
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Sistemas Paralelos y Distribuidos

# Prototipo para un Archivo de Documentos Digitales provenientes de la Web. Modulo de Control, Almacenamiento Estructurado Spiders y Plataforma Cloud Computing

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela
Por los Bachilleres:

**De Almada A., Cindy - C.I. 16.972.363**
**Chai A., Frederick - C.I. 15.206.349**

Para optar al título de Licenciado en Computación
Tutor: **Prof. Andrés Sanoja**
Caracas, Mayo 2011

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Sistemas Paralelos y Distribuidos

**ACTA DEL VEREDICTO**

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por los Bachilleres Cindy De Almada C.I.: 16.972.363 y Frederick Chai C.I.: 15.206.349, con el título "Prototipo para un Archivo de Documentos Digitales provenientes de la Web. Modulo de Control, Spiders y Plataforma Cloud Computing", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 27 de mayo de 2011, a las 11:00 am, para que sus autores lo defendieran en forma pública, en la Escuela de Computación, Facultad de Ciencias de la Universidad Central de Venezuela, lo cual estos realizaron mediante una exposición oral de su contenido, y luego respondieron a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas el 27 de mayo de 2011, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Tutor Andrés Sanoja.

_____
Prof. Andrés Sanoja
(Tutor)

_____ _____
Prof. Eugenio Scalise Prof. Carlos Acosta
(Jurado Principal) (Jurado Principal)

# RESUMEN

En el presente Trabajo Especial de Grado se expone los detalles del desarrollo de una estructura de control y procesamiento de datos centralizado para el Prototipo para un Archivo de Documentos Digitales provenientes de la Web, proyecto encarado por el Centro de Computación Paralela y Distribuida (CCPD) de la Escuela de Computación, el cual realiza un almacenamiento histórico de un grupo de páginas Web introducidas por el usuario.

Para dicho desarrollo, nuestra competencia consiste en la generación de los siguientes componentes: Módulo de control, manejador de tareas en segundo plano, analizador de documentos, almacenamiento estructurado y spider (módulo de procesamiento de URIS), además de toda la estructura lógica para la comunicación entre todos los módulos.

El objetivo que persigue el desarrollo de este trabajo es realizar la centralización de carga de tareas en un módulo de control el cual es el encargado de toda la fase de procesamiento, ejecución, verificación y almacenamiento de datos, además de brindar seguridad a los mismos ya que ésta estrategia evita el contacto directo de los datos e instrucciones de procesamiento con el usuario.

La funcionalidad en grandes rasgos del desarrollo presentado consiste en generar un ambiente el cual partiendo de un grupo de URIs de entrada, pueda extraer información y documentos de dichas direcciones, y de esta manera almacenar todo su contenido y obtener un compendio de información que caracteriza a estas URIs.

Para el desarrollo de este trabajo se planteó el uso de una metodología ágil, debido a tener características que se adaptan a nuestro medio como lo son la programación en pareja, continua colaboración con el cliente y la posibilidad de realizar refactorizaciones, ésta última de gran importancia ya que durante la realización de los módulos se generaron diversos cambios tanto de estructuras como de esquemas hasta llegar a un esquema estable capaz de cumplir con los requerimientos.

En consecuencia, se logró el desarrollo de todos los módulos planteados, una conexión estable entre ellos y el almacenamiento tangente de todos los datos necesarios, por lo cual podemos argumentar que los objetivos planteados fueron alcanzados.

**Palabras Claves:** Cloud Computing, Web Archiving, Datawarehouse, Almacenamiento Estructurado

# TABLA DE CONTENIDO

## INDICE DE FIGURAS

## INDICE DE CUADROS

# INTRODUCCION

Para que el hombre aprenda de sus errores no debe olvidarlos. Cientos de años atrás el hombre se dio cuenta de ello y empezó a construir un sueño, en Egipto, la Biblioteca de Alejandría, el cual fue quemado con todo lo que contenía, pero la idea no se esfumó si no que perduró en el tiempo hasta el punto de que dicha biblioteca se convirtió en una leyenda, en un mito. Muchos dudan de si llegó a ser lo que se cuenta de ella.

Recogiendo esta filosofía de proteger la mente del olvido se han formado muchos otros proyectos a lo largo del tiempo. Pero de una manera extraña han faltado quien guardara las grandes revoluciones mediáticas. Falta un lugar donde encontrar lo más importante relacionado con el mundo de la televisión, del cine o de la radio, aquello que se produjo con los nuevos instrumentos comunicativos.

El último de estos medios es Internet, pero a diferencia de lo ocurrido en Egipto, existen en la actualidad entes dispuestos a guardar su memoria. Para poder lograr esta avariciosa misión se crean los Archivos de Internet (Internet Archive) los cuales buscan la preservación de un histórico de las páginas Web alojadas en Internet.

Como se puede inferir, el explicado proyecto tiene grandes niveles de dificultad, por lo cual, para llevar a cabo un desarrollo de tal magnitud, éste debe dividirse en módulos y desarrollos con cargas de trabajo estratégicamente balanceadas.

En las ciencias de la computación, el término *divide y vencerás* hace referencia a uno de los más importantes paradigmas de diseño algorítmico. El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original. Esta técnica es la base de los algoritmos eficientes para casi cualquier tipo de problema, así mismo sirve para el desarrollo de aplicaciones con múltiples niveles de dificultad.

En el presente Trabajo Especial de Grado se expone el desarrollo del Módulo de Control, Almacenamiento Estructurado, Spider, y plataforma Cloud Computing como fragmento del proyecto que lleva a cabo el Centro de Computación Paralela y Distribuida (CCPD) de la Escuela de Computación, Prototipo para un archivo de documentos digitales provenientes de la Web.

Esto se llevó a cabo con la intención de obtener un software tolerante a fallas y configurable, además de centralizar el procesamiento en un módulo de control general, el cual presta servicios a una aplicación de administración. De esta manera poder enfocar el desarrollo a la arquitectura Modelo- Vista- Controlador y definir las actividades de manera más balanceada.

Para el desarrollo de los módulos presentados en este Trabajo Especial de Grado, se tomaron en cuenta ciertas acotaciones, como lo son, el desarrollo de aplicaciones utilizando lenguajes de programación basados en código abierto, utilización de bases de datos como repositorio de información, utilización del framework de aplicaciones Web Ruby On Rails y comunicación entre módulos utilizando el esquema REST.

Para mejorar la comprensión del lector, este documento se ha estructurado en los siguientes capítulos:
CAPÍTULO 1: PLANTEAMIENTO DEL PROBLEMA. Describe el problema y la solución planteada, además se define el objetivo general, los objetivos específicos, el alcance y la justificación del presente trabajo de investigación.

CAPÍTULO 2: MARCO TEÓRICO REFERENCIAL: presenta los fundamentos conceptuales que sustentan el trabajo de investigación y la presentación de cada una de las tecnologías usadas para crear las aplicaciones y librerías que conforman el controlador general y spider.

CAPÍTULO 3: MARCO METODOLOGICO: presenta el método de desarrollo usado y su adaptación al proyecto propuesto

CAPITULO 4: MARCO APLICATIVO: presenta las historias de usuarios generadas durante el desarrollo de las aplicaciones y librerías y la descripción del las iteraciones sujetas a estas historias de usuarios.

Finalmente se presentan las Conclusiones, recomendaciones y bibliografía consultada a lo largo de Trabajo Especial de Grado.

# CAPITULO 1 - PLANTEAMIENTO DEL PROBLEMA

## 1.1 Título

Prototipo para un archivo de documentos digitales provenientes de la Web. Módulo de control, Almacenamiento Estructurado, Spiders y plataforma Cloud Computing.

## 1.2 Planteamiento del Problema

En la actualidad, una de las herramientas con mayor auge de utilización a nivel mundial y de mayor capacidad de información a distancia es la gran red de comunicaciones llamada Internet. Esta se ha convertido en un medio idóneo para mantener un contacto en tiempo real, tanto a nivel empresarial como personal entre sus usuarios, además de un funcionamiento tangible en el ámbito de información, educación, comercialización y entretenimiento.

Esta gran red de información se presenta con una amplia gama de sitios Web donde se puede indagar temas de cualquier índole. Se tiene por ejemplo: sitios de descargas de archivos, blogs, sitios de información de empresas, sitios de comercio electrónico, wiki, sitios de comunidades virtuales, sitios de juegos, etc.

Varios son los esfuerzos para preservar documentos, libros, entre otros, para su consulta pública y para las generaciones futuras. Cada día más información se encuentra en formato digital, las noticias se generan y se publican en la Web. Preservar este tipo de información es una necesidad para el mundo. Esta preservación de información digital se denomina Archivo Digital. Un Archivo Digital es un amplio archivo de Internet que permite almacenar y permitir el acceso a versiones anteriores de páginas Web que ya no existen.

En el marco de las directrices para la preservación del patrimonio digital de la UNESCO (2003) sobre la digitalización y la accesibilidad en línea del material cultural y la conservación digital, se encuentra el proyecto del Centro de Computación Paralela y Distribuida (CCPD) el cual cubre el desarrollo de un prototipo para la Construcción del Archivo de Documentos Digitales provenientes de la Web, con el fin de conservar los productos culturales "archivos digitales", para que puedan convertirse en un patrimonio cultural e intelectual perdurable para las generaciones presentes y venideras de Venezuela.

Como parte de este proyecto se plantea desarrollar un módulo de control que gestione, verifique y clasifique las entradas que le provee un panel de control, además de interfaces para el uso del repositorio No-SQL, desarrollo de interfaces para el uso de una plataforma de Cloud Computing, desarrollo de librería que implementa las políticas de uso de los URI (Uniform Resource Identifier) requeridas para el proyecto, desarrollo de los spiders y exportación de documentos a varios formatos (Ej: PDF).

## 1.3 Objetivo General y Específico

En la siguiente sección se definen los objetivos que surgieron en cuanto a la investigación realizada, los cuales se pretenden lograr según el alcance que se plantee.

### 1.3.1 Objetivo General

Este Trabajo Especial de Grado tiene como objetivo el desarrollo del módulo de control y del módulo de procesamiento de URLs (spider) para la gestión de una aplicación prototipo de archivo Web el cual se encargue de la gestión, almacenamiento y procesamiento de grupos de páginas que se encuentran alojadas en internet de manera eficiente y orientado al esquema de software libre.

### 1.3.2 Objetivos Específicos

- ✓ Adaptar y aplicar el Método de Programación Extrema (XP por sus siglas en inglés) y seguir los principios de la modelación ágil para conseguir un desarrollo dinámico, rápido y evolutivo.

- ✓ Desarrollar las aplicaciones utilizando lenguajes de código abierto.
- ✓ Desarrollar librerías cuya ejecución se realice desde una plataforma que use el paradigma de Cloud Computing y proveer múltiples servicios a la aplicación.
- ✓ Desarrollar ambiente de programación capaz de ejecutar tareas en segundo plano.
- ✓ Desarrollar controlador que verifique entrada, genere salida de ser necesario y guarde datos en una base de datos.
- ✓ Desarrollar una librería para la validación de formato valido de URIs.
- ✓ Desarrollar un módulo de extracción de información (código HTML, documentos, multimedia, etc.), de una página Web dada la URL.
- ✓ Desarrollar módulo de generación de documentos a PDF capaz de almacenar la apariencia de una página Web dada la URL.
- ✓ Analizar salidas que el spider genera para guardarlas en base de datos.

## 1.4 Componentes Involucrados

- ✓ **Cpanel:** módulo externo a nuestra competencia en el desarrollo de este trabajo, el cual provee una interfaz de usuario con la cual se empaquetan en grupos las URIs que se desean procesar, y se realiza la petición de ejecución de procesamiento. Este módulo está conectado al módulo de control (Rest_controller) utilizando el esquema REST, el cual permite una comunicación orientada a objetos en ambiente Ruby on Rails, y envía los paquetes a éste mismo.
- ✓ **Rest_controller**: módulo controlador capaz de verificar los datos recibidos por el Cpanel, en este caso una lista de URIs, verifica y almacena datos de importancia del procesamiento de estas URIs en una base de datos MySQL, realiza una petición y envía al módulo Robot las URIs verificadas.
- ✓ **Robot:** módulo encargado de generar y gestionar los procesos en segundo plano que se ejecutan al iniciar el procesamiento de alguna URI, se encarga de la asignación de URIs a los Spiders, almacena la información de documentos extraídos de páginas Web en una base de datos MongoDB y finalmente recibe de los spider una lista de URIs incrustadas en las páginas procesadas.
- ✓ **Spider**: módulo encargado de la extracción de información de páginas de las URIs recibidas, creación de objetos con la URI procesada, conseguir el tipo MIME, y el contenidos de éstas, y envío al Robot para su almacenamiento.

- ✓ **Cloud Server:** El Cloud Server es una solución de Cloud Computing que permite la virtualización de los servidores con alto rendimiento y aloja los recursos de manera individual y segura para cada cliente. Es ideal para sustituir su infraestructura física de TI por una solución virtualizada robusta, segura y escalable con costo reducido, el Cloud Server provee un ambiente de ejecución de las aplicaciones y principalmente de gestión de nodos llamados cloud node.
- ✓ **Cloud node:** es la representación virtual de una máquina capaz de ejecutar aplicaciones y generar respuestas a peticiones dadas.

## 1.5 Importancia y Justificación

Basado en el modelo-vista-controlador (MVC) en primera instancia la idea es equilibrar la carga de responsabilidades generadas por la aplicación dividiéndolo en módulos con propósitos específicos, es por esto que nace la idea de dividir el prototipo de Archivo Web en CPanel, Rest_Controller, el ambiente Cloud Computing y el Datawarehouse. El propósito de realizar un controlador general (Rest_Controller) en el proyecto es permitir centrar las responsabilidades pertinentes a procesamiento de datos generados por el CPanel. Es importante mencionar que debido a la importancia de preservar la consistencia de los datos, el usuario debe tener acceso al módulo de procesamiento de datos; por esto se crea esta capa intermediaria entre el CPanel y los Spiders.

## 1.6 Propuesta de la Solución

Es importante el diseño de una arquitectura que permita entender el alcance y la solución que se requiere para el sistema final. En la figura 1, se presenta el flujo de datos, control e interacción que hay entre los módulos.

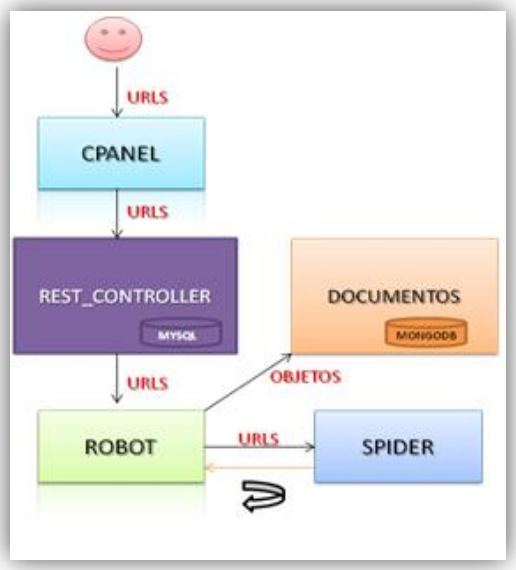



**Figura 1 - Diagrama lógico de la aplicación**

Teniendo en cuenta el problema planteado y los objetivos propuestos, se propone la siguiente solución:

- ✓ Crear una aplicación de control que reciba como entrada un grupo de URLs que envía el panel de control CPanel, verificar que estas URLs sean válidas y no estén bien formadas, realizar la operación requerida y guardar o actualizar los datos en la base de datos dependiendo de la operación requerida.

- ✓ Generar un grupo de URLs válidas, enviarlas al spider para que verifique el tipo MIME de cada una de las URLs, descomponer cada una en documentos (imágenes, audio, video, etc.) o URLs, guardarlo en un objeto que contenga la URL, el contenido y el tipo MIME si es un documento, y enviarlos como salida (objetos y URLs) para que luego sea procesado y almacenarlo en una base de datos MongoDB. Este proceso se debe de realizar hasta que la salida contenga sólo documentos y no URLs.

Con la idea de poder procesar un grupo de URIs simultáneamente, nace la idea de generar varios spiders que ejecuten el procesamiento de cada URI de manera concurrente. Con la idea de tener varios spiders ejecutándose concurrentemente y a la espera de recibir las URIs para procesar, nace la idea de alojar cada spider en maquinas virtuales, de tal manera de que se puedan tener múltiples

spiders ejecutándose sin tener como impedimento la falta de hardware para alojarlos, y es con esta idea suscita la necesidad de una plataforma que proporcione herramientas de software, plataformas e infraestructura de servicios. Estas consideraciones nos llevan a concluir que la plataforma de Cloud computing se adapta de manera adecuada a las exigencias planteadas anteriormente, además de sus características de flexibilidad y adaptabilidad no genera mayor problema en su incorporación a nuestro sistema.

## 1.7 Alcance

Se espera que esta aplicación de control (Rest_Controller), el Spider y el ambiente de Cloud Computing tenga un aporte positivo y productivo para el desarrollo final de la aplicación de Archivo Web cuyo proyecto lleva a cabo el Centro de Computación Paralela y Distribuida de la Escuela de Computación, ya que permite realizar las siguientes actividades:

- ✓ Recibe del CPanel un grupo de URLs, las cuales verifica, clasifica y procesa a una base de datos para luego mandarlas a ejecutar.
- ✓ Implementa las políticas de uso de las URI (Uniform Resource Indentifier) requeridas para el proyecto.
- ✓ Ejecutar tareas en ambiente Cloud.
- ✓ Generar archivo PDF de URIs procesadas y almacenadas en base de datos estructurada.
- ✓ Generar módulo de acceso a la base de datos estructurada para guardar contenido de URIs.
- ✓ Ejecutar códigos en segundo plano utilizando el plugin de Rails Workling en conjunto con el manejador de colas Starling.

