Proyecto de Sistemas Informáticos

Curso académico 2008 / 2009

Sistema de gestión de información de usuarios y resultados para el paquete informático GeneCodis

Autores

Victor Acón Aceña

Eva García Vega

Profesor Director

Alberto Pascual-Montano

Dpto. de Arquitectura de Computadores y Automática

Facultad de Informática. Universidad Complutense de Madrid

Resumen

GeneCodis cuenta en la actualidad con una media de 500 trabajos reales provenientes de distintas partes del mundo, en especial de Europa, USA y Japón. Así mismo, el número de procesos realizados por el mismo usuario es también alto y la tendencia es aumentar. Este número elevado de procesos por usuario hace que la gestión de la información sea imprecisa poco fiable y prácticamente imposible de gestionar de una manera organizada ya que la manera de notificación existente en la actualidad está basada en el correo electrónico o en el almacenamiento manual de las URL con los resultados, por lo tanto, este proyecto pretende minimizar estos problemas mediante la realización de una gestión de los trabajos.

Palabras Claves

- GeneCodis
- Ruby
- Rails
- Camping
- Bioinformática
- Análisis funcional
- Bases de datos
- HTML

Abstract

Nowadays, Genecodis has about 500 real works which come from different places of the world, especially from Europe, USA and Japan. Moreover, the number of process which is realized by the same user is usually large and it normally goes on increasing. This large number produces that information management could be vague and very little reliable. Furthermore, notifications are sending by e-mail or storage by URL and the results, so that, tidily management is almost impossible. Therefore, this project wants to decrease this kind of problems thanks to a work management.

Keywords

- GeneCodis
- Ruby
- Rails
- Camping
- Bioinformatics
- Functional analysis
- Database

Autorización

Autorizamos a la facultad de Informática de la Universidad Complutense de Madrid, así como al resto de sus centros adscritos a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado: …….. ……..

Victor Acón Aceña Eva García Vega

ÍNDICE

1. INTRODUCCIÓN

1.1. Introducción a la bioinformática

1.1.1 ¿Qué es la bioinformática?

Bioinformática [1] es una disciplina científica emergente que utiliza tecnología de la información para organizar, analizar y distribuir información biológica con la finalidad de responder preguntas complejas en biología. Bioinformática es un área de investigación multidisciplinaria, la cual puede ser ampliamente definida como la interfase entre dos ciencias: Biología y Computación y está impulsada por la incógnita del genoma humano y la promesa de una nueva era en la cual la investigación genómica puede ayudar dramáticamente a mejorar la condición y calidad de vida humana.

Avances en la detección y tratamiento de enfermedades y la producción de alimentos genéticamente modificados son entre otros ejemplos de los beneficios mencionados más frecuentemente. Involucra la solución de problemas complejos usando herramientas de sistemas y computación. También incluye la colección, organización, almacenamiento y recuperación de la información biológica que se encuentra en base de datos.

Según la definición del Centro Nacional para la Información Biotecnológica "National Center for Biotechnology Information" (NCBI por sus siglas en Inglés, 2001): "Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información. El fin último de este campo es facilitar el descubrimiento de nuevas ideas biológicas así como crear perspectivas globales a partir de las cuales se puedan discernir principios unificadores en biología. Al comienzo de la "revolución genómica", el concepto de bioinformática se refería sólo a la creación y mantenimiento de base de datos donde

se almacena información biológica, tales como secuencias de nucleótidos y aminoácidos. El desarrollo de este tipo de base de datos no solamente significaba el diseño de la misma sino también el desarrollo de interfaces complejas donde los investigadores pudieran acceder los datos existentes y suministrar o revisar datos.

Luego toda esa información debía ser combinada para formar una idea lógica de las actividades celulares normales, de tal manera que los investigadores pudieran estudiar cómo estas actividades se veían alteradas en estados de una enfermedad. De allí viene el surgimiento del campo de la bioinformática y ahora el campo más popular es el análisis e interpretación de varios tipos de datos, incluyendo secuencias de nucleótidos y aminoácidos, dominios de proteínas y estructura de proteínas.

El proceso de analizar e interpretar los datos es conocido como biocomputación. Dentro de la bioinformática y la biocomputación existen otras sub-disciplinas importantes: El desarrollo e implementación de herramientas que permitan el acceso, uso y manejo de varios tipos de información
El desarrollo de nuevos algoritmos (fórmulas matemáticas) y estadísticos con los cuales se pueda relacionar partes de un conjunto enorme de datos, como por ejemplo métodos para localizar un gen dentro de una secuencia, predecir estructura o función de proteínas y poder agrupar secuencias de proteínas en familias relacionadas."

La Medicina molecular y la Biotecnología constituyen dos áreas prioritarias científico-tecnológicas como desarrollo e Innovación Tecnológica. El desarrollo en ambas áreas está estrechamente relacionado. En ambas áreas se pretende potenciar la investigación genómica y postgenómica así como de la bioinformática, herramienta imprescindible para el desarrollo de estas debido al extraordinario avance de la genética molecular y la genómica, la Medicina Molecular se constituye como arma estratégica del bienestar social del futuro inmediato. Se pretende potenciar la aplicación de las nuevas tecnologías y de los avances genéticos para el beneficio de la salud. Dentro de las actividades financiables, existen acciones

estratégicas, de infraestructura, centros de competencia y grandes instalaciones científicas. En esta área, la dotación de infraestructura se plasmará en la creación y dotación de unidades de referencia tecnológica y centros de suministro común, como Centros de Bioinformática, que cubran las necesidades de la investigación en Medicina Molecular. En cuanto a centros de competencia, se crearán centros de investigación de excelencia en hospitales en los que se acercará la investigación básica a la clínica, así como centros distribuidos en red para el apoyo a la secuenciación, DNA microarrays y DNA chips, bioinformática, en coordinación con la red de centros de investigación genómica y proteómica que se proponen en el área de Biotecnología. En esta área la genómica y proteómica se fundamenta como acción estratégica o instrumento básico de focalización de las actuaciones futuras.

Las tecnologías de la información jugarán un papel fundamental en la aplicación de los desarrollos tecnológicos en el campo de la genética a la práctica médica como refleja la presencia de la Bioinformática médica y la Telemedicina dentro de las principales líneas en patología molecular. La aplicación de los conocimientos en genética molecular y las nuevas tecnologías son necesarias para el mantenimiento de la competitividad del sistema sanitario no sólo paliativo sino preventivo. La identificación de las causas moleculares de las enfermedades junto con el desarrollo de la industria biotecnológica en general y de la farmacéutica en particular permitirán el desarrollo de mejores métodos de diagnóstico, la identificación de dianas terapéuticas y desarrollo de fármacos personalizados y una mejor medicina preventiva

1.2. Introducción al análisis funcional

El análisis de datos de expresión mediante técnicas estadísticas y de detección de patrones suministra información muy valiosa acerca de qué genes tienen un patrón de expresión similar en respuesta a determinados estímulos o están diferencialmente expresados en distintos tipos de muestras. No obstante, el propósito último en el proceso de análisis de datos generados mediante microchips de ADN es el de extraer información biológica relevante que permita interpretar qué procesos biológicos o redes de regulación son los responsables o están asociados a los cambios en los patrones de expresión. Este tipo conocimiento no puede ser inferido de los datos de expresión en si por lo que se requiere la incorporación de otras fuentes de información acerca de distintas propiedades de los genes y proteínas como por ejemplo anotaciones funcionales, regiones reguladoras en promotores o la información contenida en la literatura de los genes de interés.

En los primeros trabajos en el campo, este nivel en el proceso analítico se abordaba con un análisis manual, aunque resultaba tedioso e ineficiente y muy pronto se empezaron a desarrollar metodologías que permiten llevar a cabo un análisis automático de la información biológica contenida en grandes listas de genes así como metodologías capaces de integrar datos de expresión con otros tipos de datos biológicos.

1.2.1 Extracción de información biológica a partir de listas de genes

Una forma de extraer información biológica a partir de grandes listas de genes es determinar anotaciones funcionales que están presentes de forma significativa en la lista de genes a analizar. Estas anotaciones están disponibles en distintas bases de datos y hay una gran batería de métodos para llevar a cabo este tipo análisis, aunque gran parte de las mismas están centrados en el análisis de anotaciones de *Gene Ontology* (GO) (Ashburner *et al*., 2000). El consorcio de *GO* ha

desarrollado un vocabulario estandarizado y dinámico que describe los productos génicos a tres niveles diferentes: Función Molecular, Proceso Biológico y Componente Celular. Distintas iniciativas han llevado a cabo un proceso de anotación de genomas completos utilizando las categorías de GO por lo que esta información se puede utilizar para la extracción automática de información biológica a partir de grandes listas de genes. En la página web de GO hay un amplio listado de herramientas para el análisis funcional de listas de genes (http://www.geneontology.org/).

Todos estos métodos trabajan de una forma similar y están basados en la búsqueda de anotaciones que están enriquecidas de forma significativa en una lista de genes con respecto a una lista de referencia. Utilizando una fuente de anotación, por ejemplo la categoría de Proceso Biológico, estos métodos buscan todas las anotaciones que están asociadas con los genes en la lista de interés. Para cada anotación se determina el número de genes asociados a la misma en la lista de interés y en una lista de referencia, normalmente el genoma completo o todos los genes incluidos en el chip. Con estos valores se puede aplicar un test estadístico, normalmente basado en la distribución hipergeométrica, binomial, el test de la Chi-cuadrado o el test exacto de Fisher para determinar un *p*-valor para cada anotación, los cuales se ajustan para evitar el problema de las comparaciones múltiples. Aquellas anotaciones con *p*-valores significativos se consideran descriptoras de los genes presentes en la lista de interés y de los eventos biológicos asociados al sistema experimental. La idea que subyace a este tipo de análisis es que si una proporción significativa de todos los genes asociados a un determinado proceso biológico están presentes en la lista de interés, por ejemplo genes expresados en un tejido patológico, se puede asumir que ese proceso está asociado con la característica fenotípica responsable del patrón de expresión. Una buena revisión de este tipo de metodologías se puede encontrar en (Dopazo, 2006; Khatri y Draghici, 2005).

A parte de anotaciones extraídas directamente de bases de datos como GO o KEGG (http://www.genome.jp/kegg/) o de este tipo de análisis estadístico basado en

la distribución de anotaciones en dos listas de genes también se han explorado otras fuentes de información como términos de MeSH (Djebbari *et al*., 2005; Masys, 2001) o el análisis de texto libre (Shatkay *et al*., 2000), así como otras aproximaciones más sofisticadas basadas en el análisis de la distribución de anotaciones en el conjunto total de genes ordenados por su expresión diferencial en dos condiciones experimentales (Al-Shahrour *et al*., 2005a; Subramanian *et al*., 2005).

1.2.2 Integración de información biológica y datos experimentales

El segundo paso es la integración de información biológica de genes y proteínas. Un tipo de aproximación más sofisticada consiste en la integración y el análisis simultáneo de datos de expresión con otros tipos de información funcional de genes y proteínas para identificar asociaciones existentes entre ellos. Este tipo de análisis integrado ha centrado la atención de muchos investigadores y en los últimos años varios métodos han sido propuestos en este contexto. Por ejemplo, Alter y Golub integraron datos de expresión con datos de experimentos de *ChIP on chip* descubriendo nuevas correlaciones entre proteínas involucradas en la iniciación de la replicación y la transcripción de algunos genes durante el ciclo celular (Alter y Golub, 2004). Para ello crearon una matriz para cada tipo de datos y usaron la proyección con la pseudoinversa para la integración de ambas fuentes de información. Tanay *et al*. construyeron matrices en las que a cada gen se le asignaron distintos tipos de características como expresión, interacción proteína-proteína o unión de factores de transcripción y mediante el uso de un algoritmo de agrupamiento fueron capaces de determinar distintos módulos génicos (grupos de genes que presentaban una alta similitud a lo largo de fuentes heterogéneas de datos) y predecir la función de más de 800 genes desconocidos (Tanay *et al*., 2004). Cui *et al*. usaron un análisis de homogeneidad para integrar datos de expresión con información funcional de genes y proteínas para la extracción de patrones de correlación entre ambos tipos de datos (Cui *et al*., 2004).

