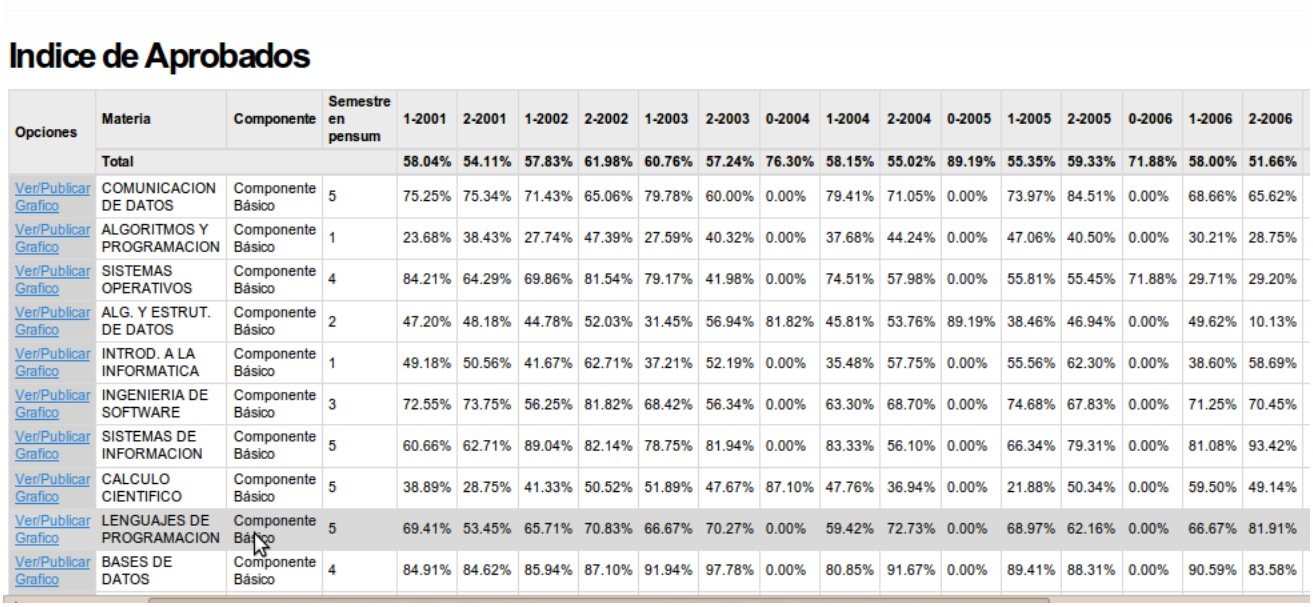


Figura 3.10: Modelo Entidad Relación Iteración 3

3.5.2. Diseño

Para mantener las entidades necesarias en el almacenamiento de los datos por asignaturas e índices, se realizó el modelo de la figura 3.10. En el modelo se muestran las entidades `Componentes` y `Menciones`. Las materias están relacionadas a menciones y a componentes según la última actualización del pensum. Se pueden diferenciar dos tipos de vista de datos por materia. La vista por índice académico, que muestra el índice académico por materia en el rango de períodos lectivos seleccionados(Fig.3.11) y la vista por período lectivo, que muestra todo el conjunto de materias y sus índices académicos por año y semestre seleccionado (Fig. 3.12). La tabla debe reflejar el componente y



Indice de Aprobados

Opciones	Materia	Componente	Semestre en pensum	1-2001	2-2001	1-2002	2-2002	1-2003	2-2003	0-2004	1-2004	2-2004	0-2005	1-2005	2-2005	0-2006	1-2006	2-2006
	Total			**58.04%**	**54.11%**	**57.83%**	**61.98%**	**60.76%**	**57.24%**	**76.30%**	**58.15%**	**55.02%**	**89.19%**	**55.35%**	**59.33%**	**71.88%**	**58.00%**	**51.66%**
Ver/Publicar Grafico	COMUNICACION DE DATOS	Componente Básico	5	75.25%	75.34%	71.43%	65.06%	79.78%	60.00%	0.00%	79.41%	71.05%	0.00%	73.97%	84.51%	0.00%	68.66%	65.62%
Ver/Publicar Grafico	ALGORITMOS Y PROGRAMACION	Componente Básico	1	23.68%	38.43%	27.74%	47.39%	27.59%	40.32%	0.00%	37.68%	44.24%	0.00%	47.06%	40.50%	0.00%	30.21%	28.75%
Ver/Publicar Grafico	SISTEMAS OPERATIVOS	Componente Básico	4	84.21%	64.29%	69.86%	81.54%	79.17%	41.98%	0.00%	74.51%	57.98%	0.00%	55.81%	55.45%	71.88%	29.71%	29.20%
Ver/Publicar Grafico	ALG. Y ESTRUT. DE DATOS	Componente Básico	2	47.20%	48.18%	44.78%	52.03%	31.45%	56.94%	81.82%	45.81%	53.76%	89.19%	38.46%	46.94%	0.00%	49.62%	10.13%
Ver/Publicar Grafico	INTROD. A LA INFORMATICA	Componente Básico	1	49.18%	50.56%	41.67%	62.71%	37.21%	52.19%	0.00%	35.48%	57.75%	0.00%	55.56%	62.30%	0.00%	38.60%	58.69%
Ver/Publicar Grafico	INGENIERIA DE SOFTWARE	Componente Básico	3	72.55%	73.75%	56.25%	81.82%	68.42%	56.34%	0.00%	63.30%	68.70%	0.00%	74.68%	67.83%	0.00%	71.25%	70.45%
Ver/Publicar Grafico	SISTEMAS DE INFORMACION	Componente Básico	5	60.66%	62.71%	89.04%	82.14%	78.75%	81.94%	0.00%	83.33%	56.10%	0.00%	66.34%	79.31%	0.00%	81.08%	93.42%
Ver/Publicar Grafico	CALCULO CIENTIFICO	Componente Básico	5	38.89%	28.75%	41.33%	50.52%	51.89%	47.67%	87.10%	47.76%	36.94%	0.00%	21.88%	50.34%	0.00%	59.50%	49.14%
Ver/Publicar Grafico	LENGUAJES DE PROGRAMACION	Componente Básico	5	69.41%	53.45%	65.71%	70.83%	66.67%	70.27%	0.00%	59.42%	72.73%	0.00%	68.97%	62.16%	0.00%	66.67%	81.91%
Ver/Publicar Grafico	BASES DE DATOS	Componente Básico	4	84.91%	84.62%	85.94%	87.10%	91.94%	97.78%	0.00%	80.85%	91.67%	0.00%	89.41%	88.31%	0.00%	90.59%	83.58%