# CAPITULO 2 - MARCO TEÓRICO REFERENCIAL

## MARCO REFERENCIAL

## 2.1 Preservación digital

La preservación digital se basa en la búsqueda de soluciones para conservar aquellos documentos digitales almacenados, sea cual sea su formato, el software, hardware o sistema que se utilizó para su creación, manteniendo así la información pese a los rápidos cambios tecnológicos.

Según algunos autores, la preservación digital consiste en asegurar el acceso continuo a los materiales digitales, y de manera general se puede decir que consiste en mantener la capacidad de visualizar, recuperar y utilizar colecciones digitales frente a las infraestructuras y elementos tecnológicos y de organización que cambian con mucha rapidez.

En la preservación digital se encuentra la dicotomía muy interesante. ¿En qué consiste realmente, en la digitalización de los originales, en la conservación de los propios documentos digitales o en ambas cosas? Se puede entender por preservación digital la preservación de los artefactos físicos mediante su digitalización, pero también la preservación de los propios recursos digitales. Una vez teniendo en cuenta esto, se puede pensar en los objetivos que tiene esa preservación a través de la digitalización. [1]

### 2.1.1 Objetivos de la preservación digital

- ✓ Incrementar el acceso a la colección.
- ✓ Mejorar los servicios para un grupo de usuarios en expansión aumentando el acceso a las fuentes documentales de la institución.
- ✓ Reducir la manipulación de los materiales frágiles y hacer una “copia de seguridad” de los mismos mediante su digitalización.

- ✓ Brindar a la institución la oportunidad de desarrollar su infraestructura técnica y la destreza de su plantilla.
- ✓ Desarrollar fuentes de información en colaboración con otras instituciones accesibles a través de Internet.
- ✓ Rentabilizar los avances tecnológicos de otras instituciones compartiéndolos.

Una de las principales razones para llevar a cabo la digitalización es incrementar el acceso a la colección sobre todo a la hora de acceder a fondos delicados como manuscritos, libros raros, etc. La digitalización de este tipo de fondos supone facilitar el acceso a unos fondos que deben ser visitados en un lugar específico bajo unas condiciones de consulta concreta. Estos fondos pueden ser fácilmente dañados y una manera de salvarlo es su digitalización. Además es más productiva su consulta a través de herramientas informáticas.

### 2.1.2 Plan de preservación

La definición de un plan de preservación pasa por dar respuesta a las siguientes preguntas [2]:

- ✓ ¿Qué es lo que hay que guardar y porque?
- ✓ ¿Dónde se debe guardar?
- ✓ ¿Hasta cuándo guardarlo?
- ✓ ¿Cómo recuperarlo después?
- ✓ ¿Cómo mantenerlo inalterado?
- ✓ ¿Cómo evitar que se vuelva obsoleto?

*¿Qué es lo que hay que guardar y porque?*
Los criterios de selección pueden basarse en las políticas de otras instituciones a la hora de llevar a cabo un plan de digitalización. En general, se debe seleccionar y crear colecciones digitales con un valor duradero y de interés intelectual. La demanda de los documentos también es un factor a tener en cuenta ya que si los fondos están digitalizados serán de más fácil acceso.

La condición física en que se encuentren los documentos también se ha de tener en cuenta ya que será más urgente la preservación de un documento que corre un gran riesgo físico por deterioro, por ejemplo que otro en mejores condiciones.

Esta selección debe estar avalada por una política de preservación bien definida y apoyada económicamente. Esta política debe ser revisada periódicamente junto con la selección de documentos a preservar. A estos documentos se les debe dar un valor límite de vida ya que algunos serán más perecederos que otros.

*¿Dónde se debe guardar?*

Como se sabe los soportes digitales son frágiles y efímeros, además adolecen de ciertas características como posibles fallos técnicos y mecánicos, errores del operador, virus, cambios no autorizados y no documentados, obsolescencia o incompatibilidad del software, pérdida de programas, metadatos incompletos, envejecimiento de la información, entre otros. La selección de formatos donde guardar la información debe ser parte del plan global de preservación del proyecto, que debe abarcar también otros aspectos tales como los procedimientos de preservación a seguir.

Por otro lado, ninguno de los medios digitales de hoy en día garantiza la longevidad de la información. Los medios magnéticos tienen una vida sorprendentemente corta, los discos compactos son más duraderos pero nadie puede predecir su durabilidad que depende en gran medida de la calidad de los mismos.

*¿Hasta cuándo guardarlo?*

A los objetos preservados se les debe asignar un valor límite de vida y llegado esa fecha, la institución, mediante su política de preservación digital deberá decidir qué hacer con ese objeto digital.

*¿Cómo recuperarlo después?*
*¿Cómo mantenerlo inalterado?*
*¿Cómo evitar que se vuelva obsoleto?*

Estas tres cuestiones están sumamente relacionadas entre ellas ya que la correcta recuperación depende de cómo se mantengan los recursos digitales y su mantenimiento depende en gran medida de evitar su obsolescencia.

Entre los métodos para evitar la obsolescencia de los recursos digitales, se puede citar, por ejemplo, la renovación periódica de los archivos para evitar la pérdida de datos debida al envejecimiento del soporte, la conversión de viejos formatos a otros más nuevos para evitar la obsolescencia, los

criterios de redundancia, y la elección adecuada de medios y lugares de almacenamiento.

Para favorecer la longevidad de la información digital se puede almacenar la información en formatos que sean ampliamente usados hoy en día. Esto aumenta la probabilidad de que cuando un formato se vuelva obsoleto aún existan programas para su conversión. XML, HTML y PDF son ejemplos de estos. Otra interesante sugerencia es crear un archivo que contenga las definiciones de los formatos, estándares de metadatos, protocolos y otros elementos constructivos fundamentales de las bibliotecas digitales. Si los formatos y los esquemas de codificación son preservados, la mayoría de la información puede ser descifrada posteriormente. [2]

## 2.2 Internet Archive

Internet Archive es una de las bibliotecas digitales más importantes de la red. Como la mítica biblioteca de Alejandría, entre sus cimientos se encuentra almacenado gran parte del conocimiento de los últimos años. Ante todo especificar que la mayor parte de la documentación que guarda y protege es de origen estadounidense, aunque algunas secciones dan cabida también al conocimiento que proviene de otros países. Internet Archive asume el reto de preservar el "pasado" de Internet (aunque se limite a finales de los años 90). [3]

### 2.2.1 Origen

Nace en 1996 en el presidio de San Francisco, crece gracias a las aportaciones de Alexa Internet y de la Library of Congress (una de las cinco mayores bibliotecas del mundo). Tiene una gran cantidad de archivos misceláneos como audio, video y texto, todo ello bajo dominio público. De las secciones de la página deben destacarse algunas colecciones de archivos debido a su gran magnitud, por ejemplo:

- ✓ Moving Images almacena más de 18.592 archivos.
- ✓ Prelinger Archives contiene 1977 archivos relacionados con el cine efímero publicitario, educacional, industrial y amateur.

- ✓ Wayback Machine, sección con 40 mil millones de páginas grabadas desde 1996 (por ejemplo: se encuentra la primera versión de Amazon, la primera página de ventas por Internet).

Internet Archive no pretende conservarlo todo, entre otras razones porque no es posible. Por eso es interesante saber que hay muchas más bibliotecas digitales.

Fue creado con la intención de empezar un nuevo gran proyecto, no dejar que el tiempo acabe con la memoria de quienes somos ahora. La labor de toda biblioteca es guardar la sabiduría, pero también dejar un reflejo de la sociedad en un tiempo determinado, para que después, años más tarde haya alguna cosa que recordar, que entender, que investigar, que perdonar, que aprender.

Esta razón de ser explica que esta Web haya sido creada sin ningún ánimo de lucro, que todos sus archivos sean de dominio público y que comparta sus contenidos con otras instituciones, por ejemplo en el año 2003, Internet Archive cedió una copia de sus 10 billones de archivos a la Biblioteca Alexandrina (otra de las cinco más importantes del planeta situada en Egipto).

El Internet Archive es además un espacio que permite que cualquier webmaster publique sus contenidos, suba videos, audios, etc.

### 2.2.2 Filosofía

Recogiendo la filosofía de proteger la mente del olvido se han formado muchos proyectos a lo largo del tiempo. Falta un lugar donde encontrar lo más importante relacionado con el mundo de la televisión, del cine o de la radio.

Internet es el último de estos medios, con la ventaja añadida del multimedia, es decir, de la posibilidad de la Web de albergar archivos de imagen, video y sonido, quizás se trate también del soporte perfecto para esos documentos que podrían haberse perdido.

Algunos se preguntarán que diferencia puede haber entre este tipo de bibliotecas y los conocidos catálogos como por ejemplo Yahoo. La divergencia radica sobretodo en el uso que se le da a este servicio. Por ejemplo, se acude a Yahoo para buscar una información o site, la biblioteca en cambio aglutina archivos y documentos (aunque algunos de estos sean también páginas web). Algunos de los websites que se encuentran en Internet Archive forman parte de un pasado, que aunque cercano

ya es caduco, y que, por tanto, ya no se pueden encontrar desde ningún otro medio, catálogo o buscador. Para dar un paseo por los primeros años de la Red hay que acceder a la sección Way-back machine.

Además, la biblioteca quiere ayudar al investigador a hacerse una idea de cómo es o ha sido una sociedad concreta (o algunos de sus aspectos). Pone en manos del visitante una serie de archivos más o menos relacionados que le ayudará a sacar conclusiones y/o aprender. El buscador, tan solo quiere resolver una pregunta rápida: el nombre de un presidente, una fecha, la dirección de una página web, una tienda de discos, etc.

Internet Archive está pensado para ayudar al investigador, al historiador y al estudiante, sea de primaria o universitario, incluso puede ayudar al economista a observar como la Web afecta al comercio si se para a analizar los links de las Webs archivadas o al periodista a mirar por un espejo de la sociedad.

### 2.2.3 Evitando las limitaciones físicas

El nuevo formato de biblioteca implica nuevas ventajas en la conservación de los documentos. Por ejemplo, en la Red, los archivos están a salvo de lluvias, incendios y otras catástrofes naturales como sucedió con la primera Biblioteca de Alejandría.

Hay además, copias de seguridad de la mayoría de colecciones. Por otro lado, las nuevas tecnologías permiten cada vez almacenar mayor número de información más fácilmente y en menos espacio.

Internet Archive pretende también tener a la disposición del usuario todo aquel software y aplicaciones que ha quedado obsoleto pero que es necesario para acceder a documentos que no son compatibles con las nuevas versiones en software.

### 2.2.4 Las Limitaciones

Al principio del capítulo se menciona que Internet Archive no puede conservarlo todo. Este es su principal problema, quien y como selecciona los documentos que van a ser “rescatados” del olvido.

Esto ya no es tarea fácil en cualquier biblioteca que se pretenda, pero en el caso de Internet Archive y cualquier otra digital hay que añadir ciertas dificultades:

Internet Archive guarda documentos en varios formatos. Se puede acceder tanto a otros sites que son interesantes en sí mismos, a los que son interesantes por los archivos que contienen o, directamente, a las colecciones de vídeos y audio que alberga la página. Así pues, la magnitud del campo del que hay que extraer la selección de archivos es mucho mayor al que se enfrentaría una biblioteca tradicional.

Porque la red es el medio de la libertad por excelencia, cualquier persona con conexión a Internet y un ordenador puede acceder a ella, y muchas de estas personas tienen conocimientos necesarios para crear documentos y/o cargarlos al sistema. Con esto se aumenta la cantidad de "ruido" en el medio. Filtrar la información para los seleccionadores no es tarea fácil.

## 2.3 Wayback Machine

Wayback Machine [4] es para muchos uno de los proyectos más apasionantes e interesantes de Internet. Si hubiera que salvar para el futuro sólo uno de todos los que actualmente existen, tal vez debería ser éste. Se trata de un archivo histórico de cómo es y era la Web, desde más de una década: una especie de "memoria histórica" construida rastreando y guardando copias completas de todas las páginas web, imágenes y documentos que en ella aparecen.

Tan histórica es su importancia que hace tiempo realizaron una copia de seguridad de toda la información en grandes discos y los enviaron a la biblioteca de Alejandría, para que fueran preservados en caso de catástrofe.

El actual propietario de la Wayback Machine es The Internet Archive, una organización sin fines de lucro que consiguió una de las primeras copias del contenido completo de la Web gracias al trabajo de Alexa Internet -una empresa que ahora es propiedad de Amazon- que se dedicaba a esas labores de rastreo con finalidades de marketing.

Este archivo contiene en la actualidad más de 5 petabytes (5 millones de gigabytes) de información, y la herramienta que ofrecen a los usuarios para navegar por tan vasta biblioteca es la "máquina del tiempo" denominada Wayback Machine, que ahora se ha renovado.

### 2.3.1 ¿Cómo funciona?

La nueva Wayback Machine funciona de una manera tan simple, basta teclear la dirección (URL) de una página Web cualquiera para ver cuál es la última copia que hay guardada en el archivo. El diseño de la portada se ha simplificado y ahora es más claro y elegante. A partir de aquí, las cosas también cambian un poco: en la versión anterior se podía acceder a una lista a modo de calendario que mostraba los momentos clave en que la web en cuestión había sido rastreada y "guardada" en el archivo. Ahora hay una nueva barra en la parte superior de la pantalla, que permite ver gráficamente esos momentos a lo largo del tiempo. La longitud de las barras del gráfico indica en qué meses se hicieron más copias, lo cual está normalmente asociado a una mayor actividad en la Web en cuestión: más grandes, más actualizaciones.
También se pueden navegar con los controles por los distintos años o acceder al calendario. En esta nueva versión lo que aparece es un calendario al estilo tradicional con los días clave en los que se hizo una copia de la página Web marcados con un recuadro de color.

### 2.3.2 ¿Para qué sirve?

El archivo, que comenzó a guardarse por el año 1996, contiene todo tipo de Webs. Por un lado los «arqueólogos digitales» lo utilizan para examinar cómo eran las versiones de los sitios oficiales de ciertas empresas o proyectos en la antigüedad, dado que las páginas actuales raras veces conservan los diseños anteriores.

Por otro lado, si una Web desaparece o se borra, aun así permanecerá en el archivo. Este es el caso de Webs y dominios que cambian de propietario o simplemente de proyecto.

A los investigadores privados tampoco les viene mal este archivo, pues es extraño por no decir imposible que algo que estuvo publicado durante un tiempo razonable no haya sido capturado por la máquina y guardado para siempre.

La mayor parte de la gente conoce únicamente la famosa "caché de Google", un enlace que aparece junto a los resultados de las búsquedas que muestra cómo es o era una página aunque haya desaparecido temporalmente por alguna razón. Pero cuando una página cambia, es borrada o trasladada de forma definitiva, Google normalmente actualiza su memoria caché, borrando la vieja copia. Pero si bien Google guarda únicamente una versión de cada página, la Wayback Machine las guarda todas. Es una especie de "fotografía del momento" que perdura para siempre.

Aun así, quienes prefieran que ciertos contenidos desaparezcan para siempre tarea nada fácil, sólo tienen que crear un archivo llamado robots.txt y ponerlo en su Web para evitar que Google, la Wayback Machine u otros "robots" accedan al contenido. Al hacer esto el sistema de Archive.org también borrará los documentos archivados en el pasado dado que, como ellos mismos dicen, "no tenemos ningún interés en dar acceso a Webs, documentos o guardar cosas que la gente no quiere que guardemos".

# MARCO TEÓRICO

## 2.4 Ruby on Rails

Ruby on Rails es un entorno de desarrollo Web de código abierto, escrito en lenguaje Ruby que sigue el paradigma de la arquitectura Modelo-Vista-Controlador (MVC). [5][6][7]

Ruby on Rails es un framework cuyo lema es facilitar el desarrollo, despliegue y mantenimiento de aplicaciones Web. Persigue combinar simplicidad con la posibilidad de desarrollar escribiendo menos líneas de código que utilizando otros frameworks. [8]

### 2.4.1 Lenguaje de Programación Ruby

Es un lenguaje de scripts, multiplataforma, totalmente orientado a objetos. Ruby es un lenguaje de programación interpretado y su implementación oficial es distribuida bajo una licencia de software libre.

Entre las características del lenguaje se encuentran:

- ✓ Posibilidad de hacer llamadas directamente al sistema operativo.
- ✓ Muy potente para el manejo de cadenas y expresiones regulares.
- ✓ No se necesita declarar las variables.
- ✓ La sintaxis es simple y consistente.
- ✓ Gestión de memoria automática.
- ✓ Todo es un objeto.
- ✓ Métodos Singleton.

### 2.4.2 Framework Rails

Es un marco de trabajo de código abierto para Ruby que sirve para desarrollar aplicaciones Web, está basado en el patrón de diseño MVC.

Rails está basado en dos principios de desarrollo:

- ✓ **Don't Repeat Yourself (DRY - no te repitas):** según este principio, ninguna pieza de código debe estar duplicada, esto se debe a que la duplicación incrementa la dificultad en los cambios y la evolución posterior.