Con la generación de datos biológicos cada vez más diversos es deseable la utilización de metodologías capaces de integrar todas estas fuentes de información para la extracción de conocimiento biológico. El descubrimiento de asociaciones y patrones entre diferentes tipos de datos puede generar información más completa y rica para la interpretación de los procesos biológicos que subyacen a los distintos programas de expresión génica o para la predicción de la función de genes desconocidos (Troyanskaya, 2005).

1.3. Introducción a GeneCodis

GeneCodis [2] es una herramienta para el análisis funcional de grandes listas de genes basado en la extracción de anotaciones concurrentes.

A partir de una lista de genes se determinan las anotaciones biológicas y combinaciones de las mismas que están estadísticamente enriquecidas en esta lista con respecto a una lista de referencia. Antes de aplicar el test estadístico se extraen todas las combinaciones de anotaciones que están presentes en al menos *x* genes, siendo *x* un valor definido por el usuario.

Su funcionamiento queda descrito por la siguiente imagen:

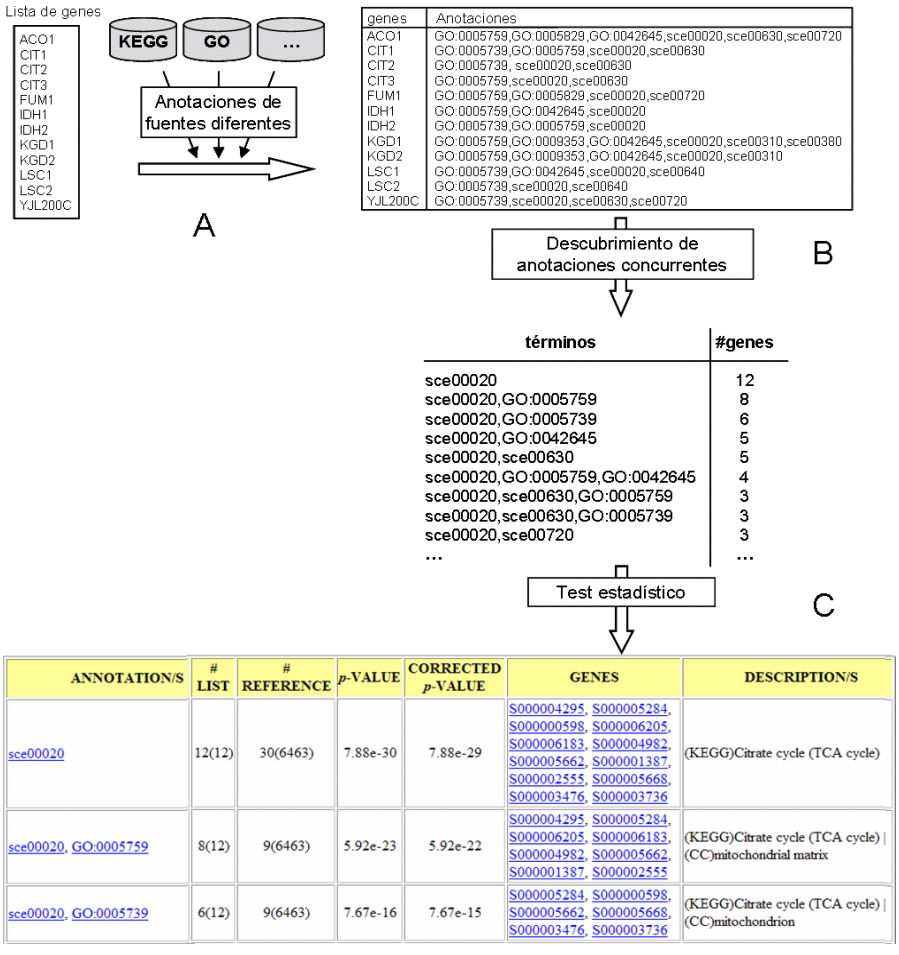


Figura 1.3.1. Esquema de la metodología de GeneCodis. A) Anotaciones procedentes de diversas fuentes son asignadas a los genes de la lista. B) El algoritmo Apriori es usado para encontrar

conjuntos de anotaciones que frecuentemente concurren en la lista de y C) un test estadístico es usado para calcular la significación estadística de las mismas.

1.3.1 Extracción de anotaciones concurrentes

Para extraer conjuntos de anotaciones concurrentes GeneCodis usa la metodología del algoritmo *A priori [3]*, basada en que el proceso empieza determinando el conjunto de anotaciones individuales que aparecen en al menos *x* genes (soporte) de la lista, siendo *x* un umbral definido por el usuario. Este paso genera un conjunto de *k*-anotaciones frecuentes con *k*=1. En la segunda iteración, el conjunto de 1-anotaciones frecuentes es usado para generar las combinaciones de dos elementos, y la base de datos es escaneada de nuevo para contar la frecuencia de cada par de anotaciones. Aquellas que no aparecen en al menos *x* genes de la lista son descartadas. Este procedimiento continúa hasta que no es posible generar más combinaciones entre anotaciones frecuentes. Al final de este proceso todas las combinaciones de anotaciones que aparecen en al menos *x* genes son obtenidas. De esta lista aquellas que contienen información redundante son eliminadas, es decir, las que son subconjunto de un conjunto mayor de anotaciones que aparecen en un número igual o mayor de genes.

1.3.2 Evaluación estadística de las anotaciones

Una vez que todas las combinaciones de anotaciones que aparecen en al menos *x* genes han sido extraídas se calcula la frecuencia de las mismas en la lista de referencia. Por defecto GeneCodis usa como lista de referencia todos los genes del genoma correspondiente, pero el usuario puede usar listas de referencia alternativas, como por ejemplo los genes contenidos en el chip. Seguidamente se aplica un test estadístico para identificar aquellas categorías y combinaciones de categorías que están enriquecidas en la lista de genes con respecto a la lista de referencia. GeneCodis implementa dos test estadísticos para este análisis, la distribución hipergeométrica y el test de χ^2.

Los *p*-valores obtenidos pueden ser ajustados para evitar el problema de las comparaciones múltiples mediante un test basado en permutaciones (Berriz *et al*., 2003; Boyle *et al*., 2004) o mediante el método de FDR de Benjamini y Hochberg (Benjamini y Hochberg, 1995).

1.3.3 La aplicación GeneCodis

Al igual que GeneCodis existen otras herramientas que realizan las siguientes operaciones: primero determinan las anotaciones que están presentes en la lista de genes y su frecuencia es calculada en la lista de genes y en la lista de referencia. Un test estadístico, usualmente basado en la distribución hipergeométrica o binomial, el test de χ^2 o el test exacto de Fisher, es entonces aplicado para calcula un *p*-valor para cada una de estas anotaciones, el cual es normalmente corregido para tener en cuenta el problema de las comparaciones múltiples. El resultado de este análisis consiste en una lista de anotaciones individuales con sus correspondientes *p*-valores. Aquellos términos con *p*-valores por debajo de un cierto umbral son términos que están estadísticamente enriquecidos en la lista de genes con respecto a la lista de referencia y pueden suministrar información muy valiosa para la interpretación de los procesos biológicos asociados al sistema experimental.

Un aspecto importante es que la mayor parte de las herramientas actuales están diseñadas para evaluar anotaciones individuales, y estas no suministran ningún tipo de información sobre las potenciales relaciones entre ellas. Encontrar asociaciones entre anotaciones funcionales basadas en patrones de concurrencia puede suministrar una información más completa para la interpretación de los resultados experimentales. Por ejemplo, un conjunto de genes diferencialmente expresado pueden estar asociados a la activación de un determinado proceso biológico restringido a una determinada localización celular.

GeneCodis es una herramienta web para la extracción y análisis de conjuntos de anotaciones que aparecen frecuentemente juntas en una lista de genes. Este método permite integrar anotaciones procedentes de diversas fuentes (GO, rutas de

KEGG, motivos de InterPro y palabras clave de SwissProt) para extraer combinaciones de las mismas y evaluar su significación estadística en una lista de genes.

2 PLATAFORMA DE DESARROLLO

La aplicación web que tenemos que mejorar, en este caso GeneCodis, está basada en la plataforma de desarrollo Ruby. Como no es un lenguaje de programación muy conocido primero vamos a introducir los aspectos más importantes y que hemos utilizado para la realización del proyecto.

2. 1 Ruby

En nuestro caso no nos toca mucho la arquitectura de Ruby ya que no es un entorno de programación web, pero es la base tanto del framework Rails y del micro-framework Camping, para ello, habrá que conocer ciertas características que nos puede ofrecer Ruby.

2.1.1 Introducción

Ruby [4] es un lenguaje de programación, reflexivo y orientado a objetos (incluyendo las clases y tipos, cosa que en otros lenguajes se describen como primitivas) que se creó hace pocos años (1995) por Yukihiro Matsumoto, es un lenguaje de programación interpretado en una sola pasada y su implementación oficial se distribuye bajo una licencia de software libre.

Una cosa curiosa es tanto su nombre que fue una coña de un amigo del creador referente a Perl (Perla) diciendo que lo llamara Ruby (rubí), y con ese nombre se quedo.

2.1.2 Objetivos

El creador del lenguaje, Yukihiro "Matz" Matsumoto, ha dicho que Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los

principios de una buena interfaz de usuario. Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina:

"A menudo la gente, especialmente los ingenieros en informática, se centran en las máquinas. Ellos piensan, "Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto..." Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos."

Ruby sigue el "principio de la menor sorpresa", lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios experimentados. Matz ha dicho que su principal objetivo era hacer un lenguaje que le divirtiera él mismo, minimizando el trabajo de programación y la posible confusión. Él ha dicho que no ha aplicado el principio de menor sorpresa al diseño de Ruby, pero sin embargo la frase se ha asociado al lenguaje de programación Ruby. La frase en sí misma ha sido fuente de controversia, ya que los no iniciados pueden tomarla como que la características de Ruby intentar ser similares a las características de otros lenguajes conocidos. En mayo de 2005 en una discusión en el grupo de noticias comp.lang.ruby, Matz trató de distanciar Ruby de la mencionada filosofía, explicando que cualquier elección de diseño será sorprendente para alguien, y que él usa un estándar personal de evaluación de la sorpresa. Si ese estándar personal se mantiene consistente habrá pocas sorpresas para aquellos familiarizados con el estándar. Matz lo definió de esta manera en una entrevista:

"Todo el mundo tiene un pasado personal. Alguien puede venir de Python, otro de Perl, y ellos pueden verse sorprendidos por distintos aspectos del lenguaje. Entonces ellos podrían decir 'Estoy sorprendido por esta característica del lenguaje, así que Ruby viola el principio de la menor sorpresa.' Esperad, esperad. El principio de la menor sorpresa no es solo para ti. El principio de la menor sorpresa significa el principio de 'mi' menor sorpresa. Y significa el principio de la menor sorpresa después de que aprendes bien Ruby. Por ejemplo, yo fui un programador de C++ antes de

empezar a diseñar Ruby. Yo programé solamente en C++ durante dos o tres años. Y después de dos años de programar en C++, todavía me sorprendía."