Figura 3.11: Tabla de detalle de indicadores

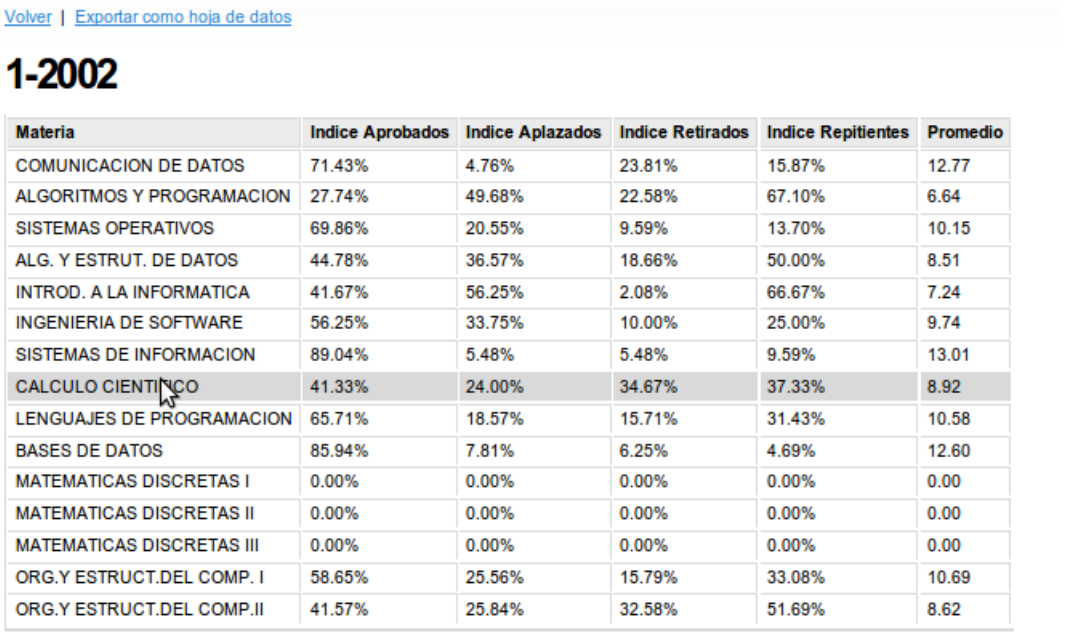
semestre de la materia.

3.5.3. Generación de la aplicación

Luego de generar los modelos diseñados de la figura 3.10 en la aplicación, se agrega el siguiente contenido al modelo `Materia`:

```
 class Materia < ActiveRecord::Base
include Calculos
set_primary_key 'codigo'
has_many :datos, :foreign_key => :materia_codigo
belongs_to :licenciatura
has_and_belongs_to_many :componentes, :foreign_key => :materia_codigo
  def indices_y_datos(ano_min,ano_max=nil,verano=false)

    datos = Curso.find(:all,:conditions=>["materia_codigo =
            ? AND ano_lectivo IN (?) AND periodo_academico <> 0",
self.codigo,ano_min ..ano_max]).sort_by{|d|d.ano_lectivo}
```



Volver | Exportar como hoja de datos

1-2002

Materia	Indice Aprobados	Indice Aplazados	Indice Retirados	Indice Repitientes	Promedio
COMUNICACION DE DATOS	71.43%	4.76%	23.81%	15.87%	12.77
ALGORITMOS Y PROGRAMACION	27.74%	49.68%	22.58%	67.10%	6.64
SISTEMAS OPERATIVOS	69.86%	20.55%	9.59%	13.70%	10.15
ALG. Y ESTRUT. DE DATOS	44.78%	36.57%	18.66%	50.00%	8.51
INTROD. A LA INFORMATICA	41.67%	56.25%	2.08%	66.67%	7.24
INGENIERIA DE SOFTWARE	56.25%	33.75%	10.00%	25.00%	9.74
SISTEMAS DE INFORMACION	89.04%	5.48%	5.48%	9.59%	13.01
CALCULO CIENTIFICO	41.33%	24.00%	34.67%	37.33%	8.92
LENGUAJES DE PROGRAMACION	65.71%	18.57%	15.71%	31.43%	10.58
BASES DE DATOS	85.94%	7.81%	6.25%	4.69%	12.60
MATEMATICAS DISCRETAS I	0.00%	0.00%	0.00%	0.00%	0.00
MATEMATICAS DISCRETAS II	0.00%	0.00%	0.00%	0.00%	0.00
MATEMATICAS DISCRETAS III	0.00%	0.00%	0.00%	0.00%	0.00
ORG.Y ESTRUCT.DEL COMP. I	58.65%	25.56%	15.79%	33.08%	10.69
ORG.Y ESTRUCT.DEL COMP.II	41.57%	25.84%	32.58%	51.69%	8.62

Figura 3.12: Tabla de datos de semestre seleccionado

```
return all(datos)
end
end
```

En el código anterior, se muestra la referencia al módulo `Calculos` mencionada en la sección 3.4.3, utilizado para obtener los índices acumulados mediante la función `all(datos)`.

En la vista, se crean las acciones `detalle` y `semestre` para mostrar las tablas por índices y por períodos respectivamente en el controlador `DatosController`. Con esto, se tiene la aplicación capaz de mostrar las tablas por materias.

Utilizando la función `render_viz` se modifican los parámetros para visualizar el gráfico por barras. El *plugin* JQPlot utiliza, a su vez, distintos *plugins* para formatear los datos, el campo, los ejes y la visualización en sí (Ej.: El *plugin* para visualizar los ejes X y Y en cadenas de caracteres es distinto al utilizado para mostrar valores numéricos).

La renderización por barras se logra incluyendo el *plugin* `jqplot.BarRenderer.js` al cargar la vista. Para renderizar el gráfico de tipo barras, se modifica la función `render_viz` para especificar el tipo de visualización por parámetros.

Las tablas por detalle en la figura 3.11 y semestre en la figura 3.12 son modeladas utilizando HTML, mostrando de cada materia, su nombre, el código, el componente al cual pertenece y los índices académicos.