  Con este principio lo que se busca es que el desarrollo sea más rápido y con menos errores, lo que hará que el código sea fácil de entender, mantener y mejorar. Las definiciones deben hacerse una sola vez, para luego ser reutilizadas de ser necesario. Un ejemplo es al definir un formulario, este puede ser llamado desde cualquier vista y cuantas veces sea necesario con tan solo una línea de código.

- ✓ **Convention Over Configuration (Convención sobre Configuración):** según este principio el seguir las convenciones de Rails se traduce en escribir menos código, ya que solo se definen aquellas configuraciones que no son convencionales. En lugar de archivos de configuración, Rails utiliza una serie de convenciones simples que le permiten averiguarlo todo mediante reflexión y descubrimiento. De esta forma el código de la aplicación y la base de datos ya contienen todo lo que este necesita saber.

  Un ejemplo de este principio es que, si hay una clase Estudiante en el modelo, la tabla correspondiente de la base de datos es estudiantes, pero si la tabla no sigue la convención (por ejemplo alumnos) debe ser especificada manualmente (set_table_name "alumnos").

La arquitectura de Rails trabaja con el patrón MVC, por lo tanto el marco de trabajo (framework) obliga al programador a realizar y mantener de una manera separada los códigos para los modelos, controladores y vistas. Esto se evidencia al momento de crear una aplicación, ya que automáticamente se genera una estructura de directorios muy bien definida que separa y encapsula cada porción de código correspondiente a los componentes del modelo MVC, y otros componentes adicionales como librerías externas, librerías definidas por el desarrollador, archivos de configuración, archivos .js, entre otros.

Al utilizar el entorno de desarrollo Ruby on Rails implementando la arquitectura MVC, la correspondencia que se genera es:

| | | |
|---|---|---|
| Modelo | → | ActiveRecord |
| Vista | → | ActionView |
| Controlador | → | ActionController |

#### *2.4.2.1 Modelo - ActiveRecord*

El Modelo consiste en las clases persistentes (que representan a las tablas de la base de datos). En Ruby on Rails, las clases del Modelo extienden la clase ActiveRecord.

Esta clase implementa el patrón de diseño ActiveRecord que permite corresponder objetos a relaciones, es decir, provee una interfaz entre las tablas de una base de datos relacional, y el código de la aplicación escrita en Ruby que manipula los registros de la base de datos.

Rails añade automáticamente atributos y métodos a la clase basándose en las columnas de la tabla, por lo que automáticamente se corresponden las clases con las tablas y los atributos con las columnas.

*Relaciones ActiveRecord* [9]

Trabajar con relaciones es una de las tareas más importantes de los marcos de trabajo de persistencia. La idea es manejar las relaciones con un alto rendimiento de cara al usuario final y gran sencillez de cara al desarrollador. *ActiveRecord* aprovecha las convenciones de Ruby para simplificar el acceso a datos relacionados, definiendo las siguientes relaciones:

- ✓ *Belongs_to:* Significa "pertenece a" y representa la relación "de muchos a uno" en el extremo de "muchos"
- ✓ *Has_many*: Representa el lado "uno" de la misma relación anterior, es decir es la otra cara de la relación *belongs_to.*
- ✓ *Has_one*: Representa la relación "uno a uno", se puede usar en conjunto con "*belongs_to*" o con *"has_one*"
- ✓ *Has_and_belongs_to_many*: Representa la relación de "muchos a muchos" donde hay una tabla adicional con las claves primarias de las tablas relacionadas, pero no hay más datos.

#### *2.4.2.2 Vista – ActionView*

La Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador. Existe una vista para cada acción del controlador y son éstas las que definen la presentación que tendrán los datos en los archivos HTML que se enviarán al navegador.

El método que se emplea en Rails para gestionar las vistas es usar Ruby Embebido, que son archivos de extensión .rhtml, que contienen una combinación de fragmentos de código HTML con código en Ruby.

#### *2.4.2.3 Controlador – ActionController*

El Controlador corresponde a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del modelo y muestra los resultados usando las vistas.

El ActionController se encuentra entre un *ActiveRecord* (interfaz para la base de datos) y un ActionView (motor de presentación de los datos). El ActionController provee facilidades para la manipulación y organización de la data que se extrae de la base de datos, y la que viene como entrada en los formularios Web; para luego redireccionar a la plantilla de vista correspondiente con los datos resultantes del procesamiento previo. Cada Controller es una clase de Ruby, y cada método dentro de dicha clase representa un Action. Utilizan las rutas para saber qué controlador se usará, qué método del mismo se ejecutará y qué parámetros se le pasarán. Por defecto, las rutas presentan la siguiente estructura:

```
http://host/<controlador>/<método>[/<id>]
```

Al crear una nueva aplicación de Ruby on Rails se crearán una serie de directorios y archivos, los cuales representan una jerarquía organizada y predeterminada por Rails para estructurar los proyectos, llamada también Andamiaje (*scaffold)*.

Cada directorio cumple una función específica, a continuación se hará la descripción de algunos de estos:

- ✓ App: Almacena los componentes de la aplicación: vistas y *helpers*, controladores y modelos.
- ✓ Config: Contiene la configuración de la aplicación: configuración de la base de datos (database.yml), estructura del entorno Rails (environment.rb) y el enrutador de peticiones Web (routes.rb).
- ✓ Public: es similar al directorio *public* de un Servidor Web. Contiene los archivos *JavaScript*, imágenes, hojas de estilos y código HTML común para las interfaces.
- ✓ Script: Este directorio contiene *scripts* que inician y administran las diversas herramientas que se pueden utilizar en Rails; por ejemplo, se tienen los *scripts* que generan código (*generate*) e indican un Servidor Web (Server).

Rails sabe dónde encontrar todos los componentes que necesita dentro de la estructura de directorios presentada anteriormente de tal forma que no es necesario especificar detalles del despliegue de la aplicación. El trabajo de los desarrolladores se centra en crear y editar archivos dentro del directorio app, específicamente en los subdirectorios, *controller*, *views* y *model*.

### 2.4.3 Gemas

En su forma más básica, una gema de Ruby es un paquete. Contiene los archivos y la información necesaria para ser instalado en el sistema. Citando RubyGems, «una gema es un empaquetado de aplicaciones Ruby o librería. Tiene un nombre (por ejemplo rake) y una versión (por ejemplo, 0.4.16)». **[10]**

Las gemas son de gran importancia en el Rubyland. Fácilmente se puede utilizar para ampliar o modificar la funcionalidad en las aplicaciones Ruby.

Las gemas son similares a los paquetes Ebuilds. Contienen información de los paquetes junto con los archivos de instalación. Por lo general, construidas a partir de archivos gemspec (.gemspec), que son archivos YAML que contienen información sobre las gemas. Sin embargo, las gemas también pueden ser construidas por código Ruby directamente. Esta práctica se utiliza generalmente con rake.

*Comando Gem*

El comando gem se utiliza para crear, cargar, descargar e instalar paquetes de Gem. Algunos comandos son:

- ✓ Para la instalación y desinstalación: gem install mygem / gem uninstall mygem
- ✓ Para listar gemas instaladas: gem list –local ó gem list
- ✓ Para listar las gemas disponibles se usa: gem list --remote

### 2.4.4 RubyGems

RubyGems [11] es un gestor de paquetes que se convirtió en parte de la librería estándar de Ruby 1.9. Permite a los desarrolladores a buscar, instalar y construir gemas, entre otras características. Todo esto se realiza mediante la utilidad de la gema de línea de comandos.

RubyGems proporciona un formato estándar (llamado gem) para poder distribuir programas o librerías en Ruby, es una herramienta destinada a gestionar fácilmente la instalación de estas gemas.

#### *2.4.4.1 Características*

- ✓ Sencilla instalación y desinstalación de los paquetes y sus dependencias.
- ✓ Manejo y control de paquetes locales.
- ✓ Manejo de dependencias entre paquetes.
- ✓ Buscar y listar paquetes locales y remotos.
- ✓ Soporte para múltiples versiones de los paquetes instalados.
- ✓ Interfaz Web para consultar la documentación de las gemas instaladas.
- ✓ Interfaz sencilla para la construcción de paquetes.
- ✓ Servidor sencillo para la distribución de paquetes.

## 2.5 Servicios Web RESTful

La Transferencia de Estado Representacional [12] (REST - Representational State Transfer) fue ganando amplia adopción en toda la Web como una alternativa más simple a SOAP y a los servicios

Web basados en el Lenguaje de Descripción de Servicios Web (Web Services Descripcion Language - WSDL).

### 2.5.1 Presentando REST

REST [12] define un set de principios arquitectónicos por los cuales se diseñan servicios Web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la Web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante simple de usar.

REST no tuvo mucha atención cuando Roy Fielding lo presentó por primera vez en el año 2000 en la Universidad de California, durante la charla académica "Estilos de Arquitectura y el Diseño de Arquitecturas de Software basadas en Redes", la cual analizaba un conjunto de principios arquitectónicos de software para usar a la Web como una plataforma de Procesamiento Distribuido. Ahora, años después de su presentación, comienzan a aparecer varios frameworks REST.

### 2.5.2 Los cuatro principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- ✓ Utiliza los métodos HTTP de manera explícita.
- ✓ No mantiene estado.
- ✓ Expone URIs con forma de directorios.
- ✓ Transfiere XML, JavaScript Object Notation (JSON), o ambos.

A continuación se verá en detalle estos cuatro principios, y se explicará porqué son importantes a la hora de diseñar un servicio web REST.

✓ *REST utiliza los métodos HTTP de manera explícita*

Una de las características claves de los servicios Web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor Web, o ejecuten una consulta esperando que el servidor Web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- Se usa POST para crear un recurso en el servidor
- Se usa GET para obtener un recurso
- Se usa PUT para cambiar el estado de un recurso o actualizarlo
- Se usa DELETE para eliminar un recurso

Una falla de diseño poco afortunada que tienen muchas APIs Web es el uso de métodos HTTP para otros propósitos. Por ejemplo, la petición del URI en un pedido HTTP GET, en general identifica a un recurso específico. O el string de consulta en el URI incluye un conjunto de parámetros que definen el criterio de búsqueda que usará el servidor para encontrar un conjunto de recursos. Al menos, así como el RFC HTTP/1.1 describe al GET. Pero hay muchos casos de APIs Web poco elegantes que usan el método HTTP GET para ejecutar algo transaccional en el servidor; por ejemplo, agregar registros a una base de datos. En estos casos, no se utiliza adecuadamente el URI de la petición HTTP, o al menos no se usa "a la manera REST". Si el API Web utiliza GET para invocar un procedimiento remoto, seguramente se verá algo como esto:

```
GET/agregarusuario?nombre=Zim HTTP/1.1
```

Este no es un diseño muy atractivo, el método mostrado anteriormente expone una operación que cambia estado sobre un método HTTP GET. Dicho de otra manera, la petición HTTP GET que se presenta tiene efectos secundarios. Si se procesa con éxito, el resultado de la petición es agregar un usuario nuevo (en el ejemplo, Zim) a la base de datos.

El problema es básicamente semántico. Los servidores Web están diseñados para responder a las peticiones HTTP GET con la búsqueda de recursos que concuerden con la ruta (o el criterio de búsqueda) en el URI de la petición, y devolver estos resultados o una representación de los mismos en la respuesta, y no añadir un registro a la base de datos. Desde el punto de vista del protocolo, y desde el punto de vista de servidor web compatible con HTTP/1.1, este uso del GET es inconsistente.

Más allá de la semántica, el otro problema con el GET es que al ejecutar eliminaciones, modificaciones o creación de registros en la base de datos, o al cambiar el estado de los recursos de cualquier manera, provoca que las herramientas de caché Web y los motores de búsqueda (crawlers) puedan realizar cambios no intencionales en el servidor. Una forma simple de evitar este problema es mover los nombres y valores de los parámetros en la petición del URI a tags XML. Los tags resultantes, una representación en XML de la entidad a crear, pueden ser enviados en el cuerpo de un HTTP POST cuyo URI de petición es el padre de la entidad.

Antes*:*

```
GET/agregarusuario?nombre=Zim HTTP/1.1
```

Después:

```
POST/usuarios HTTP/1.1
Host: miservidor
Content-type: application/xml

<usuario>
      <nombre>Zim</nombre>
<usuario>
```

El método anterior  es un ejemplo de una petición REST,  hay un uso correcto de HTTP POST y la inclusión de los datos en el cuerpo de la petición. Al recibir esta petición, la misma puede ser procesada de manera de agregar el recurso contenido en el cuerpo como un subordinado del recurso identificado en el URI de la petición; en este caso el nuevo recurso debería agregarse como hijo de /usuarios. Esta relación de contención entre la nueva entidad y su padre, como se indica en la petición del POST, es análoga a la forma en la que está subordinado un archivo a su directorio.

El cliente indica esta relación entre la entidad y su padre y define el nuevo URI de la entidad en la petición del POST.

Luego, una aplicación cliente puede obtener una representación del recurso usando la nueva URI, sabiendo que al menos lógicamente el recurso se ubica bajo /usuarios

```
GET/usuarios/Zim  HTTP/1.1
Host: miservidor
Accept: application/xml
```

Es explícito el uso del GET de esta manera, ya que el GET se usa solamente para recuperar datos. GET es una operación que no debe tener efectos secundarios, una propiedad también conocida como idempotencia.

Se debe hacer un refactor similar de un método Web que realice actualizaciones a través del método HTTP GET. El siguiente método GET intenta cambiar la propiedad "nombre" de un recurso. Si bien se puede usar el string de consulta para esta operación, evidentemente no es el uso apropiado y tiende a ser problemático en operaciones más complejas. Ya que el objetivo es hacer uso explícito de los métodos HTTP, un enfoque REST sería enviar un HTTP PUT para actualizar el recurso, en vez de usar HTTP GET.

Antes:

```
GET/actualizarusuario?nombre=Zim&nuevoNombre=Dib HTTP/1.1
```

Después:

```
PUT/usuarios/Zim HTTP/1.1
Host: miservidor
Content-type: application/xml

<usuario>
      <nombre>Dib</nombre>
<usuario>
```

Al usar el método PUT para reemplazar al recurso original se logra una interfaz más limpia que es consistente con los principios de REST y con la definición de los métodos HTTP. La petición PUT que se muestra es explícita en el sentido que apunta al recurso a ser actualizado identificándolo en la URI de la petición, y también transfiere una nueva representación del recurso del cliente hacia el servidor en el cuerpo de la petición PUT, en vez de transferir los atributos del recurso como un conjunto de parámetros (nombre = valor) en la misma URI de la petición. El PUT mostrado también tiene el efecto de renombrar al recurso Zim a Dib, y al hacerlo cambia la URI a/usuarios/Dib. En un servicio web REST, las peticiones siguientes al recurso que apunten a la URI anterior van a generar un error estándar "404 Not Found".

Como un principio de diseño general, ayuda seguir las reglas de REST que aconsejan usar sustantivos en vez de verbos en las URIs. En los servicios Web REST, los verbos están claramente definidos por el mismo protocolo: POST, GET, PUT y DELETE. Idealmente, para mantener una interfaz general y para que los clientes puedan ser explícitos en las operaciones que invocan, los servicios Web no deberían definir más verbos o procedimientos remotos, como ser /agregarusuario y/actualizarusuario. Este principio de diseño también aplica para el cuerpo de la petición HTTP, el cual debe usarse para transferir el estado de un recurso, y no para llevar el nombre de un método remoto a ser invocado.

- ✓ *REST no mantiene estado*

Los servicios Web REST necesitan escalar para poder satisfacer una demanda en constante crecimiento. Se usan clusters de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways de manera de conformar una topología servicial, que permita transferir peticiones de un equipo a otro para disminuir el tiempo total de respuesta de una invocación al servicio Web. El uso de servidores intermedios para mejorar la escalabilidad hace necesario que los clientes de servicios Web REST envíen peticiones completas e independientes; es decir, se deben enviar peticiones que incluyan todos los datos necesarios para cumplir el pedido, de manera que los componentes en los servidores intermedios puedan redireccionar y gestionar la carga sin mantener el estado localmente entre las peticiones.

Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o cliente de servicio Web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios Web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

*Servicios con estado vs. Sin estado*

La siguiente figura muestra un servicio con estado, del cual una aplicación realiza peticiones para la página siguiente en un conjunto de resultados múlti-página, asumiendo que el servicio mantiene información sobre la última página que pidió el cliente. En un diseño con estado, el servicio incrementa y almacena en algún lugar una variable página Anterior para poder responder a las peticiones siguientes:

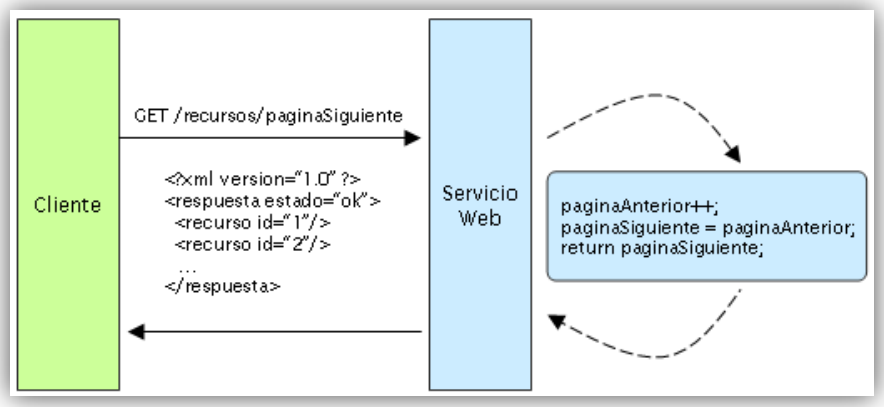



**Figura 2 - Servicio con Estado**

Los servicios con estado tienden a volverse complicados. En la plataforma Java Enterprise Edition (Java EE), un entorno de servicios con estado necesita bastante análisis y diseño desde el inicio para poder almacenar los datos eficientemente y poder sincronizar la sesión del cliente dentro de un cluster de servidores. En este tipo de ambientes, ocurre un problema que le resulta familiar a los desarrolladores de servlets/JSP y EJB, quienes a menudo tienen que revolver buscando la causa de una java.io.NotSerializableException cuando ocurre la replicación de una sesión.