2.1.3 Semántica y características

Todas las funciones en Ruby son métodos, y los métodos que están fuera del ámbito de un objeto pasan a ser objetos de la clase Object (la genérica), que hace que todo método definido así, será visto por todas las clases y objetos de nuestra aplicación.

En resumen las características básicas que posé Ruby son:

- Es un lenguaje orientado a objetos
- Tiene cuatro niveles de ámbito variable: global, clase, instancia y local
- Manejo de excepciones
- Recolección de basura automática (como Java)
- Multiplataforma
- Carga dinámica de DLL
- Introspección, reflexión y meta programación

2. 2 RubyGems

2.2.1 Introducción

RubyGems es el gestor de paquete de Ruby, donde puedes encontrarte tanto programas como librerías. Para hacerlo más fácil este gestor proporciona un formato estándar llamado gema (en inglés gem) para poder instalarlo con facilidad.

La creación de esta herramienta viene porque en estos tiempos que corren está de moda que toda la base de conocimiento de los lenguajes tenga una forma de llegar a ellos de forma sencilla y directa, eso hace que la gente tienda a reutilizar el código y Ruby lo consigue mediante este gestor.

Además de todas estas ventajas RubyGems controla las versiones de tus gemas para poder así comprobar compatibilidades y demás factores que debas tener en cuenta a la hora de realizar tu programa. En este aspecto puede que tengas que para que te funcione la gema A tengas que tener una versión antigua de la gema B, pero eso no es problema porque de todo eso se encarga él evitándonos un tiempo muy grande en buscar versiones que sean compatibles y estar comprobando para cual funciona.

Hay que recordar que RubyGems no viene instalado cuando se instala Ruby, para ello basta con descargarse el paquete para poder utilizarlo. Para nuestro caso haremos uso tanto de instalación de gemas como de plugin que más adelante describiremos.

2.2.2 Instalación

Una vez instalado Ruby tenemos que ejecutar las siguientes instrucciones en nuestra consola:

gem install rubygems-update

Y para que tengamos las últimas versiones hacemos: Update_rubygems

2. 3 Rails

Rails [5] es un framework de Ruby para dar cobertura a aplicaciones web, es muy nuevo, se creó hace apenas 7 años pero ha calado mucho entre los programadores de aplicaciones web por su sencillez.

Podríamos decir que esta plataforma tiene dos cosas que tienen que quedan muy claras a la hora de trabajar con ella, si queremos sacar el mayor provecho de la herramienta, y son:

- DRY - "Don't repeat yourself" (No te repitas): Y como su nombre indica quiere decir que no repitas código que es innecesario, es decir, si trabamos bajo una base de datos, y accedemos a una columna esta no la tenemos que tener declarada, es decir, Rails es listo y busca entre las columnas haber si existe, así nos ahorramos un montón de código que realmente es obvio para nosotros, pero no para otros lenguajes de programación.
- "convención sobre configuración": Es decir, si marcamos un patrón más o menos sencillo, y aprendemos primero a utilizar bien el framework, nos daremos cuenta de que a base de convención nos ahorramos un montón de líneas en configuración. Por ejemplo, si tenemos un controlador que se llama job, si seguimos la convención de Rails la tabla se llamará jobs, así nos evitamos tener que estar declarando. Obviamente Rails no se cierra a los que por algún motivo no quieren seguir estas recomendaciones y soporta que se pueda configurar como se quiera.

Si somos capaces de llevar estás dos sencillas reglas a la práctica nos será muy sencillo realizar aplicaciones web funcionales y con una sencillez y claridad en el código que no se puede hacer en otros lenguajes.

2.3.1 MVC

Rails trabaja bajo el patrón de diseño llamado modelo-vista-controlador (MVC), por ello, vamos hacer un pequeño análisis sobre este patrón para poder llevarlo a cabo de la mejor forma.

Para este caso la importancia de este modelo viene dada por la reutilización de código. En aplicaciones web, lo principal es tener una buena vista para que el usuario se sienta cómodo y por ello, siempre se está innovando sobre cómo hacerlo, y esto nos lleva a plantearnos este modelo para que cuando haya nuevas formas de utilizar las vistas no se vea afectado nuestro núcleo del programa, es decir, qué por poner idiomas a nuestra web no tengamos que modificar toda nuestra aplicación sí al final va a tener la misma funcionalidad interna. Por todo esto Rails ayuda a realizar las aplicaciones mediante este patrón de diseño con la ayuda de módulos que veremos más adelante.

Obviamente el modelo vista-controlador se basa en tres módulos bien diferenciados que como su propio nombre indica son el modelo, la vista y el controlador. Vamos a pasar a detallar cada uno de ellos para más tarde ver como se conectan entre ambos:

- Modelo: En lo referente a aplicaciones web como es nuestro caso el modelo es la interconexión entre el código html y la base de datos. En el modelo se especifican los datos, es decir, se asignan el tipo de todos los objetos con los que trabajamos en las tablas de nuestra base de datos. Es el código más reutilizable y más susceptible a cambios. En el caso de Rails tiene una forma bastante rápida y legible para poder especificar las reglas que tienen que cumplir los objetos con los que trabajamos. Para acceder a los datos usaremos ActiveRecord, que más adelante lo explicaremos.
- Vista: La vista como su nombre indica es la visualización de los datos en nuestra aplicación, detrás de cada una de ellas hay un método. La forma de representarlo habitualmente es .html pero en nuestro caso

Rails trabaja con .html.erb que es lo mismo pero con fragmentos de código Ruby. Es la parte menos reutilizable y más fácil de modificar sin que se vea afectado nuestro núcleo.

- Controlador: En este módulo es el que lleva a cabo las acciones del usuario es decir toda modificación o acción que realice el usuario pasa por el controlador. En el caso de Rails a las acciones se les llama métodos que son invocados desde el navegador para realizar cambios sobre nuestro modelo o mostrar nuevas vistas.

2.3.2 Estructura de una aplicación Rails

En esta imagen podemos apreciar cómo se gestiona Rails internamente:

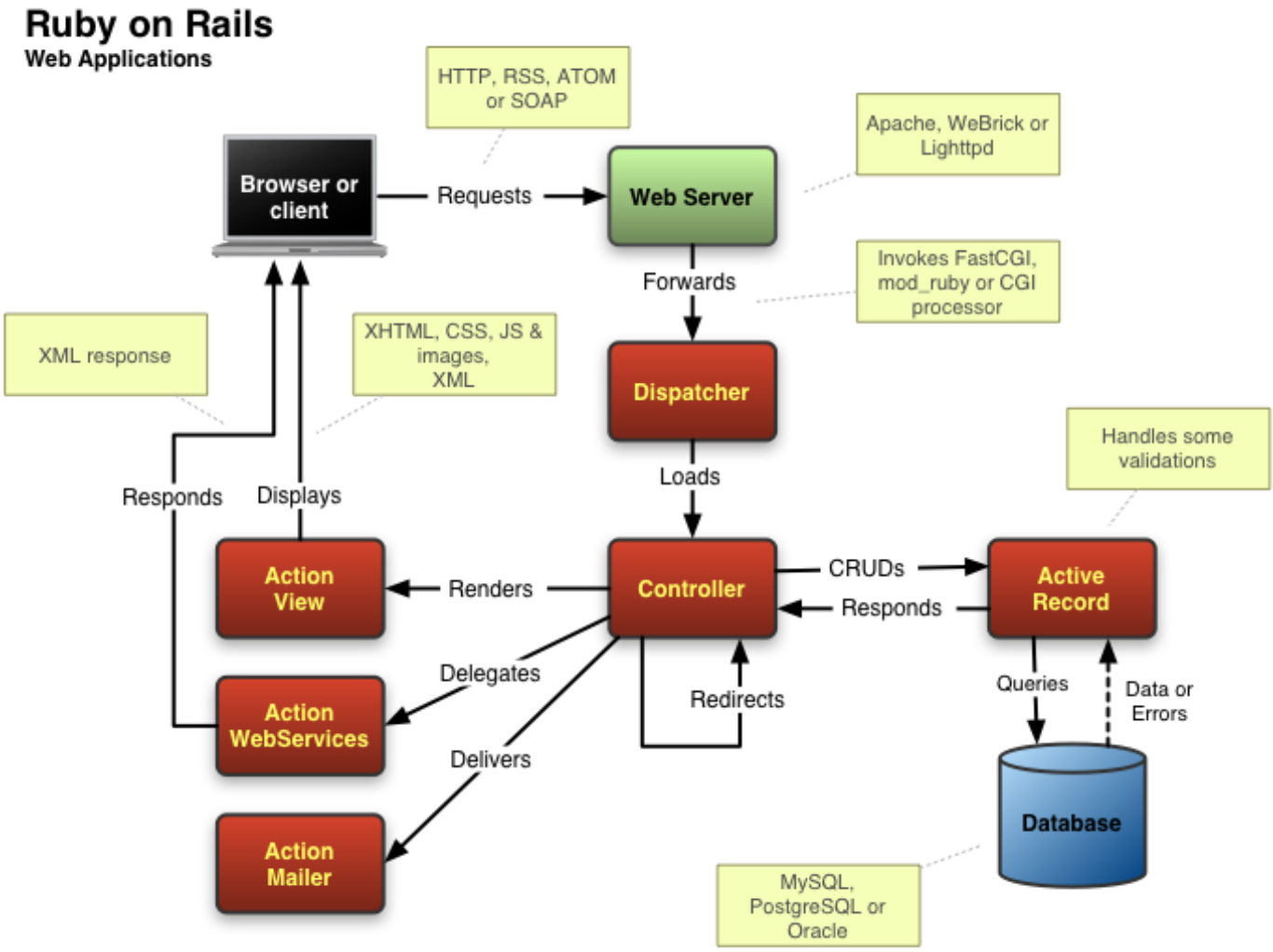


Figura 2.3.2.1 Diagrama de flujo de Rails [Rails]

Ahora después pasaremos a describir algunos de los módulos que aparecen en la imagen:

- ActiveRecord: Es el módulo que se encarga de la conexión entre la aplicación y la base de datos, es decir, consigue crear instancias de objetos que son filas de la base de datos. Digamos que es un envoltorio para facilitar la

implementación. Es decir hacer un new en la clase ActiveRecord será igual a insertar una fila en la base de datos que está implementando nuestro sistema.

- ActionController: Es la base de nuestra aplicación es donde están todas las acciones y lo que hay que mostrar al usuario. Realmente es nuestro controlador en el MVC, por ello no tiene mucho más a resaltar en este apartado.
- ActiveView: Este módulo se encarga de renderizar los datos que nuestro controlador quiere mostrar al usuario. Su uso es mediante vistas que pertenecen a acciones del controlador. Además da la opción de hacer subvistas para la eficiencia de nuestra aplicación. Otra cosa a tener en cuenta es la opción de poner layaouts que son vistas que son visibles para cualquier acción del controlador, o si lo ponemos en el principal será vista durante toda la web, salvo que no se quiera.
- ActionMailer: Como su propio nombre indica es el módulo que se encarga de mandar y recibir mails. Su interfaz es sencilla gracias a una rápida configuración del servidor (que veremos más adelante). Uno de sus puntos fuertes es poder crear vistas para las acciones de mail a mandar, es decir, en nuestro proyecto ponemos la posibilidad de recordar la contraseña, y para este caso el mail que se envía es un template pero cambiando los datos, realmente trabajamos como si fueran vistas web que él se encarga de enviar a su destinatario.
- ActionWebService: Este es el módulo que se encarga de las operaciones de WebService de nuestra aplicación, si es que lo quieres utilizar, lo que consigue es que se pueda llamar a las acciones definidas en nuestros controladores mediante llamadas al sistema y no mediante vistas. Tiene sentido el uso de este módulo cuando se quiere que la aplicación web puede ser accedida de forma directa por otro programa vía internet.
- WebServer: Los servidores para el desarrollo y las pruebas se utilizan: WEBrick o Mongrel. En nuestro caso, vamos a utilizar Mongrel, pero en la práctica cuando se quiere montar el servidor web se pueden utilizar

diferentes servidores, aunque en este caso Rails no nos da una amplia cobertura, o por lo menos sencilla.