Para exportar los datos como hojas de cálculos, se puede crear una función que genere el archivo por medio de la especificación XML Spreadsheet de la empresa *Microsoft*[1] que define un lenguaje extensible para la creación de hojas de cálculo. Esta solución, a pesar de aprovechar las ventajas del modelado por medio del lenguaje XML, genera sólamente documentos de hojas de cálculos compatibles con aplicaciones *Microsoft*.

Otra alternativa surge al recurrir a los diferentes componentes para Ruby llamados gemas[2]. La gema *spreadsheet*, es un componente que permite generar e interpretar hojas de cálculo con características como manejo de formatos, estilos y formulas. En nuestro caso, se utilizará la gema *SpreadSheet*[3]. La hoja de cálculo debe contener los mismos datos que se muestran en la tabla HTML.

3.6. Iteración 4

3.6.1. Requerimientos

- Publicar gráfico mediante un asistente para la generación y publicación.

- Gestionar usuarios y sesiones, así como la administración de las publicaciones y comentarios.

[1] http://msdn.microsoft.com/en-us/library/aa140066%28office.10%29.aspx

[2] Una gema o *Gem* es una pieza de código empaquetado siguiendo una especificación que extiende las funcionalidades de Ruby.

[3] http://spreadsheet.rubyforge.org

Figura 3.13: Diagramación de página principal

3.6.2. Diseño

En esta iteración se diseñan las interfaces e interacciones de la aplicación. Se desea implementar un menú por componentes del pensum que redireccione a sus filtros correspondientes. Los filtros por componentes se forman de la siguiente manera.

Componente Básico e Instrumental. Se filtra por materias pertenecientes a estos componentes, además de un selector para los rangos por períodos académicos.

Práctica Profesional. Filtros por laboratorios, trabajos de grado, seminarios, pasantías y servicio comunitario, junto al filtro de períodos académicos.

Componente Profesional Filtro por opción profesional. Cada opción contiene materias que la conforman. Al igual que los anteriores, debe contener un filtro por períodos académicos.

Complementario. Filtro por períodos académicos.

La interfaz del prototipo está basada en la plantilla de código abierto llamada *Fusion*, realizada por el grupo *ImageArts* y consiste en un diseño simple con un encabezado,

menu de navegación, contenido y pie de página. La interfaz del módulo de generación de datos para el prototipo se muestra en la figura 3.13. Esta diagramación consiste en un encabezado con los vínculos de inicio, foro, ayuda y usuario. El menú de la barra lateral derecha muestra los vínculos a la página de extracción de datos por componente.

El prototipo está compuesto por dos módulos. El módulo de generación de visualización y el módulo de Publicaciones o Foro (Figura 3.14). El modulo de generación de visualizaciones contiene las interfaces de extracción de datos, visualización de datos y graficación de datos. El módulo de foro brinda una interfaz simple para publicar y manejar noticias, comentarios y gráficos generados.

El módulo de Generación de visualizaciones permite filtrar los datos que se desean visualizar por componentes y asignaturas, además de permitir elegir la cronología de los datos por períodos lectivos. Luego se muestra la tabla con los índices académicos y el promedio que puede ser exportada como hoja de cálculo o explorada por índices académicos o por período lectivo.

Luego de obtener los datos tabulares, se puede ir en cualquier momento a la visualización por gráfico. En esta interfaz puede cambiarse el tipo de gráfico (Lineal y barras) además de permitir la publicación de éste.

3.6.3. Generación de la aplicación

Los modelos necesarios para la gestión de las publicaciones son `Entrada` y `Comentario`. Un usuario puede tener ninguna o muchas entradas, y a su vez, una entrada puede contener muchos comentarios, por lo que los modelos generados deben reflejar estas relaciones por medio de las asociaciones `has_many` y `belongs_to` de manera similar a la que se asociaron los modelos descritos en iteraciones anteriores.

El modelo `Modelo` se crea mediante el script `generate scaffold`, generando junto a éste el controlador `EntradasController` que contiene las acciones CRUD[4] y el enrutamiento de las vistas del foro.

La vista principal del foro muestra una tabla que contiene el título, tiempo transcur-

[4]Acrónimo de **C**reate, **R**ead, **U**pdate y **D**elete (Crear, leer, actualizar y eliminar).