Puede ocurrir tanto sea en el contenedor de Servlets al intentar replicar la HttpSession o por el contenedor de EJB al replicar un EJB con estado; en todos los casos, es un problema que puede costar mucho esfuerzo resolver, buscando el objeto que no implementa serializable dentro de un grafo complejo de objetos que constituyen el estado del servidor. Además, la sincronización de sesiones es costosa en procesamiento, lo que impacta negativamente en el rendimiento general del servidor.

Por otro lado, los servicios sin estado son mucho más simples de diseñar, escribir y distribuir a través de múltiples servidores. Un servicio sin estado no sólo funciona mejor, sino que además mueve la responsabilidad de mantener el estado al cliente de la aplicación. En un servicio Web REST, el servidor es responsable de generar las respuestas y proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta. Por ejemplo, en el mismo ejemplo de una petición de datos en múltiples páginas, el cliente debería incluir el número de página a recuperar en vez de pedir "la siguiente", tal como se muestra en la siguiente figura:

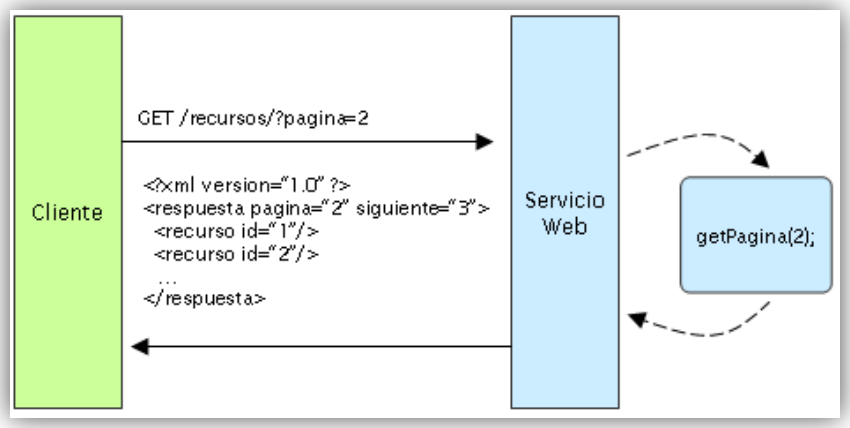



**Figura 3 - Servicio sin estado**

Un servicio Web sin estado genera una respuesta que se enlaza a la siguiente página del conjunto y le permite al cliente hacer todo lo que necesita para almacenar la página actual. Este aspecto del diseño de un servicio Web REST puede descomponerse en dos conjuntos de responsabilidades, como una separación de alto nivel que clarifica cómo puede mantenerse un servicio sin estado.

*Responsabilidad del servidor*

- Genera respuestas que incluyen enlaces a otros recursos para permitirle a la aplicación navegar entre los recursos relacionados. Este tipo de respuestas tiene enlaces embebidos. De la misma manera, si la petición es hacia un padre o un recurso contenedor, entonces una respuesta REST típica debería también incluir enlaces hacia los hijos del padre o los recursos subordinados, de manera que se mantengan conectados.
- Genera respuestas que indican si son susceptibles de caché o no, para mejorar el rendimiento al reducir la cantidad de peticiones para recursos duplicados, y para lograr eliminar algunas peticiones completamente. El servidor utiliza los atributos Cache-Control y Last-Modified de la cabecera en la respuesta HTTP para indicarlo.

*Responsabilidades del cliente de la aplicación*

- Utiliza el atributo Cache-Control del encabezado de la respuesta para determinar si debe cachear el recurso (es decir, hacer una copia local del mismo) o no. El cliente también lee el atributo Last-Modified y envía la fecha en el atributo If-Modified-Since del encabezado para preguntarle al servidor si el recurso cambió desde entonces. Esto se conoce como GET Condicional, y ambos encabezados van de la mano con la respuesta del servidor 304 (No Modificado) y se omite al recurso que se había solicitado si no hubo cambios desde esa fecha. Una respuesta HTTP 304 significa que el cliente puede seguir usando la copia local de manera segura, evitando así realizar las peticiones GET hasta tanto el recurso no cambie.
- Envía peticiones completas que pueden ser serviciadas en forma independiente a otras peticiones. Esto implica que el cliente hace uso completo de los encabezados HTTP tal como está especificado por la interfaz del servicio Web, y envía las representaciones del recurso en el cuerpo de la petición. El cliente envía peticiones que hacen muy pocas presunciones sobre las peticiones anteriores, la existencia de una sesión en el servidor, la capacidad del servidor para agregarle contexto a una petición, o sobre el estado de la aplicación que se mantiene entre las peticiones. Esta colaboración entre el cliente y el servicio es esencial para crear un servicio Web REST sin estado. Mejora el rendimiento, ya que ahorro ancho de banda y minimiza el estado de la aplicación en el servidor.

✓ *REST expone URIs con forma de directorios*

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el Web service REST, y si el servicio va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URI de los servicios Web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

Una forma de lograr este nivel de usabilidad es definir URIs con una estructura al estilo de los directorios. Este tipo de URIs es jerárquica, con una única ruta raíz, y va abriendo ramas a través de las subrutas para exponer las áreas principales del servicio. De acuerdo a esta definición, una URI no es solamente una cadena de caracteres delimitada por barras, sino más bien un árbol con subordinados y padres organizados como nodos. Por ejemplo, en un servicio de hilos de discusiones que tiene temas varios, se podría definir una estructura de URIs como esta:

```
http://www.miservicio.org/discusion/temas/{tema}
```

La raíz, /discusión, tiene un nodo /temas como hijo. Bajo este nodo hay un conjunto de nombres de temas (como ser tecnología, actualidad, y más), cada uno de los cuales apunta a un hilo de discusión. Dentro de esta estructura, resulta fácil recuperar hilos de discusión al tipiar algo después de /temas/.

En algunos casos, la ruta a un recurso encaja muy bien dentro de la idea de "estructura de directorios". Por ejemplo, en algunos recursos organizados por fecha, que son muy prácticos de organizar usando una sintaxis jerárquica.

Se puede también enumerar algunas guías generales más al momento de crear URIs para un servicio Web REST:

- Ocultar la tecnología usada en el servidor que aparecería como extensión de archivos (.jsp, .php, .asp), de manera de poder portar la solución a otra tecnología sin cambiar las URI.
- Mantener todo en minúsculas.
- Sustituir los espacios con guiones o guiones bajos (uno u otro).

- Evitar el uso de strings de consulta.
- En vez de usar un 404 Not Found si la petición es una URI parcial, devolver una página o un recurso predeterminado como respuesta.

Las URIs deberían ser estáticas de manera que cuando cambie el recurso o cambie la implementación del servicio, el enlace se mantenga igual. Esto permite que el cliente pueda generar "favoritos" o bookmarks. También es importante que la relación entre los recursos que está explícita en las URI se mantenga independiente de las relaciones que existen en el medio de almacenamiento del recurso.

✓ *REST transfiere XML, JSON, o ambos*

La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples "fotos" en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y tags XML, donde los valores de los elementos en el XML contienen los valores de las filas. O, si el sistema tiene un modelo de datos, la representación de un recurso es una fotografía de los atributos de una de las cosas en el modelo de datos del sistema. Estas son las cosas que serviciamos con servicios Web REST.

La última restricción al momento de diseñar un servicio Web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las peticiones/respuestas. Los objetos del modelo de datos generalmente se relacionan de alguna manera, y las relaciones entre los objetos del modelo de datos (los recursos) deben reflejarse en la forma en la que se representan al momento de transferir los datos al cliente. En el servicio de hilos de discusión anterior, un ejemplo de una representación de un recurso conectado podría ser un tema de discusión raíz con todos sus atributos, y links embebidos a las respuestas al tema.

```
<discusion fecha="{fecha}" tema="{tema}">
    <comentario>{comentario}</comentario>
    <respuestas>
          <respuesta de="
 gaz@mail.com" href="/discusion/temas/{tema}/gaz"/>
          <respuesta de="
 gir@mail.com" href="/discusion/temas/{tema}/gir"/>
    </respuestas>
</discusion>
```

Por último, es bueno construir los servicios de manera que usen el atributo HTTP Accept del encabezado, en donde el valor de este campo es tipo MIME. De esta manera, los clientes pueden pedir por un contenido en particular que mejor pueden analizar. Algunos de los tipos MIME más usados para los servicios Web REST son:

| MIME-Type | Content-Type |
|---|---|
| JSON | application/json |
| XML | application/xml |
| XHTML | application/xhtml+xml |

**Cuadro 1 - Tipos MIME más usados en REST**

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado *HTTP Accept* es un mecanismo conocido como negociación de contenido, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

## 2.6 ActiveResource

ActiveResource [13] es la capa responsable de la implementación de sistemas RESTful del lado del cliente. A través de ActiveResource es posible consumir servicios RESTful mediante el uso de objetos que actúan como un proxy para los servicios remotos. Ejemplo básico Modelo ActiveResource:

```
class Person < ActiveResource::Base
      self.site = "http://localhost:3000/"
end
```

ActiveResource proporciona la infraestructura necesaria para crear de manera sencilla recursos REST. Rails en sus últimas versiones usa ActiveResource desplazando así a otros modelos como SOAP y XML-RPC a los que se les daba soporte en versiones anteriores mediante Action Web Service.

ActiveResource [14] es la clase principal para la asignación de recursos REST como modelos en una aplicación Rails. Representan sus recursos REST como objetos manipulables. Para mapear recursos a objetos Ruby, ActiveResource sólo necesita el nombre de clase que se corresponde con el nombre del recurso (por ejemplo, la clase Person mapea al recurso people, de manera muy similar a Active Record) y un valor de un sitio, que es el URI de los recursos.

ActiveResorce expone los métodos para crear, buscar, actualizar y eliminar los recursos de los servicios Web REST. Los métodos ActiveResource son muy similares a los métodos de ActiveRecord en el ciclo de vida para los registros de la base de datos.

## 2.7 MongoDB

### 2.7.1 Introducción a las bases de datos NoSQL

Los sistemas de bases de datos se clasifican mayormente en tres tipos: las bases de datos relacionales, las orientadas a objetos, y las relacionales orientadas a objetos.

Sin embargo, en la práctica, la mayoría de los motores de bases de datos más populares se basan en la arquitectura relacional, y todos ellos utilizan el lenguaje de consultas SQL (con variaciones) para operar con los datos. Tanto es así, que SQL se convirtió con el paso de los años en un estándar "de facto", debido a su uso. [15]

### 2.7.2 Importancia de las bases de datos NoSQL

Las bases de datos relacionales no tienen nada de malo, precisamente gracias al transcurso de los años, se ha logrado aprender técnicas bastante comunes para normalizarlas en la medida de lo posible, escalarlas según crece la demanda, y utilizarlas como sistema de persistencia para almacenar información desde el lenguaje procedural u orientado a objetos favorito (entre otros). La cuota de uso de software como SQLite, MySQL, PostgreSQL u Oracle, por poner cuatro ejemplos conocidos, es muy alta, encontrándose en la mayor parte de los desarrollos modernos.

Pero llegó la Web, el software como servicio, los servicios en la nube y las startups de éxito con millones de usuarios. Y con todo ello llegaron los problemas de alta escalabilidad. Si bien los modelos relacionales se pueden adaptar para hacerlos escalar incluso en los entornos más difíciles, también es cierto que, a menudo, se hacen cada vez menos intuitivos a medida que aumenta la complejidad. Muchas veces se presentan triples y hasta cuádruples JOINs en consultas SQL que presentan en su mayoría incomprensibles, a veces poco eficientes, y sistemas de almacenamiento de resultados en cachés para acelerar la resolución de las peticiones y evitar ejecutar cada vez estas pesadas operaciones, son el pan de cada día en muchos de estos proyectos de software.

Los sistemas NoSQL intentan atacar este problema proponiendo una estructura de almacenamiento más versátil, aunque sea a costa de perder ciertas funcionalidades como las transacciones que engloban operaciones en más de una colección de datos, o la incapacidad de ejecutar el producto cartesiano de dos tablas (también llamado JOIN) teniendo que recurrir a la desnormalización de datos.

Algunas implementaciones NoSQL son: CouchDB, MongoDB, RavenDB, Neo4j, Cassandra, BigTable, Dynamo, Riak, Hadoop. En la aplicación del presente Trabajo de Grado se utilizó MongoBD, cuya estructura se describe a continuación:

### 2.7.3 MongoDB: Descripción y Licencia

MongoDB [15] es un sistema de base de datos multiplataforma orientado a documentos, de esquema libre, esto significa que cada entrada o registro puede tener un esquema de datos diferente, con atributos o "columnas" que no tienen por qué repetirse de un registro a otro. Está escrito en C++, lo que le confiere cierta cercanía a los recursos de hardware de la máquina, de modo que es bastante rápido a la hora de ejecutar sus tareas. Además, está licenciado como GNUAGPL 3.0, de modo que se trata de un software de licencia libre. Funciona en sistemas operativos Windows, Linux, OS X y Solaris.

Las características que más destacaría de MongoDB son su velocidad y su rico pero sencillo sistema de consulta de los contenidos de la base de datos. Se podría decir que alcanza un balance perfecto entre rendimiento y funcionalidad, incorporando muchos de los tipos de consulta que utilizaríamos en nuestro sistema relacional preferido, pero sin sacrificar en rendimiento.

### 2.7.4 Terminología básica en MongoDB

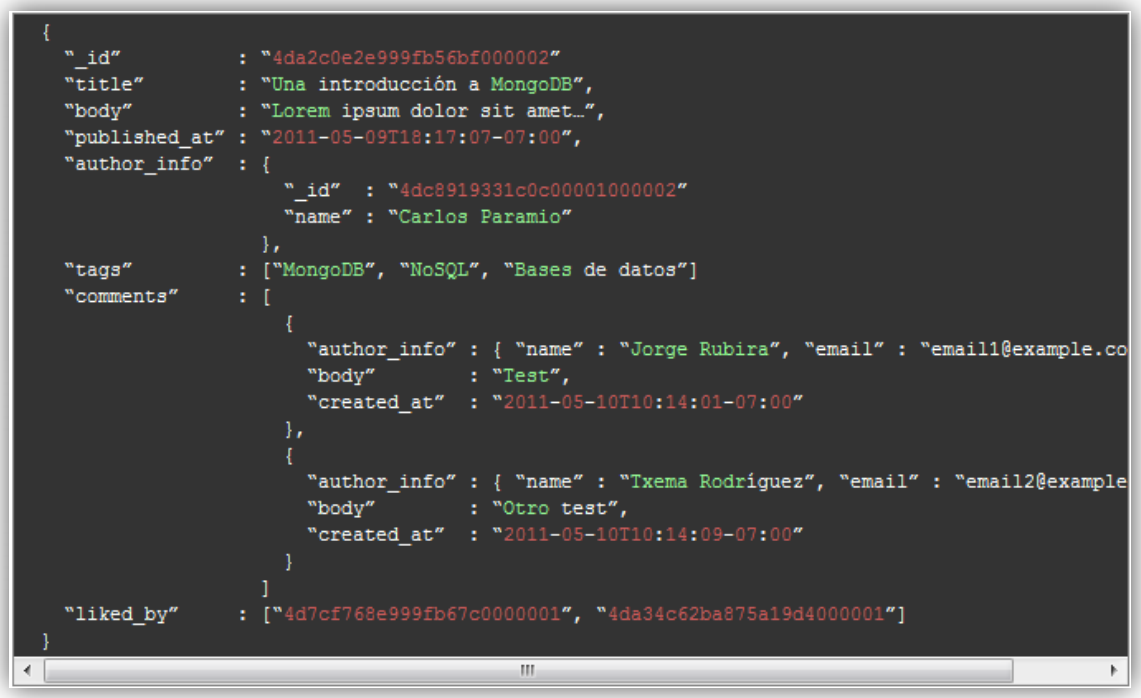
En MongoDB [15], cada registro o conjunto de datos se denomina documento. Los documentos se pueden agrupar en colecciones, las cuales se podría decir que son el equivalente a las tablas en una base de datos relacional (sólo que las colecciones pueden almacenar documentos con diferentes formatos, en lugar de estar sometidos a un esquema fijo). Se pueden crear índices para algunos atributos de los documentos, de modo que MongoDB mantendrá una estructura interna eficiente para el acceso a la información por los contenidos de estos atributos.

### 2.7.5 Formato de los documentos e ideas para la organización de datos

Los distintos documentos se almacenan en formato BSON, o Binary JSON, que es una versión modificada de JSON que permite búsquedas rápidas de datos. BSON guarda de forma explícita las longitudes de los campos, los índices de los arreglos, y demás información útil para el escaneo de datos.

Es por esto que, en algunos casos, el mismo documento en BSON ocupa un poco más de espacio de lo que ocuparía de estar almacenado directamente en formato JSON. Pero una de las ideas claves en los sistemas NoSQL es que el almacenamiento es barato, y es mejor aprovecharlo si así se introduce un considerable incremento en la velocidad de localización de información dentro de un documento.