2.3.3 Entornos de desarrollo

Rails nos ofrece tres entornos para trabajar:

- Development: Como se explica en su archivo de configuración config -> environments -> development.rb, el código de nuestra aplicación es cargado cada vez que se hace una petición al servidor. Esto baja el rendimiento de respuesta pero es perfecto para el desarrollo de la aplicación ya que no es necesario reiniciar el servidor cada vez que se hace un cambio en nuestro código. Por ello la línea config.cache_classes es igual a false.

- *Production:* (config -> environments -> production.rb) es el adecuado para correr aplicaciones finalizadas. El código no es recargado en cada petición al servidor. Otra diferencia importante es que los reportes o avisos de error en nuestro código no son mostrados en pantalla.
- *Test:* (config -> environments -> test.rb) se utiliza exclusivamente para ejecutar las pruebas y comprobaciones en la aplicación. No se trabajará en este entorno para nada más. Es importante recordar que los datos almecenados en la base de datos de prueba son destruidos por Rails en cada test que se realiza por lo que hay que tener especial cuidado en que la configuración de la sección prueba de nuestro archivo de configuración database.yml no apunte a las bases de desarrollo o producción.

2.3.4 Bases de datos

En las aplicaciones de Rails tienen una gran importancia el uso de bases de datos (BBDD), realmente el mayor potencial de Rails se obtiene en la facilidad de operar con ellas mediante el módulo ActiveRecord ya explicado.

Para que Rails tuviese una gran potencia en el mercado han conseguido que casi cualquier base de datos que existe en el mercado sea soportada bajo la arquitectura de Rails. Vamos a poner las que actualmente tienen ese soporte:

- MySQL
- PostgreSQL
- SQLite
- SQL Server
- IBM DB2
- Informix
- Oracle 8i, 9i, y 10g
- Firebird/Interbase

Además hay que señalar el cambio de tipos entre SQL y Ruby, en esta tabla se pueden ver la igualdad entre los tipos:

Tipo SQL	Ruby Class
int, integer	Fixnum
decimal, numeric	BigDecimal
interval, date	Date
clob, blob, text	String
float, double	Float
char, varchar, string	String
datetime, time	Time
Boolean	see text

2.3.5 ActiveRecord

El módulo ActiveRecord es el que se encarga de efectuar las operaciones de consulta, modificación y borrado sobre nuestra tabla de la base de datos.

2.3.5.1 Configuración

La forma de configurar nuestra base de datos es en el archivo de configuración database.yml. La forma de configurar MySql y Sqlite3 que son las dos bases de datos con las que trabajamos se hace de la siguiente forma:

MySql:

- Entorno: Es la configuración para la entorno que queremos configurar.
  - Adapter: El adaptador que vamos a utilizar para llamar a la BBDD, esto hace que haya que instalar una gema para que funcione.
  - Encoding: La codificación en la que trabajamos.
  - Reconnect: Booleano para decir si se reconecta o no.
  - Database: Nombre de nuestra Schema.
  - Pool: Tamaño del pool (buffer de BBDD).
  - Username, password: Identificadores.
  - Host: Donde se encuentra.
  - Port: Puerto por el que se accede.

Ejemplo de MySQL (configuración de nuestro proyecto):

development

adapter: mysql

encoding: utf8

reconnect: false

database: mevp_development

pool: 5

username: root

password: admin

host: localhost

Sqlite3: Es igual que el mysql pero sólo hay que configurar dos parámetros, como en el caso del anterior hay que instalar el adaptador mediante gemas:

- Adapter: sqlite3
- Database: mevp_development

2.3.5.2 Accesos a la BBDD

Vamos a describir las formas de acceder del módulo con MySQL, en todos los casos las llamadas se realizan mediante la llamada al nombre del controlador seguido de punto con las funciones que describimos a continuación:

- Create: La función que se utiliza es new, hay muchas formas de crear una nueva fila, en nuestra aplicación nos ayudamos de Rails para simplemente tener que hacer:

  Job.new(params[:jobs])

  Esto lo podemos hacer por tener definido en el modelo los parámetros que tiene nuestra tabla.

- Select: La función a la que hay que llamar en este caso es find, vamos a describir ciertos parámetros que nos han servido de ayuda para realizar el proyecto, pero si se quiere saber cómo aprovechar al máximo esta función leer los libros de la bibliografía para buscar la información. Para realizar una consulta normal será tan fácil como:

  Job.find(:all) o Job.find(:first)

  Si luego queremos buscar por id:

  Job.find(21)

  Para poner condiciones utilizaremos el parámetro conditions que nos da el módulo, y habrá simplemente que especificar el nombre de la columna con su condición:

User.find(:all,:conditions => ["login LIKE 'admin'"])

En este caso la condición no es dinámica pero podemos hacerla de la siguiente forma:

Job.find(:all,:conditions=>["login LIKE ?",current_user.login]

Hay otras formas de utilizarlo mediante params[], metes los parámetros y donde antes poníamos interrogación ahora ponemos el parámetro con ':'.

Para el caso que queramos ordenar las búsquedas tenemos un parámetro llamado order que lo que hace es ordenarlo por lo que queramos:

Job.find(:all,:order=> "login")

También podemos poner un límite a la búsqueda:

Job.find(:all,limit=>10)

Por último decir que también se pueden hacer offset, joins, seleccionar columnas, readonly, group y lock. Si al final de todo esto no tenemos suficiente para poder realizar nuestra consulta Rails tiene una solución y es directamente poder realizar una consulta SQL de la siguiente forma:

Job.find_by_sql("Select…")

Decir que no es bueno usar esta función salvo necesidad extrema ya que no estamos utilizando todo el potencial que nos ofrece ActiveRecord sobre nuestra base de datos.

- Count: Esta cláusula lo que hace es contar el número de filas que tenemos en la base de datos con ciertas condiciones, únicamente habrá que realizar una llamada para poder realizar la función. Los parámetros son los mismos que para el select (find):

Job.count (:conditions=>"",:group=>…)

- Update: Como Rails trabaja con objetos cada fila de nuestra base de datos será un objeto, valdrá simplemente con llamar a update desde el objeto para realizar la modificación con las columnas a modificar. Otra forma de poder hacerlo es mediante el mandato save que se llama de igual forma que update pero hay que haberlo modificado antes mediante operaciones básicas.

- Delete: Simplemente como ocurre con el save como trabajamos con objetos habrá que llamar a la función destroy para borrar las filas que tengamos en el objeto que puede ser un array. Otra forma es lanzar como hace el update que en este caso se llama delete y hace una llamada sql para borrar filas con condiciones.

2.3.6 ActionMailer

El ActionMailer [6] es el módulo que se encarga del servidor de correos, se puede tanto enviar como recibir, para nuestro proyecto vamos a explicar únicamente como configurar y enviar correos si se quiere consultar mayor información acuda a la bibliografía.

2.3.6.1 Configuración

Vamos a mostrar dos formas de configurar el correo, una para cuando tengamos un servidor smtp y otra para utilizar el servidor smtp que nos ofrece gmail mediante una gema.

- Servidor Smtp: Teniendo un servidor smtp simplemente debemos configurar en el archivo environment el Mailer:

```
ActionMailer::Base.delivery_method = :smtp
ActionMailer::Base.server_settings = {
  :address => "smtp.tutorialspoint.com",
  :port => 25,
  :domain => "tutorialspoint.com",
  :authentication => :login,
```

```
  :user_name => "username",
  :password => "password",
}
```

- Gmail [gmail]: Gmail, al igual que Google Apps for Domains, se puede usar como servidor de correo saliente (SMTP) para nuestras aplicaciones Rails. Lo único es que como requiere autenticación TLS, no vale con usar simplemente "smtp.gmail.com" como servidor de correo saliente. El plugin action_mailer_tls resuelve el problema rápidamente:

  script/plugin install http://code.openrain.com/rails/action_mailer_tls/

  Una vez instalado, en la carpeta /vendor/plugins/action_mailer_tls/ sample encontraremos dos ficheros. Copiamos smtp_gmail.rb dentro de tu carpeta /config/initializers, y copiamos mailer.yml.sample a la carpeta /config, renombrándolo a mailer.yml. Finalmente, editamos el fichero para usar el user_name y password de la cuenta de correo de Gmail que vamos a utilizar y listo.

2.3.6.2 Generar un mail

Lo primero que tendremos que hacer es el método que se llamara, será de la clase Mailer:

```
def notification(user, subjects, message)
  subject    subjects
  recipients  "#{user.email}"
  from       'no-reply@yourdomain.com'
  sent_on    Time.now
  body       :message => message

end
```

Luego en la vista generamos el mail, la vista se tiene que llamar de la misma forma que la acción en este caso notification:

<%= @message %>

Don't reply. Thanks

2.3.7 Rake

El rake [7] de Rails es parecido al make de Unix, por eso se llama rake es la unión de Ruby con make (R-ake). Lo que hace rails es definir unas operaciones que son interesantes para nuestra trabajar con nuestra aplicación. Describiremos aquí las que nosotros hemos utilizado, si se quiere saber más consultar el apéndice rake [rake]:

- **rake db:migrate** - Migracion de la base de datos a traves del script db-migrate, para espicificar la versión añada VERSION =X.
- **rake db:sessions:clear** - Borra las sesiones
- **rake db:sessions:create** - Crea una table para usar con CGI::Session::ActiveRecordStore
- **rake db:structure:dump** - Descarga la base de datos con estructura de SQL.
- **rake db:test:clone** - Recrea la base de datos de prueba del esquema de la base de
- **rake doc:rails** - Crea lo archivos HTML de Rails.
- **rake log:clear** - Borra todos los datos en /log.
- **rake rails:update** - Actualiza configuraciones, scripts y los javascripts de Rails.
- **rake rails:update:configs** - Actualiza config/boot.rb de la instalación actual de rails.
- **rake rails:update:javascripts** - Actualiza los javascripts de la aplicación.
- **rake rails:update:scripts** - Agrega nuevos scripts al folder scripts/.
- **rake stats** - Reporta las estadísticas de código de la aplicación.
- **rake test** - Prueba todas las unidades y funciones.

- **rake tmp:cache:clear** - Borra todos los archivos y directorios en tmp/cache.
- **rake tmp:clear** - Borra sessiones, cache y archivos socket de folder /tmp.