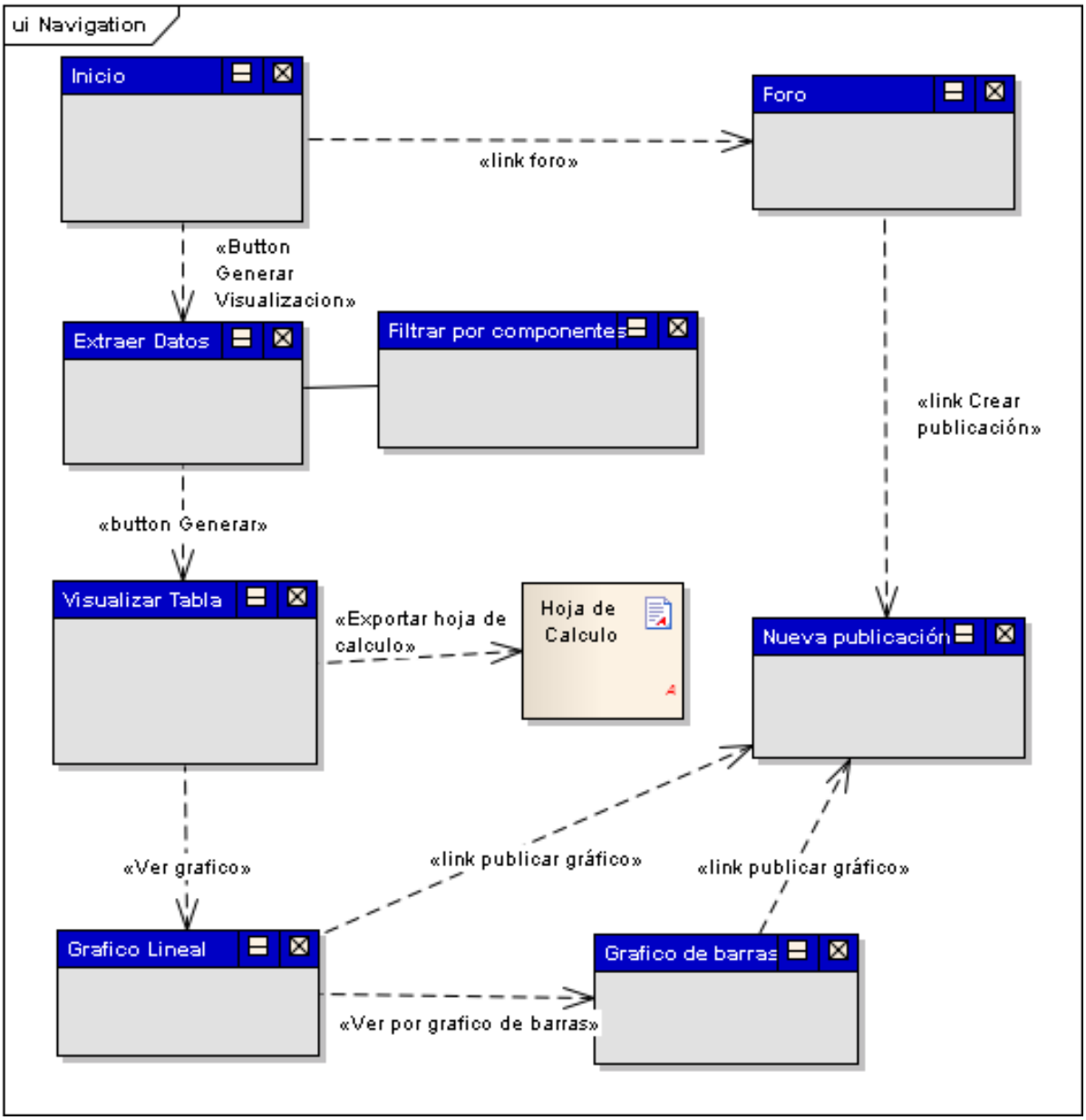


Figura 3.14: Mapa de navegacion

rido desde su creación, autor y número de comentarios realizados, tal como se muestra en la figura 3.15

Figura 3.15: Vista principal del Foro

Para incluir el gráfico en la publicación se utiliza la función `render_viz` con los parámetros que son almacenados como parte del registro en la base de datos en un objeto serializado `grafico`, que contiene el tipo, el título y los datos de la visualización publicada. Esto es posible en Rails agregando la linea `serialize :grafico` al modelo `Entrada`. La publicación creada junto con el gráfico se muestran en la figura 3.16

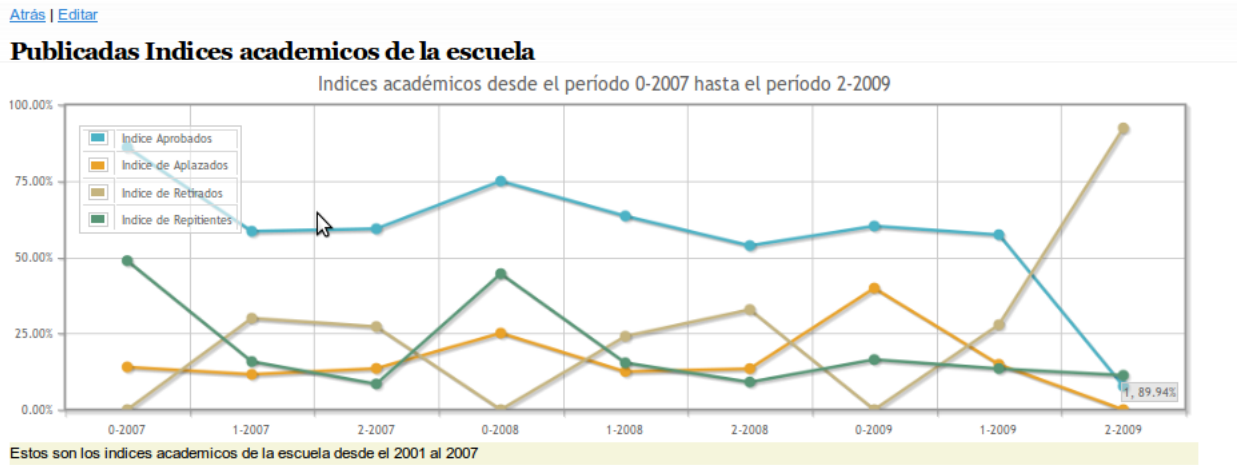


Figura 3.16: Interfaz de entrada del foro con visualización

Es importante el manejo de las sesiones y usuarios para garantizar la autenticidad y

la privacidad de la aplicación. Rails brinda funciones como `before_filter`, que permite realizar la validación del usuario antes de acceder al recurso solicitado. Para que esta validación se realice de manera adecuada, se genera el modelo `Usuario` con los atributos mostrados en la tabla `usuarios` correspondiente en la base de datos. El apéndice A.4 muestra el modelo entidad relación actualizado con la tabla `usuarios` y sus atributos. En la tabla `usuarios` se tienen los campos `salt` y `hashed_password` en lugar de un campo `password`. Esto se debe a que la creación y el almacenamiento de la contraseña se encripta con el sistema SHA1. Ruby cuenta con soporte para la encriptación con SHA1, y las aplicaciones Rails pueden implementarla facilmente.

El usuario solo puede editar o remover publicaciones creadas por él mismo. Así mismo, se tiene un usuario administrador que gestiona las entradas, usuarios y comentarios, así como otras funcionalidades descritas posteriormente. El usuario administrador en nuestra aplicación es único y sus funcionalidades son descritas en la aplicación Rails por medio de la variable `:namespace` en el archivo de configuración `config/routes.rb` que permite seleccionar las acciones y controladores que son accedidas por tipos de usuarios.

3.6.4. Implementación y Pruebas

Al generar un controlador o un modelo mediante el script `generate` de Rails, se generan los archivos para la corrida de pruebas. Para utilizar el ambiente de pruebas se crea una base de datos y se configura el framework en el archivo `database.yml` para acceder a dicha base de datos diferente a las bases de datos de desarrollo y producción. Esto es importante, debido a que en la base de datos de prueba se harán volcados, según sea el caso de pruebas.

Rails permite implementar las pruebas unitarias para los modelos y las fixtures para los controladores. El prototipo en esta iteración implementa pruebas unitarias, que definen un marco para realizar pruebas a un caso específico en el modelo y los *fixtures* que permiten realizar pruebas a nivel de entradas de datos en la vista.

Para la implementación y pruebas en ambiente de producción, se configura un servidor con acceso al grupo de usuarios seleccionado por la Comisión Curricular para re-

alizar esta tarea.