Sin embargo, en la práctica, no se verá el formato en que verdaderamente se almacenan los datos, y se trabajará siempre sobre un documento en JSON tanto al almacenar como al consultar información. Un ejemplo de un documento en MongoDB podría ser perfectamente la figura 5.



```
{
  "_id"          : "4da2c0e2e999fb56bf000002"
  "title"        : "Una introducción a MongoDB",
  "body"         : "Lorem ipsum dolor sit amet…",
  "published_at" : "2011-05-09T18:17:07-07:00",
  "author_info"  : {
                     "_id"  : "4dc8919331c0c00001000002"
                     "name" : "Carlos Paramio"
                   },
  "tags"         : ["MongoDB", "NoSQL", "Bases de datos"]
  "comments"     : [
                     {
                       "author_info" : { "name" : "Jorge Rubira", "email" : "email1@example.co
                       "body"        : "Test",
                       "created_at"  : "2011-05-10T10:14:01-07:00"
                     },
                     {
                       "author_info" : { "name" : "Txema Rodríguez", "email" : "email2@example
                       "body"        : "Otro test",
                       "created_at"  : "2011-05-10T10:14:09-07:00"
                     }
                   ]
  "liked_by"     : ["4d7cf768e999fb67c0000001", "4da34c62ba875a19d4000001"]
}
```

**Figura 4 - Ejemplo Documento MongoDB**

Como se puede observar, este documento pretende representar la manera en que podrían almacenarse los datos correspondientes a un post de un blog. Los atributos "_id" (o clave principal) pueden tener el formato que se desee, aunque MongoDB utiliza un valor parecido a un UUID en hexadecimal por defecto si no se ha especificado ninguno.

A pesar de parecer un valor completamente aleatorio (aunque ya sabemos que la aleatoriedad real no existe en informática), utilizan como base una semilla basada en la MAC de la interfaz de red de la máquina (y otros detalles de la misma) para evitar que dos máquinas diferentes puedan generar el mismo valor para la clave de un documento. Y los primeros bytes corresponden a una marca de tiempo, de modo que las claves se ordenan de forma natural por orden de creación (o casi, pues está claro que las distintas máquinas corriendo MongoDB deben tener la fecha y hora sincronizadas) sin tener que mirar cuál fue el último valor usado. El atributo "_id" es el único obligatorio para un documento.

Las etiquetas y los comentarios están en el propio documento que representa al post, en lugar de guardarlos en colecciones separadas y utilizar claves foráneas para referenciar a los mismos. Sin embargo, en el atributo "liked_by" sí que guardamos una relación de claves, que corresponden a los usuarios que han marcado el post como que les ha gustado. Utilizar una forma u otra dependerá de las necesidades de acceso a estos datos. En este caso, por ejemplo, se sabe que no se va a extraer información sobre los usuarios que han marcado un post con un "me gusta", pero sí se quiere ver cuántos lo han marcado así, o si el usuario actual ya lo ha marcado o no, con lo que almacenar únicamente las claves de esos usuarios y guardar su información personal detallada en otra colección es lo más conveniente.

Por supuesto, no es necesario pedir a MongoDB que devuelva todo el documento cada vez que se consulte. Si por ejemplo se va a buscar únicamente un listado de posts recientes, seguramente sea suficiente obtener el atributo "title", con los documentos ordenados por "published_at". Así, se ahorra ancho de banda entre el motor de base de datos y la aplicación, al mismo tiempo que ahorra memoria dado que no hay que instanciar todo el documento. Además, si se tiene muchos miles de visitantes, el atributo "liked_by" podría llegar a crecer bastante. Aunque el tamaño de un documento de MongoDB puede llegar hasta los 16 Megabytes, con lo que podemos almacenar bastante información dentro de un único documento sin necesidad de utilizar referencias, si así se necesita. En caso de que se tuviera que almacenar mucho más, habría que optar por utilizar otro esquema.

A veces, toca desnormalizar para poder tener a mano la información necesaria a la hora de buscar un post. Es por eso que en el atributo "author_info" he utilizado una versión intermedia: Si bien se tiene la clave principal del usuario que ha escrito este post, como es habitual que se busque el nombre del autor, se ha almacenado también dicho nombre en el documento que representa al post, para que no sea necesario realizar una segunda consulta a la colección "usuarios".

Estas desnormalizaciones dependen nuevamente del uso que se den a los datos. En este caso, se tiene claro que el nombre de un usuario no va a cambiar demasiado, así que recorrer todos los posts para cambiar este valor en caso de que el usuario realice esta operación, si bien es una modificación que podría llevar un tiempo considerable para ejecutarse, no es una operación habitual frente a la consulta del nombre del autor, y por tanto compensa. Incluso se podría llegar a tratar el post como algo más permanente, de modo que aunque un usuario cambiara su nombre a posteriori, el nombre utilizado para firmar los posts anteriores no varíe, o sencillamente los posts puedan firmarse con diferentes pseudónimos.

El modelado del esquema de datos con MongoDB depende más de la forma en que se consulte o actualice los datos que de las limitaciones del propio sistema.

### 2.7.6 Cómo consultar los datos

En primer lugar, MongoDB permite utilizar funciones Map y Reduce escritas en Javascript para seleccionar los atributos que interesan de los datos, y agregarlos (unificarlos, simplificarlos) en la manera deseada, respectivamente. Esto es algo habitual en muchos sistemas NoSQL, y en algunos casos es incluso la única forma posible de consultar datos.

En MongoDB se pueden utilizar consultas al valor de un atributo específico. Por ejemplo, capturar el post que tiene un determinado título:

```
Db.post.find({‘title’ : ‘Una introducción a MongoDB’})
```

El valor a consultar puede estar anidado en un tipo de datos más completo en el atributo del documento (por ejemplo, como valor de un hash asociado al atributo, o como el valor de uno de los ítems de un arreglo). Se utiliza un punto como separador de los nombres de las claves de los diferentes hashes que hay que recorrer hasta llegar al valor deseado. Por ejemplo, la siguiente consulta devolvería todos los posts escritos por un determinado autor:

```
Db.posts.find({‘author_info_id’ : ‘4da2c0e2e999fb56bf000002’})
```

Y esta otra los posts etiquetados con MongoDB:

```
Db.posts.find({‘tags’ : ‘MongoDB’})
```

El hash utilizado como conjunto de condiciones que deben cumplir los documentos a devolver puede incluir operadores de muy diversos tipos, no sólo comparadores del valor absoluto buscado. Algunos de ellos son:

- ✓ $all: Para indicar que el array almacenado como valor del atributo debe tener los mismos elementos que el proporcionado en la condición.
- ✓ $exists: Para comprobar que el atributo existe en el documento.
- ✓ $mod: Para comprobar el resto de una división del valor del atributo por un número.
- ✓ $ne: Para indicar que el valor no puede ser el proporcionado.
- ✓ $in: Para indicar que el valor debe estar entre alguno de los proporcionados.
- ✓ $nin: Contrario de $in.
- ✓ $or: Para indicar que se debe cumplir al menos una condición de entre un grupo de condiciones.
- ✓ $nor: Contrario de $or.
- ✓ $size: Para indicar el número de elementos que debe haber en el array almacenado como valor.
- ✓ $type: Para comprobar el tipo del valor almacenado (número, cadena…)
- ✓ Expresiones regulares: El valor debe concordar con la expresión regular indicada.

Estos son sólo algunos operadores, los resultados se pueden agrupar, ordenar, contar, paginar, y otras tantas operaciones comunes sin necesidad de recurrir al confuso Map / Reduce. Y siempre que los atributos consultados tengan definidos un índice, la velocidad de las consultas es espectacular.

## 2.8 Cloud Computing

Cloud Computing es un paradigma donde el objetivo es proveer servicios por un tercero a través de internet, de allí la metáfora de la Cloud (“nube”), dichos servicios son provistos en cuanto sean requeridos en ambientes distribuidos, disponibles, con menor complejidad y a gran escala. Cloud

Computing proporciona herramientas de software, plataformas e infraestructura como servicios, a saber, SaaS, PaaS, IaaS. Esta sección se enfoca en describir Cloud Computing como una red servicios, teniendo en cuenta aspectos como el diseño, los modelos de red y los protocolos de comunicación. La infraestructura física se implementa generalmente a través de Datacenters que según el tipo podrían ser Public o Private Cloud, también se utiliza en gran parte la virtualización debido a la escalabilidad y disponibilidad que presenta. Además de estos beneficios la virtualización provee una propiedad importante que es el "aislamiento", sin embargo, no hay que dejar atrás aspectos por un lado como la seguridad y por otro lado la definición de estándares para interconectar Clouds. Hay dos enfoques derivados de Cloud Computing, uno es el Cloud Networking que forma parte de la base de infraestructura, también está el enfoque Inter Cloud que básicamente trata temas relacionados a la interconexión de Clouds.

### 2.8.1 ¿Qué es Cloud Computing?

Cloud Computing [16] se refiere por una parte a las aplicaciones entregadas como servicios sobre internet y por otra parte al hardware en Datacenters que proveen estos servicios. También se menciona que Cloud Computing [17] es la convergencia y evolución de muchos conceptos de virtualización, de aplicaciones distribuidas, de Grids que habilitan un enfoque flexible para el despliegue y del escalado de aplicaciones [18]. Cloud Computing es un término utilizado para describir una plataforma y el tipo de aplicación [19], siendo una plataforma que dinámicamente aprovisiona, configura y reconfigura servidores como sea necesario.

### 2.8.2 Evolución de Cloud Computing

El término Cloud Computing no es nuevo, ni revolucionario, sino que es un término que ha venido evolucionando con el tiempo [20]. Empezó en los años 1980's bajo conceptos de Grid Computing, aunque con ciertas diferencias y enfatizado a servidores virtuales; luego en los años 1990's se expandió el concepto de virtualización elevando el nivel de abstracción de los servidores virtuales, primero como plataforma virtual y luego como aplicaciones virtuales; más adelante se conoció el término Utility Computing, que ofrece clusters como plataformas virtuales; recientemente el término software como servicio (SaaS) elevando el nivel de virtualización a las aplicaciones, con un modelo de negocio no recargado en recursos consumidos.

Debido a esta evolución el concepto de Cloud Computing combina los términos anteriores de Grid, Utility y SaaS, siendo un modelo emergente en donde los usuarios pueden tener acceso a las aplicaciones desde cualquier lugar a través de dispositivos conectados.

### 2.8.3 Beneficios y riesgos

En [21] se mencionan los siguientes beneficios de Cloud Computing:

- ✓ Reduce el costo total de propiedad: debido a que en toda organización ese es uno de los principales objetivos, minimización de costos en hardware y software.
- ✓ Incrementa la escalabilidad y fiabilidad: debido que a través de diferentes infraestructuras se puede habilitar lo escalable y fiable que pueden llegar a ser los sistemas bajo conceptos de Cloud Computing.
- ✓ Además, existen varios beneficios que también puede actuar como riesgos dependiendo si la implementación se realizó bajo los pasos sugeridos, entre estos se pueden mencionar la disponibilidad, el rendimiento, la capacidad, etc.

Por otra parte en [22] se mencionan beneficios referentes al desarrollo y despliegue:

- ✓ Reduce el tiempo de ejecución y el tiempo de respuesta.
- ✓ Minimiza el riesgo de adquirir nueva infraestructura.
- ✓ Menor costo de inversión inicial.
- ✓ Aumenta el ritmo de la innovación.

### 2.8.4 Implementación

Se mencionan tres escenarios para la formación de Cloud Computing [23] [24], a saber, Private Cloud, Public Cloud e Hybrid Cloud. Estos escenarios se han convertido en medios atractivos para el intercambio computacional, de almacenamiento y de recursos de red entre desarrolladores de servicios múltiples y de aplicaciones de prestación de servicios. No hay que olvidar la capacidad de reasignar dinámicamente los recursos utilizando tecnologías de virtualización, ayudando a mitigar la necesidad de inversiones adicionales en infraestructura en tiempos de alta demanda.

Estos tres escenarios son implementados en los Datacenters y su disposición hace que su categorización dependa del enfoque, ya sea interno, externo o combinado.

#### *2.8.4.1 Private Cloud*

Private Clouds [23] son escenarios donde las compañías realizan sus operaciones fuera de línea, ejecutando aplicaciones seguras en Datacenters. De modo que, este escenario también se puede llamar Internal Cloud.

Internal Cloud [24] aplica los conceptos de Cloud Computing a recursos propios de la empresa que consume el servicio, proveyendo la capacidad de manejar aplicaciones web nuevas y existentes, mientras se provee de seguridad y regulación. También se menciona que Private Cloud trae consigo ciertas ventajas:

- ✓ Disponible en demanda.
- ✓ Rápido aprovisionamiento de servicios de negocio.
- ✓ Reducción del costo a través de economías a escala.
- ✓ Flexibilidad y libertad de selección,
- ✓ Basado en el uso.
- ✓ Controlado y asegurado por corporación de IT.

#### *2.8.4.2 Public Cloud*

Public Clouds [23] son escenarios donde las compañías necesitan mover datos o aplicaciones desde su interior al exterior, ambos escenarios utilizan la misma arquitectura, con la diferencia que el escenario público se conecta con otros escenarios. De modo que, este escenario también se puede llamar External Cloud.

External Cloud involucra recursos y servicios IT que son vendidos [24], tales como auto servicio, aprovisionamiento en demanda y pago por utilización, todos estos servicios accedidos a través de navegadores web o a través de API's.

Bajo el escenario de External Cloud se ejecutan diferentes tipos servicios que se detallarán en la siguiente sección, que son software, plataformas e infraestructura como servicios.

#### *2.8.4.3 Hybrid Cloud*

Por último, se hace referencia a una mezcla entre los dos escenarios anteriores, llamada Hybrid Cloud, llamado también Inter Cloud [23], que es un tipo de escenario semi público, el cual se comporta como un Private Cloud con la particularidad que ciertas empresas pueden compartir su información con ciertos niveles de permiso, por ello el término semipúblico.

El control de Public Cloud lo hace el proveedor, mientras que el control de Private Cloud lo hace la empresa, y la finalidad es que a través de ambos enfoques satisfacer las necesidades de un sistema de aplicación. Inter Cloud como escenario [24], brindaría la capacidad de elegir los proveedores de servicio, y los proveedores de servicio federados serían capaces de compartir las cargas de servicio, siendo una relación más flexible.

### 2.8.5 Niveles de servicio

Los diferentes niveles de servicio que componen a Cloud Computing son IaaS, PaaS y SaaS. Sin embargo [26], se menciona otro nivel de servicio, aunque ciertos autores lo prefieren alojar en entre los mismo tres niveles de servicio y no crear uno nuevo, es el llamado ITaaS, IT como servicio, siendo un modelo de servicio donde una organización o individuo contrata con un proveedor de servicios para obtener conectividad de red y cualquier otro servicio incluido, como backup de red, recuperación de desastres, VPN, conferencias web, etc. Aunque este nivel es muy general y puede abarcar a los tres anteriores pero de una forma unificada.

#### *2.8.5.1 IaaS*

Infrastructure as a Service o infraestructura como servicio [17] es entregar tanto hardware como software como un servicio. El ejemplo más común es el hosting, el cual, nos provee de hardware como un servidor y de software como un webserver, sin embargo, este concepto ha evolucionado a infraestructura como EC2 y S3.

Otra manera de ver IaaS [27] es como la manera de compartir recursos para ejecutar servicios, típicamente utilizando tecnología de virtualización, en donde múltiples usuarios utilizan dichos recursos. Los recursos pueden fácilmente escalar cuando la demanda se incrementa, y generalmente se utiliza métodos como pago por uso.

Uno de los servicios que toma importancia concerniente a la infraestructura es el Cloud Storage que se menciona a continuación:

*Cloud Storage*

Es almacenamiento localizado ya sea en Datacenters públicos o privados, separados del almacenamiento primario. Pertenece al nivel de infraestructura como servicio, la manera de implementación es a través [28] del Service Oriented Architecture (SOA), y la localización tiene diferentes variaciones, Cloud Storage puede ubicarse en:

- ✓ Un Datacenter público,
- ✓ Un Datacenter privado, o
- ✓ Separado del almacenamiento primario.

La manera de acceso puede ser de dos formas: Directamente como bloques o archivos; o indirectamente a través de aplicaciones que están ubicadas en el mismo lugar del almacenamiento. Hay dos tecnologías [29] que proporcionan métodos de almacenamiento: La primera es Storage Area Network (SAN) que son switches de redes de alta velocidad que permiten que múltiples computadoras tengan acceso compartido a varios dispositivos de almacenamiento; y la segunda es Network-Attached Storage (NAS) que vienen como aplicaciones NAS o Gateways NAS, son servidores de archivos virtuales que tienen soporte a protocolos como NFS, siendo un dispositivo que directamente concede a la red y que tiene capacidades de compartir archivos.

Los protocolos utilizados para Cloud Storage son SOAP o REST. También existen ciertas barreras para su adopción, tales como, el ahorro de costos no es significante, se pone en riesgo la privacidad de los datos, cuestiones de migración, disponibilidad de datos y contratos como SLAs.

#### *2.8.5.2 PaaS*

Platform as a Service o plataforma como servicio [17], es entregar una plataforma de desarrollo de aplicaciones como un servicio para desarrolladores en la web. Generalmente se provee de herramientas tipo middleware, por ejemplo, Google AppEngine. Además de dicha entrega, también se ofrece un ambiente de ejecución como el servidor de aplicaciones.