2.3.8 Estructura de archivos

En un proyecto de Rails nos podemos encontrar las siguientes carpetas:

- /app: Donde están en carpetas separadas los controladores, modelos, vistas y helpers
- /components: Miniaplicaciones que pueden utilizar controladores, modelos y vistas de forma conjunta.
- /config: Configuración de base de datos, de rutas y ajustes de entorno.
- /db: Ficheros con los esquema de base de datos y los ficheros de migración Rails
- /doc: Documentación de la aplicación
- /lib: Código de la aplicación que no pertenece a controladores, modelos o helpers. Por ejemplo, unas librerías para la generación de .xls
- /log: Ficheros de log de acceso y de error de la aplicación.
- /public: Es donde se almacenan los archivos extras de nuestra aplicación: CSS, Javascripts, imágenes y los ficheros generados que luego enviaremos al navegador.
- /script: Scripts de generación de código, herramientas de depuración y utilidades de depuración
- /test: Ficheros para testear la aplicación, test de unidad, de integración de código, fixtures. . .
- /tmp: Directorio dónde se encuentran ficheros de sesión, caché, sockets.
- /vendor: Lugar dónde son instalados los plugins de la aplicación

3.9 Instalación

Rails es una gema de RubyGems, lo que hace que su instalación sea tan fácil como llamar a este servidor de gemas, lo único a tener en cuenta es que para que Rails funcione, como hemos visto hasta ahora, necesita de varias gemas que no están por defecto instaladas en Ruby, por tanto a la hora de llamar al servidor hay que hacer:

```
gem install rails --include-dependencies
```

Hay que recordar que Rails es un framework que se conecta a bases de datos por tanto habrá que instalar también dichas gemas. Para el caso de MySql sería:

```
gem install mysql
```

Si se trabaja sobre el sistema operativo Windows hay que tener en cuenta que hay que copiar en las librerías de Ruby la .dll del MySql para que funcione correctamente.

2.4 Camping

Figura X. Símbolo de Camping

Camping [8] es un micro-framework de Ruby muy similar a Rails, pero con unas pequeñas diferencias que hay que tener en cuenta a la hora de trabajar con él.

Los módulos en los que se basa Camping son similares a los de Rails, pero uno de los cambios es que no hay una jerarquía de vistas y controladores como en Rails, aquí en camping sólo hay por cada módulo un archivo y en él se encuentran todos los módulos declarados, es decir, en controllers estarán declarados todos los controladores de nuestra aplicación. Esto hace que tengas que tener un pequeño control para que los archivos no sean imposibles de ver, para luego poder buscar los módulos con facilidad. Hay tres tipos básicos:

- Camping::Models: El modelo usa como no, en su estructura interna la gema ActiveRecord, y como venimos diciendo es la base de nuestra aplicación y donde se declaran todos nuestros datos.
- Camping::Controllers: Es nuestro controlador, el que efectúa los acciones del usuario, en este caso es distinto a Rails. Hay que tener en cuenta tres aspectos:
    - Se declara la url a la que pertenece el controlador y no es automática la generación del link, realmente a nosotros nos da lo mismo porque siempre en Rails hemos estado cambiando las url's por claridad en nuestra aplicación.
    - La forma de interactuar entre la vista y el controlador es utilizando siempre los métodos get y post de html para poder llevar a cabo un control de la página.
    - Siempre hay que renderizar si se quiere recargar la página.

- Camping::Views: Son los html de Ruby, son las vistas de nuestra aplicación y se trabaja igual que en Rails.

Hay más módulos que no hemos descrito pero no son relevantes para nosotros y se puede consultar la referencia para obtener más información.

Sobre el tema de base de datos como Camping trabaja con las mismas gems de Rails las active no hay problemas al conectar con cualquier tipo de base de datos, además si se quiere migrar de una aplicación Rails a Camping si se han cumplido las conveciones de acceso y uso de la base de datos el cambio no supondrá ningún tipo de modificación para el programador.

2.4.1 Instalación

Como casi todo en Ruby es tan sencillo como ejecutar las siguientes instrucciones para poder utilizar la gema en nuestra aplicación:

gem install camping

o bien

gem install camping —source code.whytheluckystiff.net

Para luego poder utilizar el módulo habrá que añadir un require al principio de los archivos para que no nos devuelva error el servidor.

2.5 HTML básico

2.5.1 Introducción

El lenguaje HTML (*HyperText Markup Language) [9]* es un lenguaje para marcado de hipertexto. Se trata de un lenguaje para estructurar documentos a partir de texto en World Wide Web, que es un medio de comunicación de texto, gráficos y otros objetos multimedia a través de Internet, es decir, la web es un sistema de hipertexto que utiliza Internet como su mecanismo de transporte o desde otro punto de vista, una forma gráfica de explorar Internet. Este lenguaje se basa en etiquetas (instrucciones que le dicen al texto como deben mostrarse) y atributos (parámetros que dan valor al etiqueta).

Como ya se ha dicho, este lenguaje estructura documentos. La mayoría de los documentos tienen estructuras comunes (como títulos, párrafos, listas, entre muchos otros) que van a ser definidas por este lenguaje mediante etiquetas. Todo aquello que no sea una etiqueta es parte del documento mismo.

Este lenguaje no describe la apariencia del diseño de un documento sino que ofrece a cada plataforma que le de formato según su capacidad y la de su navegador.

Cabe destacar que HTML tiene dos ventajas que lo hacen prácticamente imprescindibles a la hora de diseñar una presentación web: su compatibilidad y su facilidad de aprendizaje debido al reducido número de etiquetas que usa.

2.5.2 Semántica

Básicamente, los documentos escritos en HTML constan del texto mismo del documento y las etiquetas que pueden llevar atributos. Esto llevado a la práctica, vendría a ser:

<etiqueta> texto afectado </etiqueta>

La etiqueta del principio activa la orden y la última (que será la del principio precedida del signo /) la desactiva. No todas las etiquetas tienen principio y final.

Atributos y valores

En las etiquetas pueden darse valor a una serie de atributos, de esta manera:

<elemento atributo='valor' atributo='valor'> ... </elemento>

- Los atributos deben estar en la etiqueta inicial, nunca en la final.
- El orden de los atributos dentro de la etiqueta es indiferente.
- Deben separarse con un espacio en blanco del nombre del elemento y de otros atributos.
- Atributo y valor deben estar unidos por el signo: =
- Los valores deben estar encerrados con comillas, dobles o simples.

2.5.3 Estructura del HTML

Las partes más importantes son:

- DOCTYPE**:** Define el tipo de documento. Este elemento le indica al navegador la versión y tipo de HTML empleado en el documento. De esta forma, el navegador usará el modelo de renderización adecuado al tipo de documento. De no poner el doctype, el navegador interpretará el código html escrito tal y como le parezca mejor. Por lo que se obtendrán resultados muy distintos de esta forma, incluso entre dos versiones de un mismo navegador.
- HTML: Delimita el documento HTML, indicando al navegador el comienzo y fin de la página html. Sus etiquetas son: <html> (siempre al comienzo después del doctype) y </html> (siempre al terminar el documento), aunque ambas son opcionales.
- HEAD: Delimita la cabecera. Sus etiquetas son: <head> y </head>, ambas son opcionales. La cabecera es la sección apropiada para

incluir información sobre el documento, la mayoría de la cual no será mostrada a los lectores. Para incluir esta información tenemos diversos elementos, como:

- TITLE: Indica el título del documento. Sus etiquetas son: <title> y </title> (obligatorias).
- BODY: Delimita el cuerpo del documento, que es la parte a mostrar a los lectores. Sus etiquetas son: <body>; y </body> (el cierre se encontrará antes de </html>). La etiqueta <body> puede tener los siguientes atributos:

  - text="..." color del texto
  - link="..." color de enlaces no visitados
  - vlink="..." color de enlaces visitados
  - alink="..." color del link activo
  - bgcolor="..." color del fondo
  - background="..." Imagen de fondo

Vamos a mostrar el tipo ejemplo de hola mundo como se haría en HTML:

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <title>Mi primera página</title>
    </head>
    <body>
        <p>Hola mundo</p>
    </body>
</html>
```

Las tablas son una herramienta muy útil y muy potente para mostrar información de una manera estructurada y para nuestro caso son esenciales. Las

tablas no son una herramienta para maquetar o dar formato a los documentos sino para tabular datos. Los elementos que tiene para crearlas son:

table (table = tabla) Es el elemento que define y delimita la tabla. Sus etiquetas son: <**table**></**table**> (ambas obligatorias)

Sus atributos principales son:

- *width* - anchura de la tabla (valor en pixeles o en porcentaje)
- *border* - grosor del borde de la tabla (valor en pixeles)
- *cellspacing* - espacio entre celdas (valor en pixeles)
- *cellpadding* - espacio entre el contenido y los bordes de la celda (valor en pixeles)

tr (table row = fila de tabla, renglón de tabla) Es el elemento que define y delimita las filas de la tabla. Sus etiquetas son: <**tr**></**tr**>.

td (table data = datos de tabla) Es el elemento con el que crearemos las celdas de la tabla. Sus etiquetas son: <**td**></**td**>.

Sus atributos principales son:

- *align* --alineación horizontal.
- *valign* --alineación vertical.
- *colspan*-- número de columnas ocupadas por la celda.
- *rowspan*-- número de filas ocupadas por la celda.

2.5.4 CSS

CSS [10],Hojas de Estilo en Cascada (Cascading Style Sheets), es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos.

CSS se utiliza para dar estilo a documento HTML y XML, separando el contenido de la presentación. Los *Estilos* definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento.

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

h1 {color: red;}

- h1 es el selector
- {color: red;} es la declaración

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto. En el ejemplo anterior, el selector h1 indica que todos los elementos h1 se verán afectados por la declaración donde se establece que la propiedad color va a tener el valor red (rojo) para todos los elementos h1 del documento o documentos que estén vinculados a esa hoja de estilos.

Las tres formas más conocidas de dar estilo a un documento son las siguientes:

Utilizando una hoja de estilo externa que estará vinculada a un documento a través del elemento <link>, el cual debe ir situado en la sección <head>.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
 <head>
  <title>Título</title>
  <link rel="stylesheet" type="text/css"
      href="http://www.w3.org/css/officeFloats.css" />
 </head>
 <body>
 .
 .
 .
 .
 </body>
</html>
```

Utilizando el elemento <style>, en el interior del documento al que se le quiere dar estilo, y que generalmente se situaría en la sección <head>. De esta forma los estilos serán reconocidos antes de que la página se cargue por completo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
 <head>
  <title>hoja de estilo interna</title>
  <style type="text/css">

   body {
    padding-left: 11em;
```

```
    font-family: Georgia, "Times New Roman", serif;
    color: red;
    background-color: #d8da3d;
   }

   h1 {
    font-family: Helvetica, Geneva, Arial, sans-serif;
   }

  </style>
 </head>
 <body>
  <h1>Aquí se aplicará el estilo de letra para el Título</h1>
 </body>
</html>
```

Utilizando estilos directamente sobre aquellos elementos que lo permiten a través del atributo <style> dentro de <body>. Pero este tipo de definición del estilo pierde las ventajas que ofrecen las hojas de estilo al mezclarse el contenido con la presentación.

Algunas normas básicas a la hora de crear una CSS son las siguientes:

En el siguiente ejemplo, h1{color: red;}, el *selector*, <h1>, le dice al navegador la parte del documento que se verá afectada por esa regla. Los selectores pueden aparecer individualmente o agrupados, separándolos con comas:

```
h1, h2, h3 {
 color: red;
}
```

lo que es lo mismo

```
h1 {color: red;}
h2 {color: red;}
h3 {color: red;}
```

La propiedad, que en este caso sería color, especifica qué aspecto se va a cambiar. En este ejemplo la propiedad cambiada será el color. Las propiedades que se desean modificar en una CSS para un mismo selector pueden agruparse, pero será necesario separar cada una de ellas con un punto y coma.

```
p {text-align:center;color:red}
```

Normalmente se describe una propiedad por línea, de la siguiente manera:

```
h1 {
padding-left: 11em;
font-family: Georgia, "Times New Roman",Times, serif;
color: red;
background-color: #d8da3d;
}
```

El *valor*, representado a la derecha de los dos puntos (:), establece el valor de la propiedad. Es importante recordar que si el valor está formado por más de una palabra, hay que ponerlo entre comillas.