3.7. Iteración 5

3.7.1. Requerimientos

En esta iteración se tienen los siguientes requerimientos:

- Administrar la aplicación:
  - Crear/Modificar/Eliminar Usuarios.
  - Moderar entradas y comentarios.
  - Actualizar datos con respecto a la base de datos **CONEST**.

3.7.2. Diseño

Antes de pasar a la creacion del modulo de implementacion, se necesita diseñar el sistema de actualización de los datos. La aplicación debe ser capaz de actualizar los datos periodicamente para mantener la coherencia y la vigencia en los índices y visualizaciones publicadas. Por ello, se propone un sistema basado en servicios web y procedimientos almacenados en la base de datos Conest:

Lo primero que se debe tener en cuenta al momento de diseñar la propuesta es mantener el bajo acomplamiento entre las aplicaciones Conest y el Prototipo desarrollado. La solución más adecuada involucra un módulo que llamaremos `servidorCC` el cual contiene un único modelo de nombre Dato. Este modelo tiene la misma estructura del modelo Curso y contiene los datos generados por un procedimiento almacenado en la base de datos **CONEST**. Para no comprometer la confidencialidad de los datos, debe crearse un usuario de acceso limitado para la invocación del procedimiento.

El procedimiento almacenado se tiene en la base de datos **CONEST** y se ejecuta de manera programada cada fin de semestre. Al ejecutarse actualiza el modelo de nuestra aplicación servidor, por lo que puede diseñarse un sistema sencillo de paso de mensajes con un identificador de tipo *timestamp* de la fecha de la última actualización. Es

recomendable utilizar un canal seguro *https* para la transferencia de datos, además de implementar un mecanismo de autenticación http básico.

El método de transferencia se conoce como servicios RESTful. Al momento de acceder a una acción en un controlador, se especifica el tipo de archivo XML como requerido y este controlador contiene las rutas y la configuración necesaria para modelar el archivo y prepararlo para su envío.

El diagrama de actividades de la figura 3.17 muestra las entidades y el flujo de trabajo relacionados al proceso de actualización de datos:

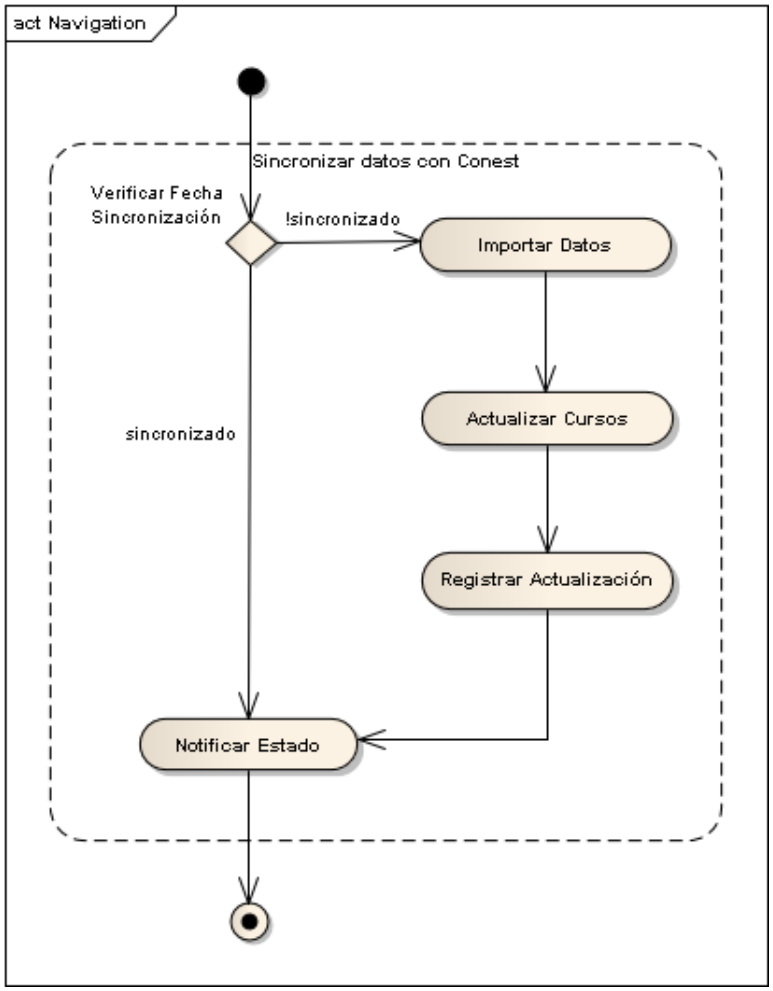


Figura 3.17: Flujo de actividades en la sincronización de datos

La sincronización comienza verificando la fecha de sincronización almacenada en la aplicación cliente con la del modulo servidor. Si las fecha registrada en el cliente

es menor a la del servidor, se considera que falta una sincronización, por lo que se actualizan los datos en el cliente y se registra la fecha de última sincronización. Por último, se envía un mensaje de notificación al usuario.

La administración de la aplicación se realiza agrupando las funciones **CRUD** de los modelos Usuario, Entrada y Comentario. Además de restringir el acceso a un usuario administrador.

3.7.3. Generación de la aplicación

Como primer paso se crea la aplicación `cc_server` que sirve de módulo servidor a nuestra aplicación para la sincronización con los datos almacenados en **CONEST**. Además, se crea una base de datos para la aplicación y se agrega el procedimiento almacenado mostrado en el Apéndice A.1 a la base de datos CONEST.

Para la conexión con el cliente, se generan el andamiaje para el modelo `Dato` mediante script `scaffold`. Los pasos para realizar la conexión se muestran en el apéndice A.2.

Al igual que los protocolos como XML-RPC o SOAP, REST sólo utiliza el protocolo HTTP para su transporte, por lo que el archivo de respuesta (Puede ser un archivo de tipo XML, JSON, XLS o cualquier otro archivo soportado por MIME) no necesita de otro protocolo adicional y se envía por medio del protocolo HTTP.

El manejo de un recurso REST en Rails se realiza por medio de la clase `ActiveResource`. Un objeto `ActiveResource` no es más que una abstracción de la conexión al recurso REST, brindando un objeto con funciones para el acceso al recurso REST.

En el siguiente código se muestra la clase `DatoResource` que extiende a `ActiveResource` utilizada para la sincronización de los datos:

```
class DatoResource < ActiveResource::Base
 self.site = "http://conest.ciens.ucv.ve:3000/"
 self.element_name = "dato"
 self.user = "usuario"
 self.password = "password"
end
```

La declaración del nombre del usuario y la contraseña, así como el sitio de hospedaje de la aplicación servidor están colocados como referencia. En la etapa de producción se definirán los datos reales, así como la implementación de un canal seguro https para el transporte de los datos.