#### *2.8.5.3 SaaS*

Software as a Service o software como servicio [17] provee la administración y hosting de aplicaciones con sus propios Datacenters, se maneja el término de múltiples inquilinos, por ejemplo Oracle CRM On Demand o Salesforce.

Es interesante lo que dice [30] acerca de SaaS, lo compara con la controversia que generó la computación cliente-servidor para reemplazar la computación de mainframes, y para el usuario final SaaS es un simple concepto, el usuario solamente ingresa a una aplicación a través del navegador web sin saber en donde se aloja o como está siendo servida, solamente sabe que es rápida y segura.

## 2.9 Workling

Workling [30] provee a las aplicaciones Rails una API la cual permite ejecutar códigos en segundo plano, y además en paralelo con la ejecución de la aplicación.

Workling permite realizar la configuración más adecuada dependiendo de las exigencias de la aplicación que lo invoca, por consecuencia, workling es compatible con Starling, BackgroundJob y Spawn Runners. Se puede definir Workling en el esquema de Rails como el Active* para las ejecuciones en segundo plano: esto es, poder escribir el código una sola vez para instanciarlo cualquier cantidad de veces que se desee mediante cualquier background Runner en cualquier otro momento. Esto permite conservar un ambiente flexible.

### 2.9.1 Instalación de Workling

La manera más sencilla de instalar Workling para empezar a desarrollar es colocando las siguientes instrucciones en el terminal:

```
script/plugin install git://github.com/purzelrakete/workling.git
script/plugin install git://github.com/tra/spawn.git
```

En caso de tener instalado una versión antigua e incompatible de Rails, existe un mirror en subversión workling del cual se puede descargar en:

```
script/plugin install http://svn.playtype.net/plugins/workling
```

### 2.9.2 Desarrollo y llamadas a Workers

Este procedimiento no tiene mucha dificultad, sólo es necesario colocar un archivo cow_worker.rb en la carpeta app/workers de la aplicación de Rails, y la subclase Workling::Base:

```
# handle asynchronous mooing.
class CowWorker < Workling::Base
  def moo(options)
    cow = Cow.find(options[:id])
    logger.info("about to moo.")
    cow.moo
  end
end
```

Es necesario tomar en cuenta que se debe tener exactamente un parámetro *hash* en el método invocador, *workling* pasa el *job :uid* utilizando esta vía. Luego, de la preparación, es necesario hacer la llamada de *workling* desde el *controller*, con lo cual quedaría de la siguiente manera:

```
class CowsController < ApplicationController

  # milking has the side effect of causing
  # the cow to moo. we don't want to
  # wait for this while milking, though,
  # it would be a terrible waste ouf our time.
  def milk
    @cow = Cow.find(params[:id])
    CowWorker.asynch_moo(:id => @cow.id)
  end
end
```

**Figura 5 - Llamada de workling desde el controller**

De esta manera se notifica al *asynch_moo* la llamada a *CowWorker*. Esto realiza la llamada al método *moo* del *CowWorker* en segundo plano, dando la oportunidad de pasar cualquier cantidad de parámetros que se desee, de hecho, workling puede realizar la llamada de cualquier método que provenga después de la instrucción *asynch_* comoun método de la instancia del worker.

### 2.9.3 Ciclo de vida del Worker

Todas las clases que se deriven deben heredar de esta misma, y ser almacenadas en la carpeta *app/worker*. El Worker es cargado una sola vez, hasta el punto en que el método de la instancia *create* es invocado.

### 2.9.4 Excepciones manejadas en los Workers

Si una excepción se presenta durante la ejecución de un Worker, esta no será atendida por workling al código invocador. Esto se debe a que el código es invocado de manera asíncrona, es decir, la excepción se produjo después de que el código invocador ha sido ejecutado. Si se necesita que el código invocador realice un manejo de excepciones, se debe manejar pasando el error y haciendo un catch del mismo.

Workling almacena en una bitácora todas las excepciones que propagan en tiempo de ejecución fuera de los métodos del worker.

### 2.9.5 Ingreso de usuario

RAILS_DEFAULT_LOGGER está disponible en todos los workers. Además tiene un método logger el cual retorna un logger por defecto, así es posible ingresar como usuario de esta manera:

```
logger.info("about to moo.")
```

## 2.10 Starling

Starling es un ligero manejador de colas, el cual opera mediante el uso de un sistema distribuido de propósito general para caché basado en memoria llamado MemCached. Fue desarrollado por el equipo de desarrollo de Twitter para el procesamiento de códigos en segundo plano, y puesto a disposición como software libre.

Para instalar Starling se coloca en el terminal las siguientes instrucciones:

```
sudo gem install starling
sudo gem install memcache-client
```

Después de tener instalado Starling, es necesario conseguir la manera de utilizar este servidor de colas para tareas en segundo plano, y es aquí donde entra Workling, presentado en el punto anterior. Workling automáticamente detecta y dispone de starling, esto ocurre siempre y cuando no se encuentre instalado otro sistema de colas como por ejemplo Spawn, si esto es así, es necesario indicar al Workling el uso especifico de Starling, eso se realiza editando el archivo *environment.rb* de la aplicación Rails con la siguiente instrucción:

```
Workling::Remote.dispatcher = Workling::Remote::Runners::StarlingRunner.new
```

### 2.10.1 Inicio de los programas requeridos

Workling copia un archivo llamado workling.yml en el directorio de configuración de la aplicación Rails que lo invoca. El archivo de configuración le informa a Workling qué puerto está utilizando Starling para esperar las peticiones. Sabiendo eso, el puerto por defecto es 15151. Esto conlleva a que Starling debe iniciarse con –p 15151 en ambiente de producción.

Además se puede utilizar este archivo de configuración para extender las opciones de configuración al cliente *memcache* el cual workling utiliza para realizar la conexión con starling. Se tiene también la opción de editar el tiempo para cada Worker.

```
development:
  listens_on: localhost:22122
  sleep_time: 2
  reset_time: 30
  listeners:
    Util:
      sleep_time: 20
  memcache_options:
    namespace: myapp_development

production:
  listens_on: localhost:22122, localhost:221223, localhost:221224
  sleep_time: 2
  reset_time: 30
```

**Figura 6 - Workling.yml**

Se pueden agrupar las instancias de Starling, listando sus valores separados por comas. *Sleep time* determina el tiempo de espera entre revisión y revisión, una simple revisión puede invocar un .get en cada cola (existe una correspondencia cola por cada método worker).

### 2.10.2 Observando que está haciendo Starling

Starling está desarrollado con su propio script, si se desea estadísticas específicas de workling se debe ejecutar:

```
script/starling_status.rb
```

### 2.10.3 Generando una nueva tarea brokers a Workling

Existen dos nuevas clases base de las cuales se pueden extender para incluir nuevos *brokers*. A continuación se presenta como se realiza esto usando *amqp* como un ejemplo. Este código se presenta como parte de workling.

Client brinda apoyo a workling para conectar las tareas del *brokers.* Para incluir un *AmpqClient,* se necesita además la implementación de un par de métodos.

```
require 'workling/clients/base'
require 'mq'

#
#  An Ampq client
#
module Workling
  module Clients
    class AmqpClient < Workling::Clients::Base

      # starts the client.
      def connect
        @amq = MQ.new
      end

      # stops the client.
      def close
        @amq.close
      end

      # request work
      def request(queue, value)
        @amq.queue(queue).publish(value)
      end

      # retrieve work
      def retrieve(queue)
        @amq.queue(queue)
      end

      # subscribe to a queue
      def subscribe(queue)
        @amq.queue(queue).subscribe do |value|
          yield value
        end
      end

    end
  end
end
```

**Figura 7 - Módulo Workling usando Amap**

*Invocadores:*

Workling trae incorporado algunos de los invokers estándares, como lo es BasicPoller. Este invocador mantiene llamando al bróker casi en segundos, buscando de alguna nueva tarea y ejecutándola de inmediato.

Seria agradable si se puede tener un invocador que realice la subscripción de las *callbacks*. De manera sencilla se observa el esquema de lo antes mencionado:

```
require 'eventmachine'
require 'workling/remote/invokers/base'

#
#  Subscribes the workers to the correct queues.
#
module Workling
  module Remote
    module Invokers
      class EventmachineSubscriber < Workling::Remote::Invokers::Base

        def initialize(routing, client_class)
          super
        end

        #
        #  Starts EM loop and sets up subscription callbacks for workers.
        #
        def listen
          EM.run do
            connect do
              routes.each do |queue|
                @client.subscribe(queue) do |args|
                  run(queue, args)
                end
              end
            end
          end
        end

        def stop
          EM.stop if EM.reactor_running?
        end
      end
    end
  end
end
```

**Figura 8 - Invocadores de Workling**

El invoker implementa dos  métodos: lister y stop, lister comienza el ciclo principal de espera de peticiones, el cual es responsable de las tareas de starling mientras ésta se coloca en un estado disponible.

# CAPITULO 3 - MARCO METODOLÓGICO

En este capítulo se describe el método utilizado para el desarrollo del módulo controlador del prototipo de archivo digitales Web. Esta elección se hace como respuesta a una posible solución a los problemas derivados de múltiples cambios en los requerimientos generados durante la definición de los alcances. El método XP se adapta correctamente a este entorno ya que la agilidad se define (entre otras cosas) por la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino.

A continuación se especifican los aspectos más resaltantes del método XP o también llamada Programación Extrema (XP) y su adaptación para el desarrollo del sistema.

## 3.1 Método XP

El método XP es un método ágil, iterativo e incremental el cual se utiliza para desarrollar software en grupos de pequeños programadores, donde la codificación es la actividad primordial sobre la documentación exhaustiva [31]

Mediante este método se libera rápidamente a producción un sistema sencillo y de igual manera, se liberan continuamente nuevas versiones en periodos cortos. Tanto los tutores de proyecto, los clientes y programadores, son parte del equipo y están involucrados en el desarrollo del software.

A continuación se destacan las características principales de este método de trabajo [32]:

- ✓ Planificación incremental.
- ✓ Programación en parejas.
- ✓ Propiedad colectiva del código.
- ✓ Comunicación constante con el cliente.
- ✓ Desarrollo guiado en pruebas.
- ✓ Integración continúa.
- ✓ Estándares de codificación.

- ✓ Refactorización de código.

## 3.2 Adaptación del Proceso de Desarrollo XP

A continuación se exponen los elementos relacionado a la adaptación del proceso XP que se utilizará durante el desarrollo del módulo controlador del prototipo de Archivos de Internet.

*Iteraciones*

Las iteraciones simbolizan los cambios incrementales generados a través de las pruebas y retroalimentaciones repetidas que a futuro dan como resultado un sistema estable pero en evolución. Las iteraciones pueden ser de dos tipos principalmente: por objetivo o por lapsos de tiempo. En nuestro caso las iteraciones se realizaran por objetivo, en donde, cada iteración comprende la puesta en marcha de un requerimiento funcional o no funcional, un evento o una necesidad.

*Historias de Usuarios*

Las historias de usuario son un elemento primordial en el desarrollo y planificación dentro del método XP, permite establecer un vínculo comunicacional entre el cliente y los miembros del equipo.

Ayuda a priorizar y equilibrar las necesidades con la finalidad de mejorar la toma de decisiones en cuanto a que se debe desarrollar primero. En lo que a nuestro caso se refiere, se trabajaran en función del tiempo (utilizando los días como unidad de medición) y con el formato de: un número que servirá de identificador, un nombre, el tipo (nueva o modificación/mejora), una estimación del tiempo y una breve descripción sobre la historia de usuario. El formato es el que a continuación se muestra:

| | | |
|---|---|---|
| Número: - | Nombre: - | |
| Tipo: - | | Tiempo Estimado: - |
| Descripción: - | | |

**Cuadro 2 - Formato de registro para una Historia de Usuario**

*Actores y Responsabilidades*

Los actores son todas las personas involucradas en el desarrollo del proyecto, los cuales a su vez cumplen distintos roles o responsabilidades según su importancia y nivel de participación. A continuación se destacan los roles existentes en el presente proceso de desarrollo:

- ✓ Programador: Es el pilar fundamental del desarrollo en XP, tiene grandes habilidades en cuanto a la comunicación y al desarrollo en equipo. Adicionalmente, tiene la capacidad de poder abordar de forma simple y sencilla problemas complejos.
- ✓ Cliente: Es el encargado de proveer las historias de usuario, realizar las pruebas de aceptación, requisitos funcionales y no funcionales deseables en la aplicación y la toma de decisiones acertadas sobre las características esenciales de la aplicación.
- ✓ Probador: Su función se centra en realizar las pruebas de integración al sistema del código provisto por los programadores y de verificar el correcto funcionamiento de la aplicación. También realiza pruebas regulares y da mantenimiento siempre sustentando los resultados con informes precisos.
- ✓ Rastreador: Se encarga de dar seguimiento al proceso general del grupo, calculando el tiempo que toman sus tareas y el progreso general a las metas que se quieren alcanzar. Realiza estimaciones de tiempo y da la retroalimentación al equipo con el fin de mejorar el rendimiento.

| | Programador | Cliente | Probador | Rastreador |
|---|---|---|---|---|
| Cindy De Almada | X | | X | |
| Frederick Chai | X | | X | |
| Andrés Sanoja | | X | | X |

**Cuadro 3 - Esquema de actores y roles**

*Actividades de XP*

El método XP está compuesto por cuatro actividades fundamentales las cuales están contenidas en cada una de las iteraciones del proceso de desarrollo. A continuación una breve descripción de cómo será la adaptación de cada una de ellas:

- Grafico: Se elabora un gráfico el cual representa el comportamiento del desarrollo de implementación de las historias de usuarios que conforman una iteración dada.

- Planificación: La actividad de planificación comienza creando una serie de historias de usuario que describen las funcionalidades y actividades requeridas por el cliente, proporcionando a su vez una estimación del tiempo necesario para el desarrollo.

  En principio se adapta el entorno de programación adecuado para desarrollar aplicaciones en Ruby on Rails y conexión a otros modelos utilizando el esquema REST, (por sus siglas en inglés Representational State Transfer - Transferencia de Estado Representacional) que se llevó a cabo al inicio del desarrollo, del cual surgirá una lista de historias de usuarios que serán organizadas y desarrolladas en un conjunto de iteraciones.

  Se utilizará un esquema (cuadro 4) al inicio de cada iteración el cual contendrá el número de la iteración, una descripción, el número, nombre y tipo de las historias de usuarios a desarrollar, la fecha de inicio y la fecha de fin de la iteración.

| Iteración - | | |
|---|---|---|
| Descripción | - | |
| Fecha Inicio / Fecha Fin | - | |
| Número | Historia | Tipo |
| - | - | - |

**Cuadro 4 - Esquema de planificación de cada iteración**

- Diseño: El diseño en XP sigue de forma rigurosa el principio de simplicidad, prefiriendo siempre un diseño simple respecto de una presentación más compleja. Además el diseño debe ofrecer una guía de implementación para una historia de usuario determinada.

  Basados en las prácticas XP, en cada iteración de la presente etapa se realizarán prototipos mostrando las interfaces a desarrollar que permitan mejorar la comprensión de las historias planteadas.

- Codificación: Este método sugiere la programación en pareja, la cual consiste en que dos programadores trabajen juntos en una estación de trabajo al momento de crear el código de una historia de usuario, siguiendo en todo momento los estándares de programación, lo cual es otro aspecto de gran importancia en el método XP.

  También recomienda realizar frecuentes integraciones de código entre los grupos de trabajo, de tal forma que no se produzcan problemas de compatibilidad.

  En este sentido se adoptará la programación en pareja para desarrollar el código de las principales historias de usuario y siempre trabajando en una misma iteración. Aquellas historias de usuario que no sean de mayor complejidad y no representen una funcionalidad primordial en el sistema serán desarrolladas de forma individual por un programador y luego integrada a dicho sistema.

  En esta sección se realizará la instalación y configuraciones del ambiente que sean necesarias, además de toda la codificación de las historias de usuario de cada una de las iteraciones. La integración del código se realizará de forma continua a través del sistema de control de versiones (subversión).

- Pruebas: El método XP establece realizar pruebas de todo aquello que se codifique, recomienda no dejar ninguna característica del sistema sin que haya sido probada. Dicho método propone pruebas unitarias (unit test), pruebas del programador (programmer test) y pruebas de aceptación (customer test), estas últimas son especificadas por el cliente y se enfocan en las características generales y la funcionalidad del sistema, elementos visibles y revisables por el cliente [32]

Con respecto a este punto se realizarán pruebas del programador y pruebas de aceptación. Para las pruebas del programador se empleara una técnica simple que consistirá en evaluar parámetros de entrada seleccionado por los programadores y observar las salidas constatando que cumplan con lo esperado. Las pruebas de aceptación serán realizadas por los clientes, con el fin de comprobar que sus requerimientos hayan sido cumplidos satisfactoriamente. Además se utilizará un formato para registrar cada una de las pruebas que se realicen. Ver cuadro 5

| **No.** | **H.U. involucradas** | **Descripción del Caso de Prueba** | **Resultado Esperado** | **Resultado Obtenido** |
|---|---|---|---|---|
| - | - | - | - | - |

**Cuadro 5 - Formato de registro de Prueba de Aceptación**

Por lo general, en cada iteración se desarrollarán cada una de las actividades mencionadas anteriormente, sin embargo en las primeras iteraciones puede darse el caso de que no se desarrollen todas las actividades, debido a las tareas que involucran determinadas historias de usuario durante una iteración.