```
p {font-family: "sans serif";}
```

3 MOTIVACIÓN Y OBJETIVOS

GeneCodis cuenta en la actualidad con una media de 500 trabajos reales provenientes de distintas partes del mundo, en especial de Europa, USA y Japón. Así mismo, el número de procesos realizados por el mismo usuario es también alto y la tendencia es aumentar. Este número elevado de procesos por usuario hace que la gestión de la información sea imprecisa poco fiable y prácticamente imposible de gestionar de una manera organizada ya que la manera de notificación existente en la actualidad está basada en el correo electrónico o en el almacenamiento manual de las URL con los resultados.

Por lo tanto, el objetivo principal de este proyecto es proporcionar a la aplicación GeneCodis con un sistema profesional de gestión de usuarios y de procesos. Nuestro objetivo es conseguir que ese trabajo de almacenamiento lo realice GeneCodis mediante una BBDD y una gestión de usuarios, para ofrecer además no sólo la lista de trabajos, sino con posibilidad de buscarlos y ordenarlos una vez terminados, pudiendo acceder a los mismos de una forma directa y sencilla. Otra de las posibilidades que permitiremos en este nuevo sistema es la exportación de los datos y resultados a diferentes archivos para que así el usuario pueda trabajar con ello independientemente de la plataforma que utilice.

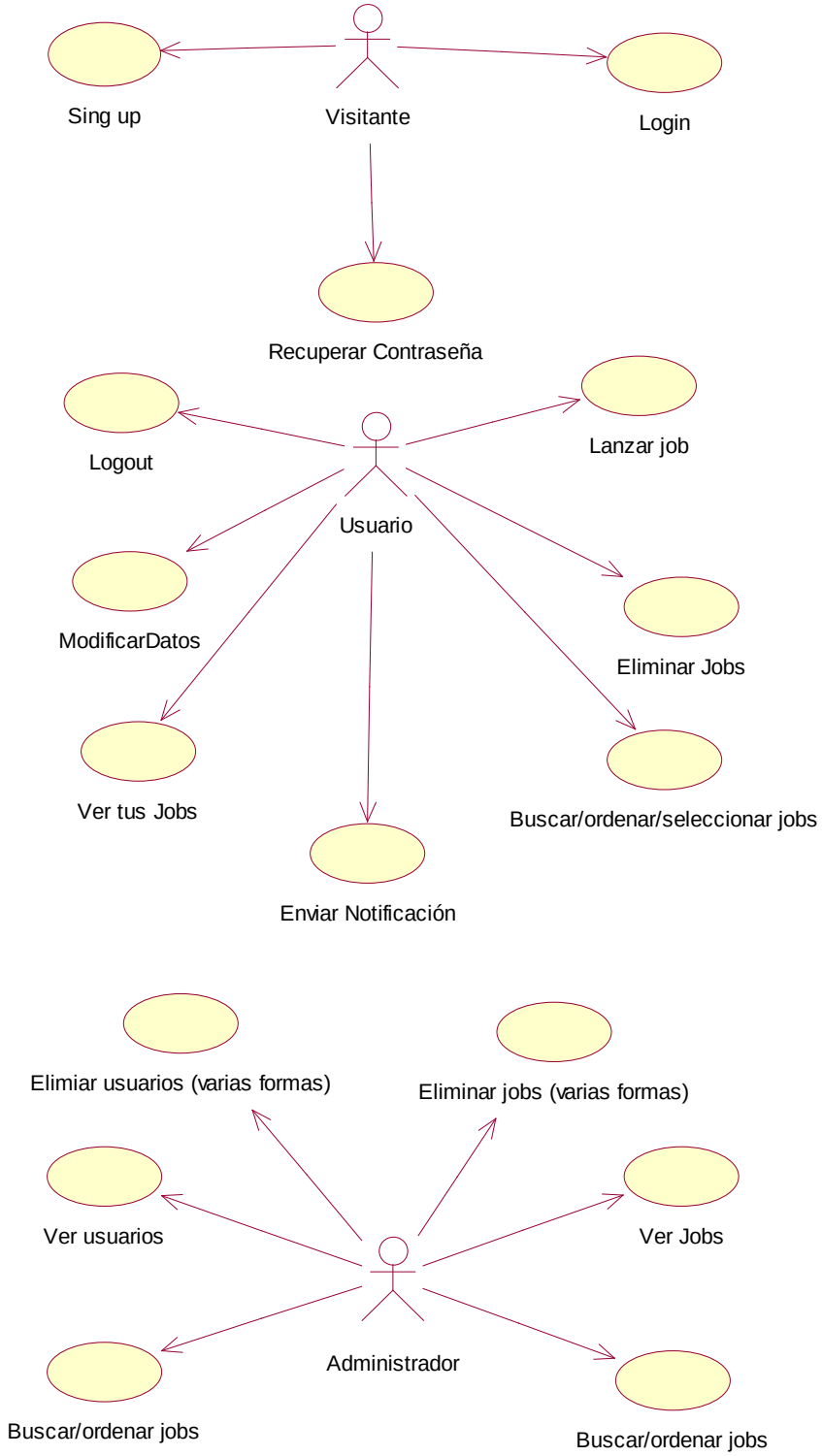
Así mismo el sistema estará basado en servicios webs (WebService) debido a que el mayor soporte de los trabajos realizados se realizan mediante este método.

4 DISEÑO

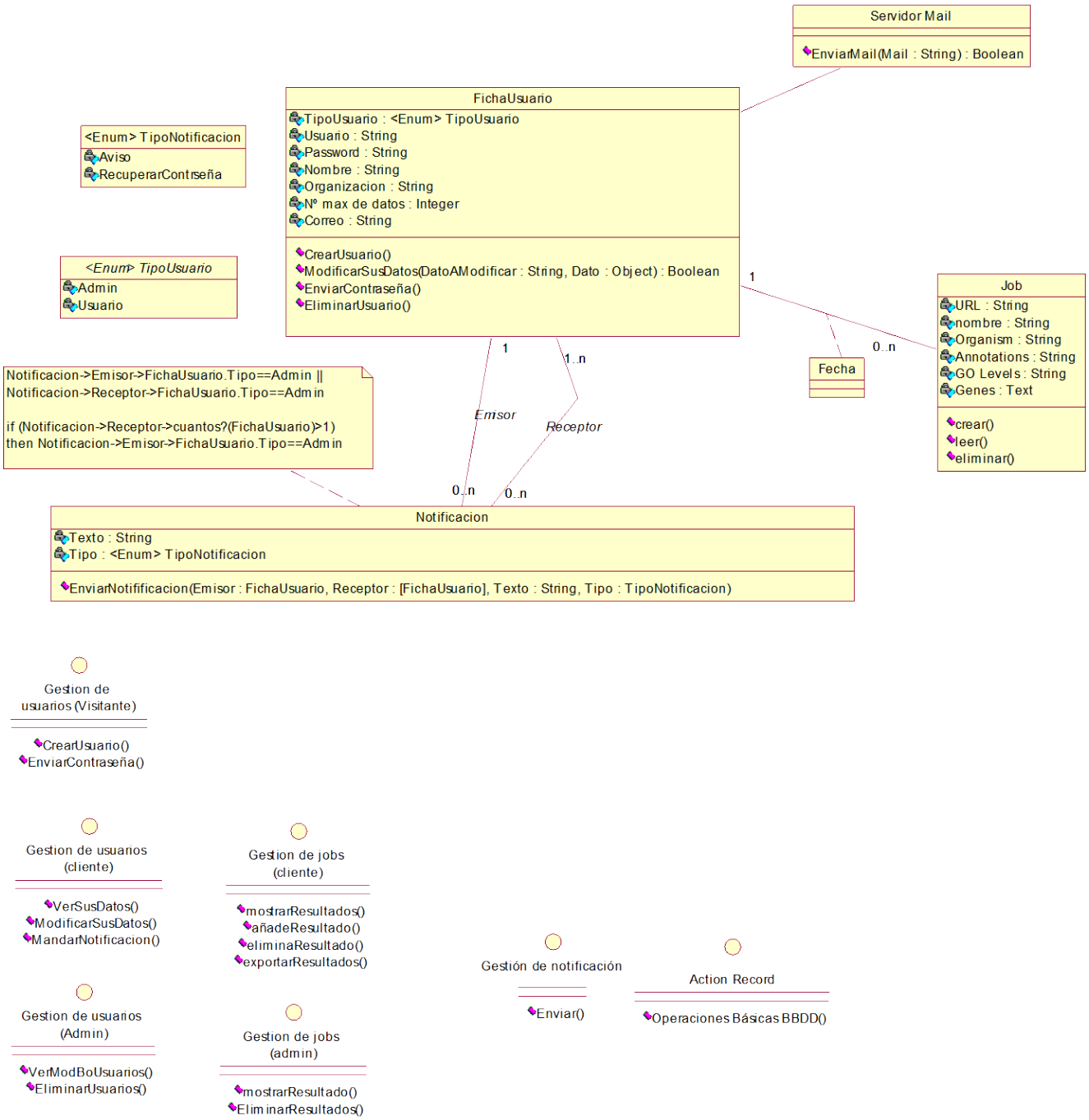
Para poder entender el proyecto desde un principio hicimos diagramas UML para que desde un principio quedara claro lo que teníamos que hacer. Para ello hemos hecho el diagrama de casos de usos, el de clases, el de componentes (basándonos en nuestro patrón de diseño MVC) y por último el entidad-relación de nuestra base de datos que no lo ponemos porque sería igual que el diagrama de clases dejando solo la ficha de usuarios y los jobs.

No hemos creído conveniente hacer más diagramas, ya que no creemos que nuestra implementación sea difícil de entender, evitándonos así los diagramas de actividades y de secuencias.

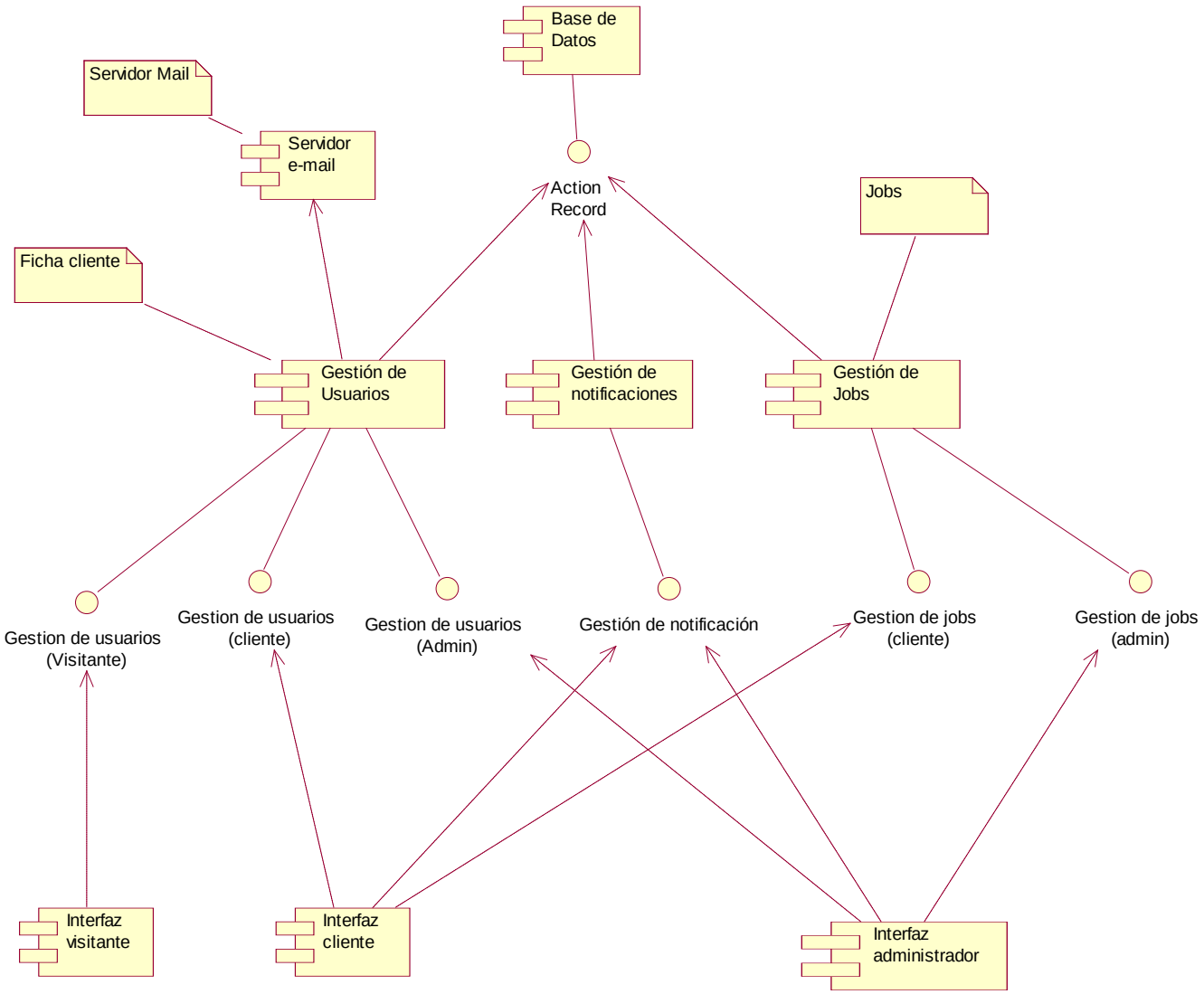
4.1 Diagrama de casos de uso:



4.2 Diagrama de clases



4.3 Diagrama de componentes / MVC



5 CONCLUSIONES Y FUTURO

Con esta nueva implementación GeneCodis se vuelve mucho mejor, ahora los usuarios que quieran podrán tener los datos ordenados y disponibles de una forma fácil y sencilla. Este planteamiento de la aplicación web hace que cada vez sea mejor y en un futuro pueda ser una aplicación con una carga mayor de la que tiene ahora mismo.

El futuro de esta aplicación es prometedor porque cada año que pasa tiene nuevas funcionalidades y algoritmos que la convierten en una referencia en el mundo de la bioinformática.

Porque no mencionar el futuro de la plataforma con la que trabaja, Ruby, que al ser un lenguaje nuevo todavía no tiene grandes logros pero que poco a poco se va abriendo un hueco en el mundo de la programación y ojalá a los programadores les empiece a gustar, además, Rails que tiene poco tiempo de vida, si que está empezando a cuajar entre grandes programadores que buscan en Rails una forma sencilla de implementar su web.

6 AGRADECIMIENTOS

Queremos mencionar a las siguientes personas que nos han ayudado a realizar el proyecto:

- Alberto Pascual Montano, director del proyecto. Por habernos elegido para realizar el proyecto y haber estado pendiente de todas nuestras dudas.
- Rubén Nogales. Por ayudarnos con Ruby on Rails y toda la parte de implementación de GeneCodis
- Y porque no a todos los amigos y familiares que nos han mostrado su apoyo para realizar el proyecto.

7 REFERENCIAS

[1] - http://www.solociencia.com/biologia/bioinformatica-concepto.htm

[2] – Tesis de Pedro Carmona Sáez (Mirar bibliografía básica)

[3] – http://educnet.decom-uv.cl/educnet/uploads/inzulza_bello.pdf?nombre=p373/inzulza_bello.pdf

[4] - http://es.wikipedia.org/wiki/Wikipedia:Portada

[5] - http://blog.rallat.com/

[6] - http://www.jaimeiniesta.com

[7] - http://www.ticoonrails.com/blog/post/opciones-de-las-tareas-rake

[8] - http://camping.rubyforge.org

[9] - www.webestilo.com/html/

www.programacion.com/html/

www.masadelante.com/faqs/www

http://es.wikibooks.org/wiki/HTML

[10] - http://www.w3c.es/divulgacion/guiasbreves/HojasEstilo

8 BIBLIOGRAFÍA

Bibliografía básica

- Sam Ruby, Dave Thomas and David Heinemeier Hansson. *Agile Web Development with Rails (3rd edition)*
- Pedro Carmona Sáez. Tesis doctoral: *Análisis y extracción de información biológica a partir de datos de expresión génica obtenidos mediante microchips de ADN*
- Nogales-Cadenas R, Carmona-Saez P, Vazquez M, Vicente C, Yang X, Tirado F, Carazo JM, Pascual-Montano. *GeneCodis: interpreting gene lists through enrichment analysis and integration of diverse biological information. Nucleic Acids Research 2009; doi: 10.1093/nar/gkp416.*
- Carmona-Saez P, Chagoyen M, Tirado F, Carazo JM, Pascual-Montano. *GENECODIS: A web-based tool for finding significant concurrent annotations in gene lists. Genome Biology 2007 8(1):R3*
- *Ryan Bates. http://railscasts.com/*

Bibliografía complementaria

- Burd, Barry A. *Ruby on Rails for dummies*
- T.K. Attwood y D.J. Parry- Smith. *Introducción a la bioinformática*
- Carson, Lucas. *Curso de Ruby*
- Vohra, Deepak. *Ruby on Rails for PHP and Java developers*
- Juan José Vidal Agustín. *http://www.um.es/atica/documentos/PRErubyONrails.pdf*
- *http://www.ruby-forum.com/*
- *http://www.tutorialspoint.com/ruby-on-rails/*

9 APÉNDICES

9.1 Apéndice A .Ruby

Object

- Obj#class -> class
- Obj#freeze -> object
- Obj#frozen? -> true or false
- Obj#inspect -> string
- Obj#is_a? (*class*) -> true or false