El acceso a los recursos se realiza mediante la declaración de un objeto de tipo `DatoResource` de manera similar al acceso a las tablas mediante los demás objetos del modelo.

3.8. Iteración 6

En esta iteración se realiza la implementación de la aplicación y se comienza la fase de pruebas con el usuario.

3.8.1. Implementación y pruebas

Aprovechando las funcionalidades de Rails para el manejo de ambientes, la aplicación en producción se implanta redireccionada por un servidor web.

La base de datos de producción se genera mediante el comando `rake db:production:clone_structure`, que duplica la estructura de la base de datos de desarrollo.

Capítulo 4

Conclusiones

El proceso de visualización de índices académicos implementado reduce los tiempos de espera y permite a los miembros de la Comisión Curricular interactuar directamente con los datos extraídos de la base de datos **CONEST**. Esta integración se logra exitosamente mediante el uso de servicios web y el framework Rails.

Los datos almacenados en la base de datos de la aplicación permiten generar los índices de manera independiente al modelo de la base de datos **CONEST**. Lo cual permite su reuso en otras implementaciones.

Por otro lado, los gráficos generados fueron diseñados tomando en cuenta los requerimientos de visualización de la Comisión Curricular de la Escuela de Computación UCV. Además, se cumplieron las tareas relacionadas con la publicación de los datos generados en un foro de discusión, como herramienta de soporte para el análisis de los datos por parte de los miembros de la Comisión.

La interfaz para la extracción de los datos se diseñó de manera que los filtros pueden ser ordenados por componentes del pensum, y estos a su vez pueden ser filtrados por asignatura para obtener el resultado deseado. Para la generación de la interfaz se utilizaron distintos componentes del framework javascript jQuery que permitieron el reuso de código, reduciendo así los tiempos de entrega.

Se logró implementar un sistema de actualización de datos utilizando las tecnologías de servicios web soportadas por Rails. El uso de esta característica en la aplicación generada garantiza la integridad de los datos e índices almacenados en la base de datos con

respecto a la base de datos **CONEST**.

Durante el desarrollo de la aplicación, se pudo observar que la metodología RAD disminuyó considerablemente los tiempos de entrega en relación al sistema manual utilizado anteriormente para la generación de gráficos. Además, el tipo de desarrollo incremental en la metodología, junto a las constantes reuniones con el usuario y el uso de Rails en la aplicación permitieron la evolución a través de las iteraciones, a pesar de que surgieron requerimientos adicionales que no se especificaron al comienzo del desarrollo. Como resultado, se creó un proceso automatizado de extracción, modelado y visualización de índices académicos que sirve de soporte de decisiones y análisis a los miembros de la Comisión Curricular de la Facultad de Ciencias UCV.

Recomendaciones

Entre las recomendaciones se pueden mencionar:

- Integrar el sistema al registro de usuarios utilizado por **CONEST** para el grupo docente.
- Implementar la exportación de datos en formatos adicionales, como pdf, csv, entre otros.
- Implementar la exportación del gráfico generado a archivos de tipo imágen.
- Extender el uso de la aplicación para las diferentes escuelas de la Facultad de Ciencias. La aplicación generada puede utilizarse incorporando datos de las distintas licenciaturas de la Facultad de Ciencias, tomando en cuenta la organización curricular de cada una de ellas.

Apéndice A

Generación de Código

A.1. Actualizar datos

Para extraer y almacenar los datos en una base de datos independiente de **CONEST**, se desarrollo el siguiente procedimiento almacenado o *Stored Procedure*:

```
CREATE DEFINER=‘conest‘@‘localhost‘ PROCEDURE ‘cc_extraer_datos‘(IN ‘ano_min‘
smallint, IN ‘semestre_min‘ tinyint, IN ‘ano_max‘ smallint, IN ‘semestre_max‘ tinyint)
    MODIFIES SQL DATA
BEGIN

DECLARE  materia_semestre INT;
DECLARE pensum INT;
DECLARE i INT;
DECLARE a INT;
DECLARE ap INT;
DECLARE r INT;
DECLARE rep INT;
DECLARE eq INT;
DECLARE done INT DEFAULT 0;
DECLARE cod VARCHAR(4);
```

```
DECLARE ano SMALLINT;
DECLARE periodo TINYINT;
DECLARE prom FLOAT DEFAULT 0;
DECLARE cur1
CURSOR FOR
SELECT
materia_codigo,
ano_lectivo,
periodo_academico
FROM materia_en_periodo
WHERE licenciatura_id='C'
AND ano_lectivo >=ano_min
AND ano_lectivo <=ano_max;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
CREATE TEMPORARY TABLE IF NOT EXISTS calificacion_t
SELECT calificacion.estudiante_cedula,
calificacion.ano_lectivo,
calificacion.periodo_academico,
calificacion.licenciatura_id,
calificacion.materia_codigo as codigo,
tipo_nota.nombre as nota,
calificacion.tipo_nota_id_definitiva
FROM calificacion
LEFT OUTER JOIN tipo_nota
ON tipo_nota.id=calificacion.tipo_nota_id_definitiva
WHERE calificacion.licenciatura_id='C' AND calificacion.ano_lectivo >=ano_min AND califi

OPEN cur1;
REPEAT
```

```
FETCH cur1 INTO cod,ano,periodo;
IF NOT done THEN
#Inscritos
SELECT count(c.estudiante_cedula) INTO i
FROM calificacion_t as c WHERE c.ano_lectivo=ano
AND c.periodo_academico=periodo AND c.codigo=cod;
#Aprobados
SELECT count(c.estudiante_cedula) INTO a
FROM calificacion_t as c WHERE
c.tipo_nota_id_definitiva<=23
AND c.tipo_nota_id_definitiva>=12AND c.ano_lectivo=ano
AND c.periodo_academico=periodoAND c.codigo=cod;
#Aplazados

SELECT count(c.estudiante_cedula)
INTO ap
FROM calificacion_t as c
WHERE
(c.tipo_nota_id_definitiva<=11 AND c.tipo_nota_id_definitiva>=1
OR c.tipo_nota_id_definitiva=28) AND c.ano_lectivo=ano
AND c.periodo_academico=periodo AND c.codigo=cod;

#Retirados
SELECT count(c.estudiante_cedula)
INTO r
FROM calificacion_t as c
WHERE
c.tipo_nota_id_definitiva<=27 AND c.tipo_nota_id_definitiva>=26
AND c.ano_lectivo=ano AND c.periodo_academico=periodo
AND c.codigo=cod;
```

```
#Equivalencias
SELECT count(c.estudiante_cedula) INTO eq
FROM calificacion_t as c WHERE
c.tipo_nota_id_definitiva<=25
AND c.tipo_nota_id_definitiva>=24 AND c.ano_lectivo=ano
AND c.periodo_academico=periodo AND c.codigo=cod;

#Repitientes
SELECT count(c.estudiante_cedula)
INTO rep
FROM calificacion_t as c
WHERE c.licenciatura_id='C' AND c.ano_lectivo=ano
AND c.periodo_academico=periodo AND c.codigo = ANY

(

SELECT DISTINCT(ce.materia_codigo)

FROM calificacion as ce, estudiante_en_licenciatura as eel

WHERE ce.ano_lectivo<=ano

AND !(ce.ano_lectivo=ano AND
      ce.periodo_academico=periodo)

AND ce.ano_lectivo>=eel.ano_lectivo_ingreso

AND eel.estudiante_cedula=c.estudiante_cedula
```

```
AND ce.estudiante_cedula=c.estudiante_cedula

AND (ce.tipo_nota_id_definitiva<=11
AND ce.tipo_nota_id_definitiva>=1
OR ce.tipo_nota_id_definitiva=28)

)

AND c.codigo=cod;

#Extraer promedio
SELECT AVG(nota)
INTO prom
FROM calificacion_t
WHERE
codigo=cod
AND ano_lectivo=ano AND periodo_academico=periodo
AND nota!='RET' AND nota!='AP'
AND nota!='A' AND nota!='EQ';
INSERT IGNORE
INTO servidorCC_production.datos(
periodo_academico,
ano_lectivo,materia_codigo,inscritos,aprobados,
retirados,aplazados,repitientes,equivalencia,promedio)
VALUES(periodo, ano,cod,i,a,r,ap,rep,eq,prom)
ON DUPLICATE KEY
UPDATE inscritos=i, aprobados=a, retirados=r,
aplazados=ap, repitientes = rep,
equivalencia = eq, promedio=prom;
END IF;
```

```
UNTIL done END REPEAT;
CLOSE cur1;



END
```