Antes de pasar al siguiente capítulo y ver el listado de las historias de usuario, es importante acotar que debido a la necesidad de añadir un nuevo comportamiento a la aplicación que pueda generar las salidas adecuadas para obtener los resultados esperados, se desarrollaron a partir de la historia de usuario nro. 20, una serie de historias generadas como consecuencia de refactorizaciones ante la necesidad de cumplir con nuevos criterios como lo son:

- Reestructurar el esquema utilizado para obtener mejores prestaciones en el procesamiento de datos.
- Adición de nuevas tecnologías no consideradas a principio del proyecto.
- Adición de nuevas funcionalidades no consideradas a principio del proyecto.

Es importante acotar que esta adaptación en las historias de usuarios son consecuencia de la generación de la historia nro. 20.

# CAPITULO 4 - MARCO APLICATIVO

## 4.1 Historias de Usuario

A continuación se especifican las historias de usuario obtenidas:

| | |
|---|---|
| **Número: 1** | **Nombre:** Instalar el ambiente de ejecución en las máquinas de trabajo personales |
| **Tipo:** Nueva | **Tiempo Estimado:** 5 días |
| **Descripción:** Instalar todas las herramientas necesarias para el correcto funcionamiento del ambiente de ejecución en las máquinas de trabajo personal. Estas herramientas son: Ruby, Ruby on Rails, RubyGems, Mysql. | |

| | |
|---|---|
| **Número: 2** | **Nombre:** Generar traductor en Ruby que tenga como entrada un proceso BPEL |
| **Tipo:** Nueva | **Tiempo Estimado:** 8 días |
| **Descripción:** Generar un archivo que lea un proceso BPEL, guarde datos en la base de datos y obtenga como salida un archivo que va a ser el correspondiente al proceso BPEL pero en lenguaje Ruby. | |

| **Número: 3** | **Nombre:** Crear base de datos que contenga tabla de símbolos |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Crear un base de datos que contenga una tabla de símbolos con las variables a ser usadas en el archivo de salida Ruby. Los datos de esta tabla deben ser temporales, además de otra tabla que contenga las direcciones que envía el proceso BPEL. | |

| **Número: 4** | **Nombre:** Realizar un generador que haga la traducción de BPEL a Ruby y cree recursos |
|---|---|
| **Tipo:** Modificación / Mejora | **Tiempo Estimado:** 10 días |
| **Descripción:** Realizar un generador que lea un proceso BPEL, cree un archivo traductor Ruby y recursos por cada URL que haya en el proceso BPEL y los mande a ejecutar en una aplicación Rails que contendrá todos los recursos agrupados en procesos. | |

| **Número: 5** | **Nombre:** Realizar un modelo ActiveRecord que establezca la conexión con la base de datos |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Realizar un modelo ActiveRecord que establezca la conexión con la base de datos utilizada en la aplicación Ruby y defina las tablas como clases para ser usadas en el generador y en la base de datos. | |

| **Número: 6** | **Nombre:** Crear una tabla proceso en la base de datos de la aplicación |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 2 días |
| **Descripción:** Crear una tabla proceso en la base de datos ya creada que guarde el nombre del proceso leído y las URLs asociadas a él. | |

| **Número: 7** | **Nombre:** Crear una tabla en la base de datos que guarde el proceso de entrada BPEL |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 2 días |
| **Descripción:** Crear una base de datos que contenga una tabla en donde se guarde el proceso BPEL y su correspondiente generado en Ruby como backup del sistema. | |

| **Número: 8** | **Nombre:** Realizar una aplicación Rails cliente orientada a recursos que haga consultas hacia una aplicación servidor |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 5 días |
| **Descripción:** Generar aplicación cliente que realice consultas (crear, modificar, mostrar y borrar procesos y direcciones) hacia una aplicación servidor. La comunicación entre las aplicaciones es con ActiveResource. | |

| **Número: 9** | **Nombre:** Realizar una aplicación en Rails servidor orientada a recursos que atienda todas las peticiones que genera la aplicación cliente |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 5 días |
| **Descripción:** Generar aplicación servidor que atienda todas las peticiones que genera la aplicación cliente (crear, modificar, mostrar y borrar). La comunicación entre las aplicaciones se realiza con ActiveResource. | |

| **Número: 10** | **Nombre:** Realizar una aplicación spider en Rails orientada a recursos que reciba y envíe datos desde y hacia el servidor. |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 5 días |
| **Descripción:** Generar aplicación spider que realice peticiones al servidor y atienda las peticiones que el servidor envíe. | |

| **Número: 11** | **Nombre:** Crear una base de datos en el lado del servidor |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 3 días |
| **Descripción:** Crear una base de datos en el servidor el cual será utilizado por la aplicación cliente y el spider. El servidor debe ser capaz de comunicarse con el spider para enviarle las direcciones que fueron creadas por la aplicación cliente, y éste se las asigne a un spider (servidor) para que sean ejecutadas. El spider debe ser capaz de modificar el estado de la dirección en la base de datos. | |

| **Número: 12** | **Nombre:** Crear una base de datos en el spider |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 3 días |
| **Descripción:** Crear una base de datos en el spider el cual será utilizado por el servidor para enviar las direcciones que llegan del cliente y enviárselas al spider para procesarlas. | |

| **Número: 13** | **Nombre:** Modificar base de datos que contenga todos los procesos, direcciones y spider disponibles. |
|---|---|
| **Tipo:** Modificación / Mejora | **Tiempo Estimado:** 2 días |
| **Descripción:** Modificar base de datos del servidor para que contenga información de los procesos, direcciones, spider disponibles y los posibles estados que puede tener una dirección. Crear las relaciones entre las entidades. | |

| **Número: 14** | **Nombre:** Generar una tabla log en la base de datos |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 2 días |
| **Descripción:** Generar un tabla log en el servidor que contenga todas las eventualidades ocurridas durante el ciclo de vida del proyecto. | |

| **Número: 15** | **Nombre:** Desarrollar código en el spider que actualice el estado de las direcciones en la base de datos |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar código en el spider el cual debe estar actualizando el estado de una dirección en la base de datos cuando ha tenido un cambio en el proceso de ejecución (ejecutando, pausa, error, finalizado). | |

| **Número: 16** | **Nombre:** Desarrollar código del lado del servidor que verifique la base de datos. |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar código del lado del servidor el cual debe estar revisando constantemente la base de datos para ver si se crean nuevas URLs, asignarles un spider a cada URL y enviárselas al spider para que las mande a ejecutar. | |

| **Número: 17** | **Nombre:** Realizar validaciones en el cliente |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar código del lado del cliente que valide los datos a ser enviados como entrada al servidor | |

| **Número: 18** | **Nombre:** Realizar validaciones en el servidor |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar código del lado del servidor que valide las entradas al servidor para poder procesar con éxito la salida. | |

| **Número: 19** | **Nombre:** Realizar validaciones en el spider |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar código del lado del spider que valide las entradas y salidas desde y hacia el servidor. | |

## 4.2 Plan de Iteración

El ciclo de vida del método XP se enfatiza en el carácter interactivo e incremental del desarrollo, una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas que en el caso de XP corresponden a un conjunto de historias de usuarios. [33]

Las iteraciones en este proyecto son relativamente cortas ya que XP define que entre más rápido se entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye planificación, diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo.

El proyecto y su desarrollo, comprende un conjunto de 11 (once) iteraciones, estimadas en un tiempo de cumplimiento de 2 a 4 semanas cada una y su culminación implica la puesta en marcha de un requerimiento funcional, no funcional, evento o necesidad. El periodo para llevar a cabo dicho proyecto y su documentación correspondiente fue desde el 13 de septiembre de 2010 hasta el 20 de mayo de 2011.

Es importante recalcar que estas 11 (once) iteraciones comprenden dos fases importantes en lo que fue el desarrollo de este proyecto, podemos definir la primera fase como *Prueba de Concepto* en la cual se llevo a cabo el proceso de definir el entorno adecuado para cumplir con las exigencias que el proyecto requería, durante ésta fase se hicieron muchas modificaciones de arquitecturas y criterios hasta llegar a la orientación a recursos y utilización de la "nube" (cloud computing) la cual definimos como *Puesta Punto del Prototipo***,** segunda fase definida por las iteraciones.

Por esto, se puede definir que las iteraciones desde la primera hasta la octava forman parte de la fase *Prueba de Concepto***,** el resto pertenece a la fase de *Puesta Punto del Prototipo***.**

A continuación se presentan las iteraciones que conforman el desarrollo del proyecto:

### 4.2.1 ITERACIÓN 0:

- Planificación

Como primera iteración es necesario acondicionar el entorno de trabajo instalando el lenguaje de programación a utilizar y todas las dependencias necesarias para llevar a cabo las primeras pruebas y luego continuar con las fases iniciales de programación del proyecto a desarrollar.

| **Iteración 0** | | |
|---|---|---|
| **Descripción** | Instalación y configuración del entorno de desarrollo de la Aplicación | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 1 | Instalar el ambiente de ejecución en las máquinas de trabajo personales. | Nueva |

- Descripción

Durante esta primera iteración se llevó a cabo el proceso de instalación del framework de Ruby, además de las múltiples gemas necesarias como por ejemplo Rails, ActiveRecord, etc., necesarias para que el sistema funcionara en las máquinas de trabajo personal, además de un conjunto con las dependencias necesarias. Es importante destacar que dicha instalación fue realizada en dos puestos de trabajo, bajo dos ambientes distintos, Linux y Windows, de manera de poder preservar y verificar la compatibilidad en estos dos ambientes.

Una vez instalado todo el ambiente de desarrollo, se realizó la búsqueda de la versión estable del framework de Ruby compatible con las versiones de las gemas a utilizar, a continuación se lista las versiones de las gemas utilizadas (Cuadro 6):

| | |
|---|---|
| actionmailer (2.3.9) | MIME-types (1.15) |
| actionpack (2.3.9) | mongrel (1.1.5 x86-mingw32) |
| activerecord (2.3.9) | mysql (2.8.1 x86-mingw32) |
| activeresource (2.3.9) | rack (1.1.2) |
| activesupport (2.3.9) | rails (2.3.9) |
| cgi_multipart_eof_fix (2.5.0) | rake (0.8.7) |
| gem_plugin (0.2.3) | rest-client (1.6.1) |
| json (1.4.6 x86-mingw32) | |

**Cuadro 6 - Gemas instaladas en máquinas personales**

Para el ambiente en Windows, se descargó el ejecutable para la instalación de Ruby, la cual anexa el editor Scite, en donde se desarrollaron los códigos en Ruby, y para el ambiente en Linux se descargó el framework de Ruby y se utilizó para editar los códigos el gedit que se encuentra instalado en el sistema operativo.

- Pruebas

Las pruebas que se realizaron durante esta iteración fueron la escritura de códigos capaces de imprimir por pantalla datos, manipular datos de entrada, verificación de conexiones con base de datos y comunicación entre modelos, además de verificar la correcta instalación de todos los componentes y gemas de manera que se puedan embeber sus funcionalidades.

### 4.2.2 ITERACIÓN 1:

- Gráfico:

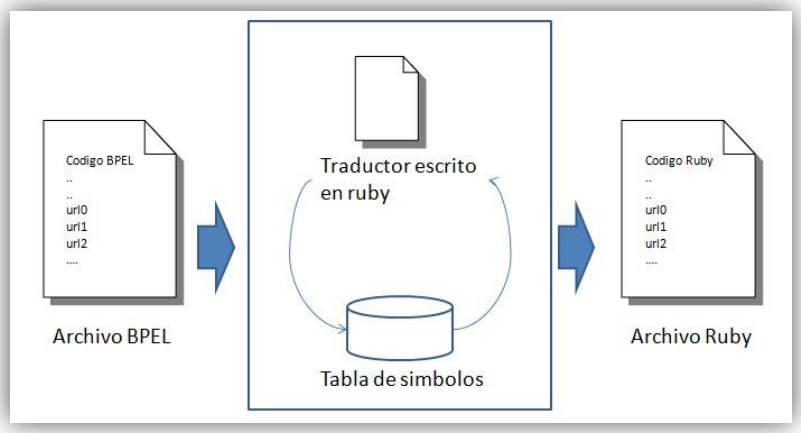



**Figura 9 - gráfico descriptivo de la iteración 1**

- Planificación

Después de tener instalado el entorno de programación adecuado, se empieza el desarrollo de una aplicación capaz de traducir código BPEL a Ruby, esto es debido a que los procesos se encuentran desarrollados en lenguaje BPEL, y como requisitos del proyecto se quiere una aplicación escrita en Ruby además de generar las estructuras necesarias para su ejecución, por ejemplo, la base de datos.

| **Iteración 1** | | |
|---|---|---|
| **Descripción** | Creación de traductor Ruby que tiene como entrada un proceso BPEL y creación de base de datos para el traductor. | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 2 | Generar traductor en Ruby que tenga como | Nueva |

| | | |
|---|---|---|
| | entrada un proceso BPEL | |
| 3 | Crear una base de datos que contenga una tabla de símbolos, y una tabla URLs | Nueva |

- Diseño

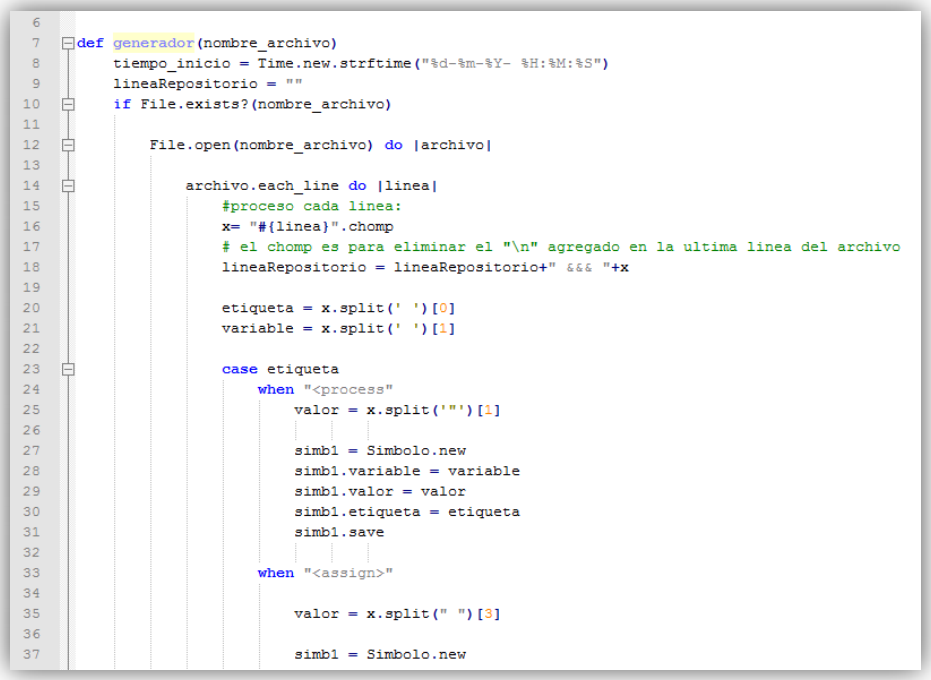
  El desarrollo del traductor de Ruby consistió en crear un archivo que lea el proceso de entrada BPEL línea por línea buscando la información necesaria para la tabla de símbolos de la base de datos con lo cual se va a crear el archivo traductor.

  El administrador de la aplicación cliente crea el proceso agregando las URLs necesarias para él, y lo manda a ejecutar. El traductor recibe esta entrada, guarda las variables y las URLs agregadas al proceso en su base de datos y comienza a crear el archivo de salida. El archivo BPEL tendrá un formato preestablecido, es decir, que el modelo del mismo no cambiará, solo en el nombre del proceso, las URLs de entrada, cantidad de URLs, etc.

  El archivo traductor de salida que se genera es un archivo, que al igual que el proceso BPEL tiene un formato preestablecido.

- Codificación

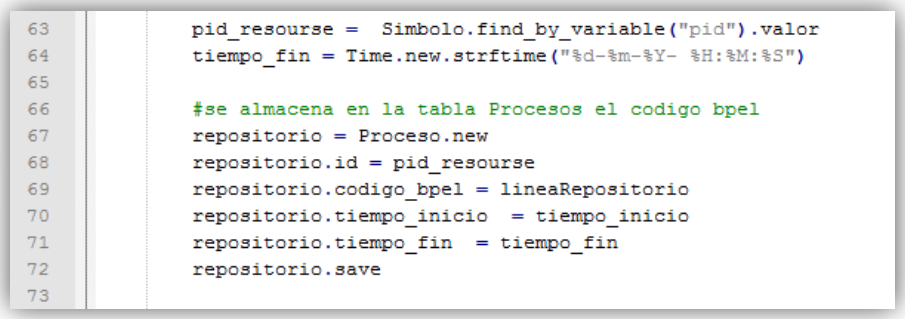
  Inicialmente se desarrolló un método cuya responsabilidad es leer el proceso BPEL de entrada y obtener los datos necesarios para la generación del archivo traductor. Además de buscar el nombre del proceso para asignárselo como nombre al archivo de salida. Ver Figura 10.



```
def generador(nombre_archivo)
    tiempo_inicio = Time.new.strftime("%d-%m-%Y- %H:%M:%S")
    lineaRepositorio = ""
    if File.exists?(nombre_archivo)

        File.open(nombre_archivo) do |archivo|

            archivo.each_line do |linea|
                #proceso cada linea:
                x= "#{linea}".chomp
                # el chomp es para eliminar el "\n" agregado en la ultima linea del archivo
                lineaRepositorio = lineaRepositorio+" &&& "+x

                etiqueta = x.split(' ')[0]
                variable = x.split(' ')[1]

                case etiqueta
                    when "<process"
                        valor = x.split('"')[1]

                        simb1 = Simbolo.new
                        simb1.variable = variable
                        simb1.valor = valor
                        simb1.etiqueta = etiqueta
                        simb1.save

                    when "<assign>"

                        valor = x.split(" ")[3]

                        simb1 = Simbolo.new
```

**Figura 10 - Código que lee archivo de entrada y almacena datos**

Los datos que obtiene el proceso BPEL son almacenados en la base de datos, para luego implementarlos en el archivo traductor y así obtener la salida deseada para que luego sea procesada por otros módulos. En la figura 11 se muestra como se guarda los datos en la base de datos con ActiveRecord.



```
        pid_resourse =  Simbolo.find_by_variable("pid").valor
        tiempo_fin = Time.new.strftime("%d-%m-%Y- %H:%M:%S")

        #se almacena en la tabla Procesos el codigo bpel
        repositorio = Proceso.new
        repositorio.id = pid_resourse
        repositorio.codigo_bpel = lineaRepositorio
        repositorio.tiempo_inicio  = tiempo_inicio
        repositorio.tiempo_fin  = tiempo_fin
        repositorio.save
```

**Figura 11 - Código que guarda los datos en la base de datos**

Los datos de la base de datos son temporales, sólo existen mientras el proceso existe, es decir, mientras se procede a crear el archivo traductor los datos persisten, luego de obtener la salida esos datos son eliminados de la base de datos, para que se proceda a realizar el proceso y llenado de la base de datos con otro proceso de entrada.