- Obj#methods -> array
- Obj#respond_to? (*sym*) -> true or false
- Obj#to_s -> string

String

- Str#[*num, num/range/regx*] -> str
- Str#capitalize! -> string
- Str#center (*int* [,*str*]) -> str
- Str#chomp! ([*str*]) -> str
- Str#count -> integer
- Str#delete! ([*string*]) -> string
- Str#downcase! -> string
- Str#each ([*str*]) do |str| ... end
- Str#each_line do |line| ... end
- Str#gsub! (*rgx*) do |match| ... end
- Str#include? (*str*) -> true / false
- Str#index (*str/reg* [,*off*]) -> int
- Str#insert (*int, string*) -> string
- Str#length -> integer
- Str#ljust (*int* [,*padstr*]) -> str
- Str#rindex (*str/reg* [,*off*]) -> int
- Str#rjust (*int* [,*padstr*]) -> str
- Str#scan (*rgx*) do |match| ... end
- Str#split (*string*) -> array
- Str#strip! -> string
- Str#sub! (*rgx*) do |match| ... end
- Str#swapcase! -> string
- Str#to_sym -> symbol
- Str#tr! (*string, string*) -> string
- Str#upcase! -> string

Kernel

- block_given?
- eval (*str* [,*binding*])
- raise (*exception* [,*string*])
- fork do ... end => fixnum or nil
- proc do ... end => proc
- print (*obj*)
- warn (*msg*)

Array

- Array::new (*int* [,*obj*]) -> array
- Array#clear
- Array#map! do |x| ... end
- Array#delete (*value*) -> obj or nil
- Array#delete_at (*index*)-> obj or n
- Array#delete_if do |x| ... end
- Array#each do |x| ... end
- Array#flatten! -> array
- Array#include? (*value*) -> t or f
- Array#insert (*idx, obj...*)-> array
- Array#join ([*string*]) -> string
- Array#length -> integer
- Array#pop -> obj or nil
- Array#push (*obj...*) -> array

Hash

- Hash#clear
- Hash#delete (*key*) -> obj or nil
- Hash#delete_if do |*k, v*| ... end
- Hash#each do |*k, v*| ... end
- Hash#has_key? (*k*) -> true or false
- Hash#has_value? (*v*) -> t or f
- Hash#index (*value*) -> key
- Hash#keys -> array
- Hash#length -> integer
- Hash#select do |*k, v*| ... end -> array
- Hash#values -> array

Test::Unit

- assert (*boolean* [,*msg*])
- assert_block (*message*) do ... end
- assert_equal (*expected, actual* [,*msg*])
- assert_in_delta (*exp, act, dlt* [,*message*])
- assert_kind_of (*klass, object* [,*msg*])
- assert_match (*pattern, string* [,*msg*])
- assert_nil (*object* [,*msg*])
- assert_no_match (*pattern, string* [,*msg*])
- assert_not_equal (*expected, actual* [,*msg*])
- assert_not_nil (*object* [,*msg*])
- assert_not_same (*expected, actual* [,*msg*])
- assert_respond_to(*obj, method* [,*msg*])
- assert_same (*expected, actual* [,*msg*])

File

- File#new (*path, modestring*)-> file
- File#new (*path, modestring*) do |file| ... end
- File#open (*path, modestring*) do |file| ... end
- File#exist? (*path*) -> t or f
- File#basename (*path* [,*suffix*]) -> string
- File#delete (*path, ...*)
- File#rename (*old, new*)
- File#size (*path*) -> integer

r	Read-only, from beginning
r+	Read-write, from beginning
w	Write-only, trunc. / new
w+	Read-write, trunc. / new
a	Write-only, from end / new
a+	Read-write, from end / new
b	Binary (Windows only)

Dir

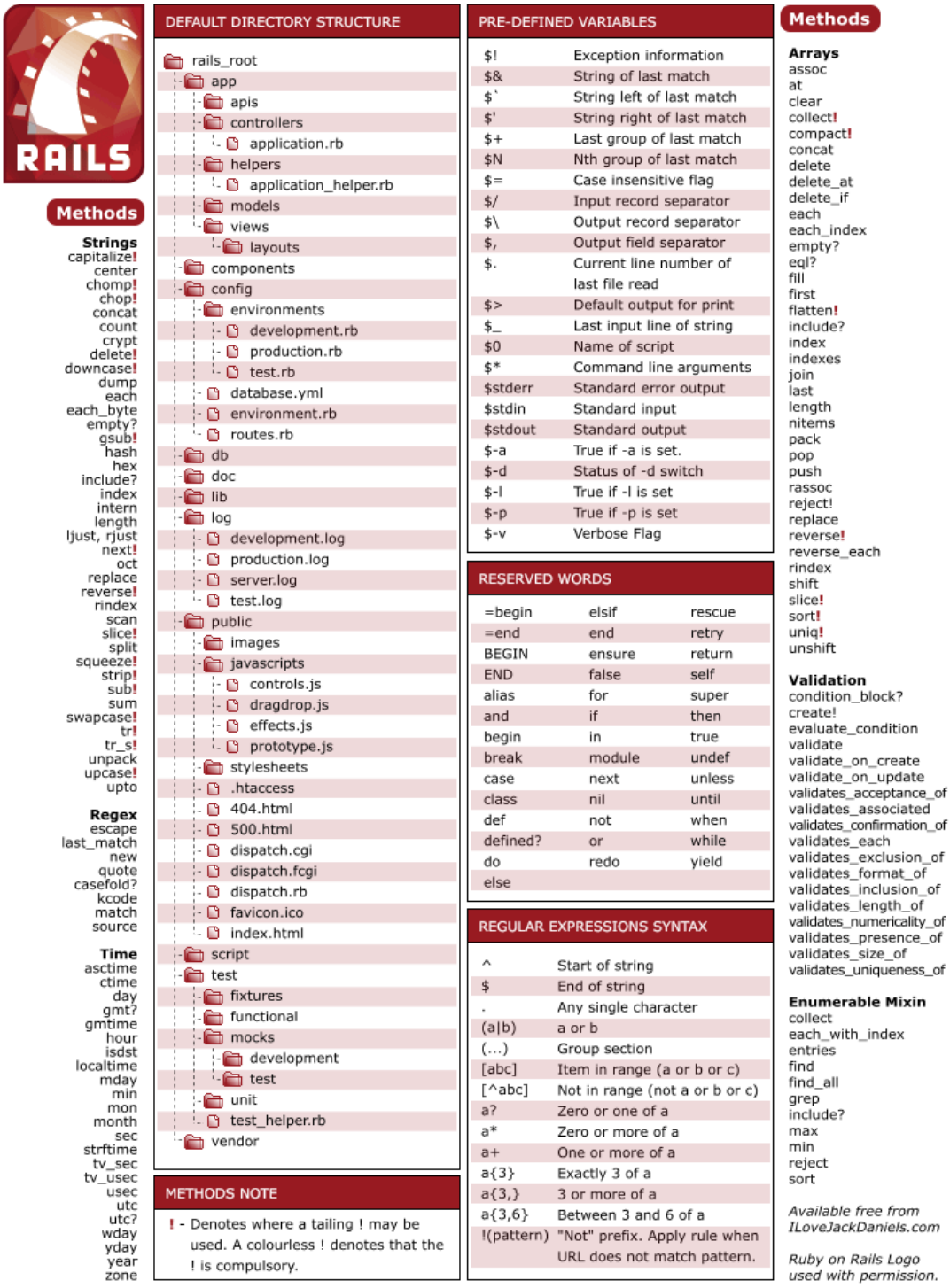
- Dir[*string*] -> array
- Dir::chdir ([*string*])
- Dir::delete (*string*)
- Dir::entries (*string*) -> array
- Dir::foreach (*string*) do |file| ... end
- Dir::getwd -> string
- Dir::mkdir (*string*)
- Dir::new (*string*)
- Dir::open (*string*) do |dir| .. end
- Dir#close
- Dir#pos -> integer
- Dir#read -> string or nil
- Dir#rewind

DateTime

- DateTime::now
- DateTime::parse (str)
- DateTime::strptime (str, format)
- DateTime#day
- DateTime#hour
- DateTime#leap?
- DateTime#min
- DateTime#month
- DateTime#sec
- DateTime#wday
- DateTime#year

Ruby: www.ruby-lang.org
Doc: www.ruby-doc.org

9.2 Apéndice B. Rails



Methods

Strings
capitalize!
center
chomp!
chop!
concat
count
crypt
delete!
downcase!
dump
each
each_byte
empty?
gsub!
hash
hex
include?
index
intern
length
ljust, rjust
next!
oct
replace
reverse!
rindex
scan
slice!
split
squeeze!
strip!
sub!
sum
swapcase!
tr!
tr_s!
unpack
upcase!
upto

Regex
escape
last_match
new
quote
casefold?
kcode
match
source

Time
asctime
ctime
day
gmt?
gmtime
hour
isdst
localtime
mday
min
mon
month
sec
strftime
tv_sec
tv_usec
usec
utc
utc?
wday
yday
year
zone

DEFAULT DIRECTORY STRUCTURE

- rails_root
  - app
    - apis
    - controllers
      - application.rb
    - helpers
      - application_helper.rb
    - models
    - views
      - layouts
  - components
  - config
    - environments
      - development.rb
      - production.rb
      - test.rb
    - database.yml
    - environment.rb
    - routes.rb
  - db
  - doc
  - lib
  - log
    - development.log
    - production.log
    - server.log
    - test.log
  - public
    - images
    - javascripts
      - controls.js
      - dragdrop.js
      - effects.js
      - prototype.js
    - stylesheets
    - .htaccess
    - 404.html
    - 500.html
    - dispatch.cgi
    - dispatch.fcgi
    - dispatch.rb
    - favicon.ico
    - index.html
  - script
  - test
    - fixtures
    - functional
    - mocks
      - development
      - test
    - unit
    - test_helper.rb
  - vendor

METHODS NOTE

! - Denotes where a tailing ! may be used. A colourless ! denotes that the ! is compulsory.

PRE-DEFINED VARIABLES

$!	Exception information
$&	String of last match
$`	String left of last match
$'	String right of last match
$+	Last group of last match
$N	Nth group of last match
$=	Case insensitive flag
$/	Input record separator
$\	Output record separator
$,	Output field separator
$.	Current line number of last file read
$>	Default output for print
$_	Last input line of string
$0	Name of script
$*	Command line arguments
$stderr	Standard error output
$stdin	Standard input
$stdout	Standard output
$-a	True if -a is set.
$-d	Status of -d switch
$-l	True if -l is set
$-p	True if -p is set
$-v	Verbose Flag

RESERVED WORDS

=begin	elsif	rescue
=end	end	retry
BEGIN	ensure	return