Este procedimiento puede ser almacenado en la base de datos **CONEST** y programado para ejecutarse cada fin de semestre. El procedimiento modifica el contenido de la tabla `Datos` de la base de datos de producción de nuestra aplicación servidor `servidorCC`.

Para asegurar la consistencia de los datos en el tiempo, el procedimiento actualiza los registros de las tablas que contengan los campos `ano_lectivo`, `materia_codigo` y `periodo_academico`.

A.2. Servicio Web REST

El patrón de diseño **REST** consiste principalmente en utilizar los tipos de peticiones `HTTP POST`, `GET`,`PUT` y `DELETE` para realizar las acciones de modificar, acceder, agregar y eliminar respectivamente sobre los elementos de la capa de Modelo.

En el siguiente código se muestra un Controlador que implementa las acciones **REST**.

```
class DatosController < ApplicationController
  # GET /datos
  # GET /datos.xml
  def index
    @datos = Dato.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @datos }
    end
  end
```

```
# GET /datos/1
# GET /datos/1.xml
def show
  @dato = Dato.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.xml  { render :xml => @dato }
  end
end

# GET /datos/new
# GET /datos/new.xml
def new
  @dato = Dato.new

  respond_to do |format|
    format.html # new.html.erb
    format.xml  { render :xml => @dato }
  end
end

# GET /datos/1/edit
def edit
  @dato = Dato.find(params[:id])
end

# POST /datos
# POST /datos.xml
```

```
def create
  @dato = Dato.new(params[:dato])

  respond_to do |format|
    if @dato.save

      format.html { redirect_to(@dato) }
      format.xml  { render :xml => @dato, :status => :created, :location => @dato }
    else
      format.html { render :action => "new" }
      format.xml  { render :xml => @dato.errors, :status => :unprocessable_entity }
    end
  end
end

# PUT /datos/1
# PUT /datos/1.xml
def update
  @dato = Dato.find(params[:id])

  respond_to do |format|
    if @dato.update_attributes(params[:dato])

      format.html { redirect_to(@dato) }
      format.xml  { head :ok }
    else
      format.html { render :action => "edit" }
      format.xml  { render :xml => @dato.errors,
                    :status => :unprocessable_entity }
    end
```

```
    end
  end

  # DELETE /datos/1
  # DELETE /datos/1.xml
  def destroy
    @dato = Dato.find(params[:id])
    @dato.destroy

    respond_to do |format|
      format.html { redirect_to(datos_url) }
      format.xml  { head :ok }
    end
  end
  # GET /datos/1
  # GET /datos/1.xml
  def sync(year)
    @datos = Dato.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @datos }
    end

  end
end
```

Como se puede ver, en cada acción del controlador se invoca al bloque `respond_to` cuya función es preparar el tipo de archivo que se desea en la respuesta. Por ejemplo, la acción `sync` contiene en su bloque de respuesta el formato .html y .xml. Si se desea la respuesta de tipo xml se accede por medio de la petición `http://dominio/datos/sync.xml`

A.3. Función render_viz

La función o *helper* desarrollado para generar la visualización se muestra a continuación:

```
  def render_viz(labels,data,title,format='line',id='chart1')

   bars = "seriesDefaults: {
                                      renderer: $.jqplot.BarRenderer,
                                      rendererOptions: {
                                      barDirection: 'vertical',
                                      barPadding: 6,
                                      barMargin: 40
                                  }
                               }," if format.eql? "bars"
   series=String.new

   series += labels.map{ |l| "{label:'#{l}'}," }.to_s
   <<EOF
          <script type="text/javascript" language="javascript">

                   jQuery(document).ready(function(){


                plot = jQuery.jqplot('#{id}', #{data}, {
                                                          highlighter: {
   tooltipAxes: 'y',
   useAxesFormatters: 'x'
},
                     legend: {show:true,location:'nw',offset:'-99px'},
                     grid:{background:"#FFF;"},
```

```
                    title: "#{title}",
                    #{bars}
                    series:[#{series}],
                    axes:{
                          xaxis: {renderer: $.jqplot.CategoryAxisRenderer,

                                  },
                          yaxis: {max:100,
                                  min:0,
                                  tickOptions:{formatString:'%.2f%'} }
                          },
                    cursor:{zoom:true, showTooltip:true, clickReset:true}
                  });
                  });

                </script>
EOF

  end
```

El código generado es un Javascript que se incluye dentro del elemento `head` del archivo HTML. Los parámetros contienen un arreglo con las series a graficar, los datos, el tÃtulo de cada serie, el tipo de gráfico (barras o lineal) y el identificador donde se desea generar la visualización.

A.4. Modelo E/R Final

Se muestra a continuación el modelo Entidad Relación utilizado por la aplicación.

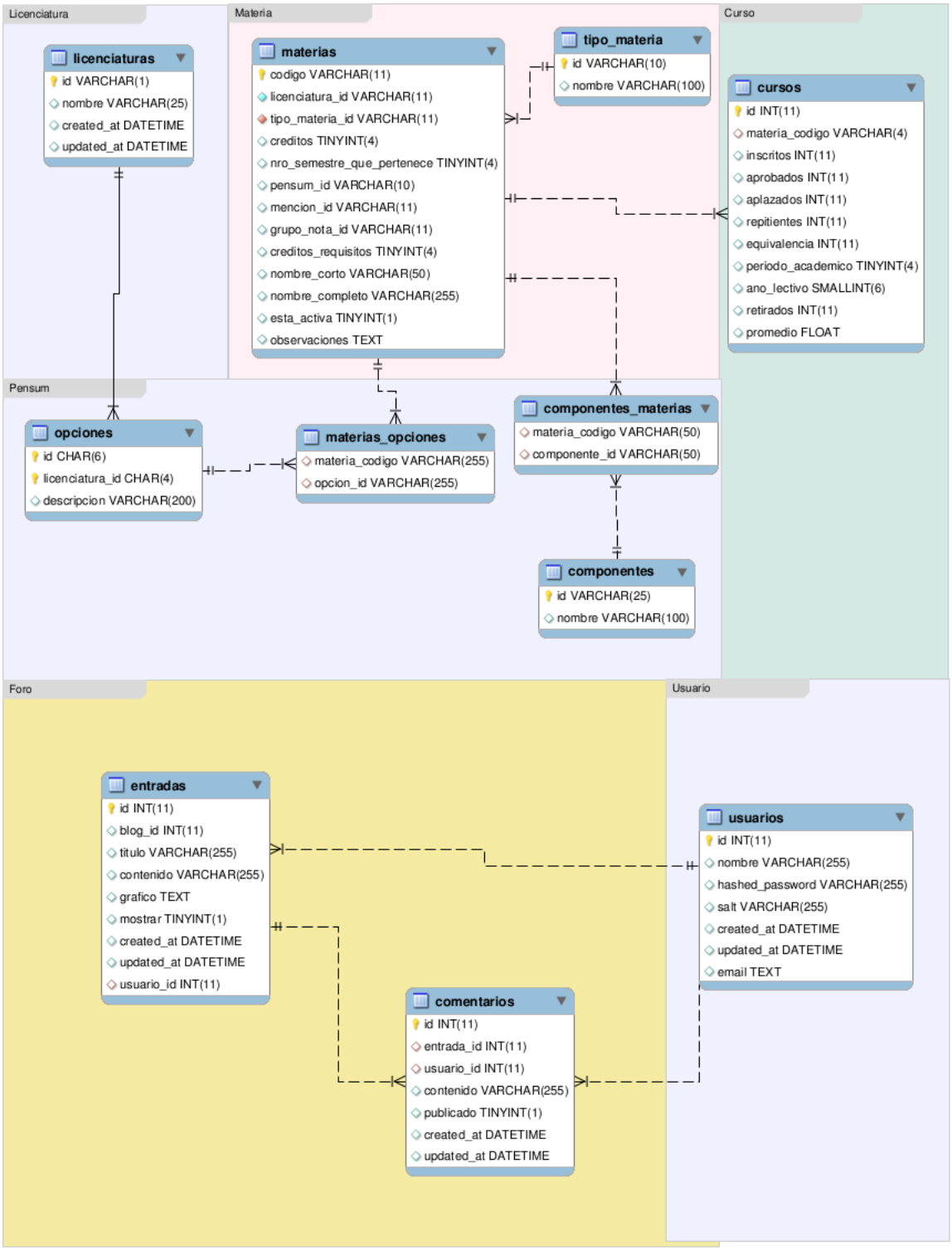


Figura A.1: Modelo E/R actualizado

Bibliografía

[1] Michael Champion, Chris Ferris, Eric Newcomer, and David Orchard. Web services architecture. Technical report, 2002.

[2] Usama M. Fayyad, Georges G. Grinstein, and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Elsevier, 2002.

[3] Vitaly Friedman. Data visualization: Modern approaches. *Smashing Magazine*, 2007. `http://www.smashingmagazine.com/2007/08/02/data-visualization-modern-approaches/`.

[4] Vitaly Friedman. Data visualization and infographics. *Smashing Magazine*, 2008. `http://www.smashingmagazine.com/2008/01/14/monday-inspiration-data-visualization-and-infographics/`.

[5] M. Friendly. Milestones in the history of thematic cartography, statistical graphics, and data visualization. Technical report, 2008.

[6] C. Hibbs and B. Tate. *Ruby on Rails: Up and Running*. O' Reilly, 2006.

[7] James Kerr and Richard Hunter. *Inside RAD: how to build fully functional computer systems in 90 days or less*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[8] James Kerr and Richard Hunter. *Inside RAD: how to build fully functional computer systems in 90 days or less*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[9] Craig Larman. *Aplying UML Aplying UML and Patterns. An introduction to Object-Oriented Analysis and Design and Unified Process*. Prentice Hall, 2nd edition, 2001.

[10] J. Martin. *Rapid Application Development*. MacMillan, New York, 1991.

[11] R. Orsini. *Rails Cookbook*. O' Reilly, 2007.

[12] Frits H. Post, Gregory M. Nielson, and Georges-Pierre Bonneau, editors. *Data Visualization: The State of the Art*. Kluwer Academic Publishers, 2003.

[13] Roger Pressman. *Ingeniería de Software, Un Enfoque Práctico*. McGraw Hill, fifth edition.

[14] S. Rubym and D. Thomas. *Agile Web Development with Rails*. The Pragmatic Programmers, LLC., third edition, 2008.