END	false	self
alias	for	super
and	if	then
begin	in	true
break	module	undef
case	next	unless
class	nil	until
def	not	when
defined?	or	while
do	redo	yield
else		

REGULAR EXPRESSIONS SYNTAX

^	Start of string
$	End of string
.	Any single character
(a|b)	a or b
(...)	Group section
[abc]	Item in range (a or b or c)
[^abc]	Not in range (not a or b or c)
a?	Zero or one of a
a*	Zero or more of a
a+	One or more of a
a{3}	Exactly 3 of a
a{3,}	3 or more of a
a{3,6}	Between 3 and 6 of a
!(pattern)	"Not" prefix. Apply rule when URL does not match pattern.

Methods

Arrays
assoc
at
clear
collect!
compact!
concat
delete
delete_at
delete_if
each
each_index
empty?
eql?
fill
first
flatten!
include?
index
indexes
join
last
length
nitems
pack
pop
push
rassoc
reject!
replace
reverse!
reverse_each
rindex
shift
slice!
sort!
uniq!
unshift

Validation
condition_block?
create!
evaluate_condition
validate
validate_on_create
validate_on_update
validates_acceptance_of
validates_associated
validates_confirmation_of
validates_each
validates_exclusion_of
validates_format_of
validates_inclusion_of
validates_length_of
validates_numericality_of
validates_presence_of
validates_size_of
validates_uniqueness_of

Enumerable Mixin
collect
each_with_index
entries
find
find_all
grep
include?
max
min
reject
sort

Available free from ILoveJackDaniels.com

Ruby on Rails Logo used with permission.

Resumen de una aplicación Rails

9.3 Apéndice C. Rake

En este apéndice vamos a poner todas las operaciones que podemos realizar con rake en Rails [Rake]:

- **rake cache:clear** - Borra todas las páginas del cache
- **rake db:bootstrap** - Inserta el schema.rb dentro de la base de datos y luego añade los accesorios iniciales de la base de datos.
- **rake db:bootstrap:copy_default_theme** - Copia el tema principal al tema del sitio.
- **rake db:migrate** - Migracion de la base de datos a través del script db-migrate, para especificar la versión añada VERSION =X.
- **rake db:schema:dump** - Crea un archivo schema.rb que puede usarse contra cualquier DB soportada por AR.
- **rake db:schema:load** - Inserta schema.rb a la base de datos.
- **rake db:bootstrap:load -** Carga los accesorios iniciales en db/bootstrap/*.yml en el ambiente actual. Para cargar accesorios específicos utilizar FIXTURES=x,y.
- **rake db:fixtures:load** - Carga los accesorios en el ambiente actual de la base de datos. Para cargar accesorios específicos utilizar FIXTURES=x,y.
- **rake db:sessions:clear** - Borra las sesiones
- **rake db:sessions:create** - Crea una tabla para usar con CGI::Session::ActiveRecordStore
- **rake db:structure:dump** - Descarga la base de datos con estructura de SQL.
- **rake db:test:clone** - Recrea la base de datos de prueba del esquema de la base de datos de ambiente actual.
- **rake db:test:clone_structure** - Recrea la base de datos de prueba con la estructura de la base de datos de desarrollo.
- **rake db:test:prepare** - Prepara la base de datos de prueba y carga el esquema.
- **rake db:test:purge** - Borra la base de datos de prueba.

- **rake deploy** - Envía la última revisión a producción utilizando el ¨release manager¨.
- **rake diff_from_last_deploy** - Describe las diferencias entre HEAD y la última versión en producción.
- **rake doc:app** - Crea la documentación API para la aplicación en /doc.
- **rake doc:clobber_app** - Remueve los productos de rdoc.
- **rake doc:clobber_plugins** - Remueve la documentación de los plugins instalados.
- **rake doc:clobber_rails** - Remueve los productos de rdoc.
- **rake doc:plugins** - Genera la documentación para todos los plugins instalados.
- **rake doc:rails** - Crea lo archivos HTML de Rails.
- **rake doc:reapp** - Fuerza la recreación de los archivos RDOC.
- **rake doc:rerails** - Fuerza la recreación de los archivos RDOC.
- **rake edge** - Congela a Edge Rails.
- **rake log:clear** - Borra todos los datos en /log.
- **rake rails:freeze:edge** - Congela a la última revisión de Edge Rails. Se puede especificar la revisión con REVISION=X o con TAG=Y.
- **rake rails:freeze:gems** - Congela la aplicación a la versión actual.
- **rake rails:unfreeze** - Descongela la versión o revisión de la aplicación.
- **rake rails:update** - Actualiza configuraciones, scripts y los javascripts de Rails.
- **rake rails:update:configs** - Actualiza config/boot.rb de la instalación actual de rails.
- **rake rails:update:javascripts** - Actualiza los javascripts de la aplicación.
- **rake rails:update:scripts** - Agrega nuevos scripts al folder scripts/.
- **rake remote_exec** - Ejecuta una acción específica del ¨release manage¨.
- **rake rollback** - Restaura a la última versión antes de la actual.
- **rake show_deploy_tasks** - Enumera todos las tareas para el lanzamiento de la aplicación.
- **rake stats** - Reporta las estadísticas de código de la aplicación.

- **rake test** - Prueba todas las unidades y funciones.
- **rake test:functionals** - Prueba para functionals db:test:prepare
- **rake test:integration** - Prueba para integration db:test:prepare
- **rake test:plugins** - Prueba para plugins environment
- **rake test:recent** - Prueba para recentdb:test:prepare
- **rake test:uncommitted** - Prueba para uncommitteddb:test:prepare
- **rake test:units** - Prueba para unitsdb:test:prepare
- **rake tmp:cache:clear** - Borra todos los archivos y directorios en tmp/cache.
- **rake tmp:clear** - Borra sesiones, cache y archivos socket de folder /tmp.
- **rake tmp:create** - Crea directorios para sessions, cache, and sockets
- **rake tmp:pids:clear** - Borra todos los archivos en tmp/pids
- **rake tmp:sessions:clear** - Borra todos los archivos en tmp/sessions
- **rake tmp:sockets:clear** - Borra todos los archivos en tmp/sockets
- **rake update_dialog_helper -** Copia los últimos dialog.js a la carpeta public de la aplicación.