- Pruebas

Las pruebas realizadas fueron de tipo funcional, verificando que se leyera correctamente el proceso de entrada BPEL, que se almacenara de forma correcta en la base de datos los datos requeridos y que la salida fuera la esperada. A continuación las pruebas de aceptación que se realizaron.

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 1 | 2 | Proporcionar la entrada necesaria para generar el traductor codificado en Ruby | El archivo deberá crear un traductor que corresponda con la entrada del proceso BPEL pero con el lenguaje Ruby | El archivo lee correctamente los datos de entrada y genera su correspondiente traductor de salida |
| 2 | 3 | Proporcionar datos de entrada para que se guarden en la BD | Los datos que se leen del proceso BPEL se guardan correctamente y luego de procesar y generar el traductor borra los datos de la tabla | Los datos se guardan correctamente y luego de procesar el traductor se borra correctamente de la BD |

### 4.2.3 ITERACIÓN 2:

- Gráfico

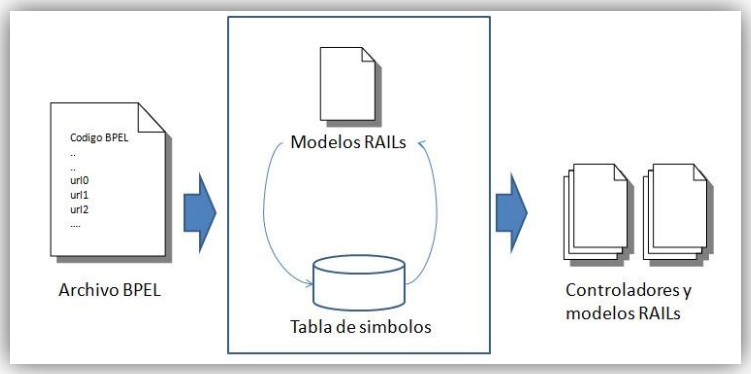



**Figura 12 - Gráfico descriptivo de la iteración 2**

- Planificación

Después de estudiar el comportamiento de la ejecución de la aplicación, se decide modificar el traductor e incorporar un generador de recursos, debido a que Rails se ventaja bajo este esquema, por lo tanto, se cambia el esquema a uno orientado a recursos.

| **Iteración 2** | | |
|---|---|---|
| **Descripción** | Modificación y desarrollo de un traductor que reciba como entrada un proceso BPEL y genere como salida un archivo Ruby y tantos recursos como URLs hayan en el proceso BPEL | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 4 | Realizar un generador que haga la traducción de BPEL a Ruby y cree recursos | Modificación/ Mejora |

| | | |
|---|---|---|
| 5 | Realizar un modelo ActiveRecord que establezca la conexión con la base de datos | Nueva |
| 6 | Crear una tabla proceso en la base de datos de la aplicación | Nueva |
| 7 | Crear una tabla en la base de datos que guarde el proceso de entrada BPEL | Nueva |

- Diseño

Durante ésta iteración se desarrolló una aplicación en Ruby que tiene como entrada un proceso BPEL del cual se extraen datos como el nombre del proceso y las URLs asociadas a este proceso. Luego de esto, se crea una base de datos en la cual se almacenan los datos extraídos del archivo de entrada BPEL. A continuación para generar una estructura asociada a un proceso Rails, por cada una de las URLs presentes en el proceso se crea un recurso como modelo de Rails, de esta manera se va a crear un esqueleto de lo que sería un proyecto Rails a partir de un archivo de entrada escrito en BPEL.

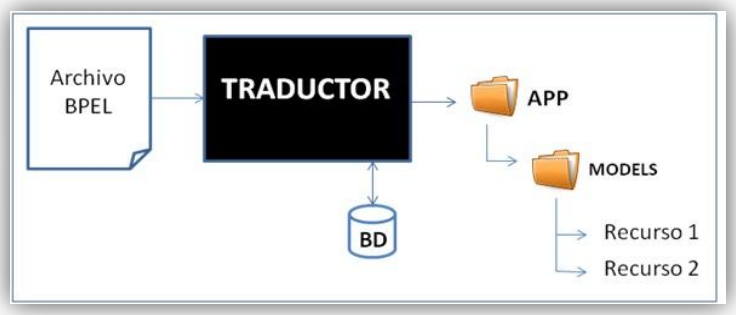



**Figura 13 - Diseño traductor Ruby**

Con el generador sólo se crea el directorio app/models donde se guardarán los recursos creados por la aplicación, los otros directorios necesarios para completar la estructura de una aplicación Rails no se crean ya que éstos se encuentran generados al crear la aplicación Rails en donde se procederá a ejecutar los recursos agrupados por proceso.

- Codificación

La idea principal de desarrollar esta aplicación Traductor es poder generar a partir de un archivo BPEL su equivalente a Ruby, a continuación un ejemplo del esquema del archivo de entrada (Figura 14):

```
<process name = "sitios_varios">
<variable name="uri_count"/>
<variable name="bot_code"/>
<variable name="pid"/>
<variable name="uri_list"/>
<variable name="operation"/>
<variable name="respuesta"/>
<variable name="k"/>
<variable name="href"/>

<resource name = "bot">
    <onPost>
        <sequence>
            <assign> uri_count = 5 </assign>
            <assign> pid = $request.pid </assign> <!-- Por ejemplo: pid == 21-->
            <assign> uri_list[1] = 'http://www.ciens.ucv.ve/' </assign>
            <assign> uri_list[2] = 'http://www.ucv.ve/' </assign>
            <assign> uri_list[3] = 'http://www.noticias24.com/' </assign>
            <assign> uri_list[4] = 'http://www.gobiernoenlinea.ve/' </assign>
            <assign> uri_list[5] = 'http://www.ultimasnoticias.com.ve/' </assign>
            <while>
                <condition> k <= uri_cont </condition>
                    <assign>bot_code = "bot" + pid + "r" + k</assign>
                    <resource name=bot_code>
                        <onPost>
                            <sequence>
                                <post uri="http://iticven.org/bots" request="url_list[k]" response="respuesta"/>
                            </sequence>
                        </onPost>
                        <onGet>
                            <sequence>
                                <assign>href = "http://iticven.org/bots/"+bot_code</assign>
                                <get uri=href response="respuesta"/>
                                <if>
                                    <condition>respuesta == "200"</condition> <!-- termino el robot code=200-->
                                    <then>
                                        <response code="200">
```

**Figura 14 - Modelo proceso entrada BPEL**

Como se puede observar, el archivo BPEL contiene una arreglo *uri_list* el cual almacena las URLs asociadas a un proceso, la cantidad de elementos de este arreglo se encuentra definido por la variable *uri_count*, además el nombre del proceso al que pertenecen todas éstas URLs se encuentra asignado a la variable *resource name*, luego se componen los nombres de cada archivo recurso que representan los modelos de la aplicación Rails que se quiere generar, estos nombres de cada modelo están compuestos por la palabra "bot" concatenado con un id único para cada proceso definido por la variable *pid*, concatenado con la letra "r" que define que se trata de un recurso y la variable *k* que define la numeración de cada recurso en el pool de recursos por proceso. De esta manera cada archivo modelo generado tiene un nombre *bot21r1.rb*, el cual define un recurso de un proceso llamado *bot21* y almacena la URLs identificada por el numero 1.

Este nombre de archivo es pasado por parámetro al modulo *bot_impl* el cual se encarga de crear el archivo modelo y su conexión con el lado del servidor:

```
def bot_impl(nombre_recurso)
  nuevo_arch = File.new("app/models/"+nombre_recurso+".rb", "w")
  nuevo_arch.puts("class BotImpl < ActiveResource::Base
   self.site = 'http://iticven.org/'
end")

end
```

**Figura 15 - Módulo encargado de crear archivo modelo**

Luego de conocer el archivo de entrada a la aplicación de traductor, y conocer la ubicación de los datos necesarios para la generación de los modelos, el traductor, escrito en Ruby, se encarga de hacer una lectura de cada una de las líneas del archivo BPEL dentro de un ciclo con el cual se hace una búsqueda de las etiquetas *<process>*, *<assign> y <resourse>*, y al obtener coincidencias se almacenan en la base de datos en la tabla de símbolos.

```
x= "#{linea}".chomp
etiqueta = x.split(' ')[0]
variable = x.split(' ')[1]
case etiqueta
      when "<process"
            valor = x.split('"')[1]

            simb1 = Simbolo.new
            simb1.variable = variable
            simb1.valor = valor
            simb1.etiqueta = etiqueta
            simb1.save

      when "<assign>"

            valor = x.split(" ")[3]

            simb1 = Simbolo.new
            simb1.variable = variable
            simb1.valor = valor
            simb1.etiqueta = etiqueta
            simb1.save

      when "<resource"
            valor = x.split('"')[1]

            simb1 = Simbolo.new
            simb1.variable = variable
            simb1.valor = valor
            simb1.etiqueta = etiqueta
            simb1.save

      when "<while>"
            break
```

**Figura 16 - Almacenamiento de variables en la base de datos**

Para el almacenamiento en la base de datos se utiliza el esquema Rails el cual consiste en la utilización de modelos que se encargan de conectarse a la base de datos, presentando de esta manera el esquema orientado a recursos, en este caso para conectarse a la tabla *Símbolos* se tiene un modelo llamado símbolo el cual sirve como conector, la estructura se presenta a continuación:

```
require "rubygems"
require "active_record"

ActiveRecord::Base::establish_connection(
  :adapter=>"mysql",
  :host=>"localhost",
  :database=>"tesis",
  :user=>"root")

class Simbolo < ActiveRecord::Base

  # Sort based on artist name.
  def <=>(other)
    name <=> other.name
  end
end
```

**Figura 17 - Modelo Active Record para conexión con tabla Símbolos**

La tabla de Símbolos almacena todas las variables que utilizan las etiquetas *<assign>*, *<process> y <resourse>* de manera de poder resolver todas aquellas asignaciones de variables entre sí, además de almacenar las URLs y el nombre del proceso. A continuación se presenta la tabla Símbolos:

```
mysql> describe simbolos;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| etiqueta | varchar(50) | YES  |     | NULL    |                |
| variable | varchar(50) | YES  |     | NULL    |                |
| valor    | varchar(50) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

**Figura 18 - Estructura Tabla Símbolos**

Por último, mientras se recorre el archivo BPEL para la extracción de los datos, cada una de las líneas se va concatenando de manera de tener almacenada en una variable todo el texto que constituye el archivo BPEL y es almacenado en la tabla *procesos.* (Figura 19)

```
mysql> describe procesos;
+---------------+----------+------+-----+---------+-------+
| Field         | Type     | Null | Key | Default | Extra |
+---------------+----------+------+-----+---------+-------+
| id            | int(11)  | NO   | PRI | NULL    |       |
| codigo_bpel   | longtext | NO   |     | NULL    |       |
| tiempo_inicio | datetime | NO   |     | NULL    |       |
| tiempo_fin    | datetime | NO   |     | NULL    |       |
+---------------+----------+------+-----+---------+-------+
```

**Figura 19 - Estructura Tabla Procesos**

- Pruebas

Para realizar las pruebas funcionales de la aplicación, se tomó como archivo de entrada el proceso BPEL de la figura 9, esperando como resultado la creación de un proyecto Rails que contenga los directorios y recursos creados en el proceso de la traducción. Los resultados fueron los siguientes:

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 3 | 6, 7 | Proporcionar archivo de entrada para que sea leído y guarde datos en la BD | El traductor debe leer línea por línea el archivo de entrada y guardar los datos necesarios en la BD | Se lee el archivo de entrada y se guardan los datos en la base de datos de forma correcta. |
| | 4, 5, 6, 7 | Proporcionar el archivo de entrada necesaria | El traductor deberá generar tantos recursos como URLs tenga el proceso de entrada y | El traductor lee de forma correcta el archivo de entrada y |

| | | | | |
|---|---|---|---|---|
| 4 | | para generar el traductor y los recursos asociados al proceso. | guardar los datos necesarios en la base de datos de la aplicación | genera los recursos necesarios por proceso, además de crear los directorios de Rails y guardar los datos correctamente en la base de datos. |

### 4.2.4 ITERACIÓN 3:

- Gráfico

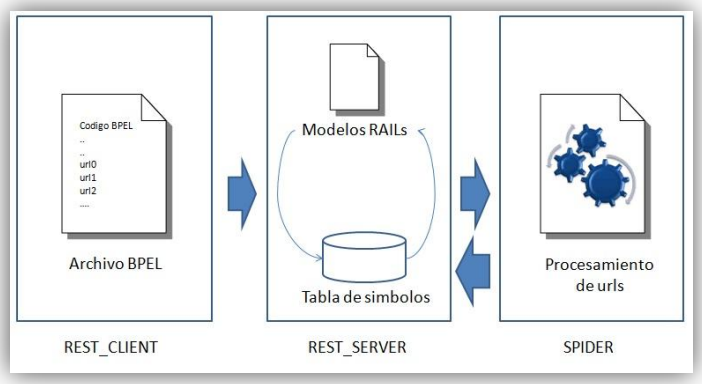



**Figura 20 - Gráfico descriptivo iteración 3**

- Planificación

En esta iteración se busca definir tareas en la aplicación y su distribución en tres módulos orientados al esquema cliente-servidor, además incorporar el spider el cual se encarga de procesar las URL generadas por el cliente, de esta manera el servidor (módulo de control que recibe entrada del cliente, la procesa y almacena en base de datos) es visto como centro de procesamiento y validación de datos, todo esto orientado a recursos y manteniendo ese esquema para la comunicación entre módulos.

| **Iteración 3** | | |
|---|---|---|
| **Descripción** | Creación de los módulos en Rails necesarios y conexión entre ellos. | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |

| | | |
|---|---|---|
| 8 | Realizar una aplicación Rails cliente orientada a recursos que haga consultas hacia una aplicación servidor | Nueva |
| 9 | Realizar una aplicación en Rails (llamada servidor) orientada a recursos que atienda todas las peticiones que genera la aplicación cliente | Nueva |
| 10 | Realizar una aplicación spider en Rails orientada a recursos que reciba y envíe datos desde y hacia el servidor. | Nueva |

- Diseño

  En la presente iteración, se realiza una serie de cambios de enfoque importantes con respecto a la iteración anterior, en primera instancia se cambia el enfoque de realizar un traductor y se canalizan las ideas en realizar una estructura divida en tres facetas:

  - Cliente: interfaz a la cual tiene acceso el panel de control (la cual manipula el usuario), y que se encarga de realizar las peticiones de crear, obtener, actualizar y eliminar los procesos y URLs asociados.
  - Servidor: interfaz en la que se reciben todas estas peticiones del cliente, se verifica y se procesan, de manera que cualquier eventualidad queda documentado en un registro almacenado en la base de datos, y finalmente a cada proceso le es asignado un spider para su ejecución.
  - Spider: módulo alojado en un servidor externo en el cual, dado cada proceso se inspecciona las URL asociadas, se descarga su contenido, se cataloga y retorna una respuesta al cliente del resultado del procedimiento.

- Codificación:

  La codificación de esta iteración está centrada en una programación orientada a recursos, de ésta manera se garantiza la comunicación y compatibilidad entre los módulos.

  Los módulos involucrados en esta aplicación son: el cliente (rest_client), el servidor (rest_server) y el spider (rest_spider).

Esta aplicación está desarrollada conforme al framework de Ruby on Rails, por lo tanto cada uno de los módulos sigue el paradigma de la arquitectura Modelo - Vista - Controlador (MVC), durante el desarrollo, cada uno de estos modelos son generados usando el método de andamiaje denominado *scaffold*, el cual permite al compilador generar la estructura necesaria para obtener las funcionalidades básicas de datos de un modelo en un controlador, estas funciones son las CRUD (Create, Retrieve, Update, Delete), típicas de cualquier sistema transaccional, es importante destacar que sólo el módulo de rest_client y rest_server, fueron generadas con esta técnica.

A continuación se presentan la estructura de cada uno de los módulos relacionados con la aplicación:

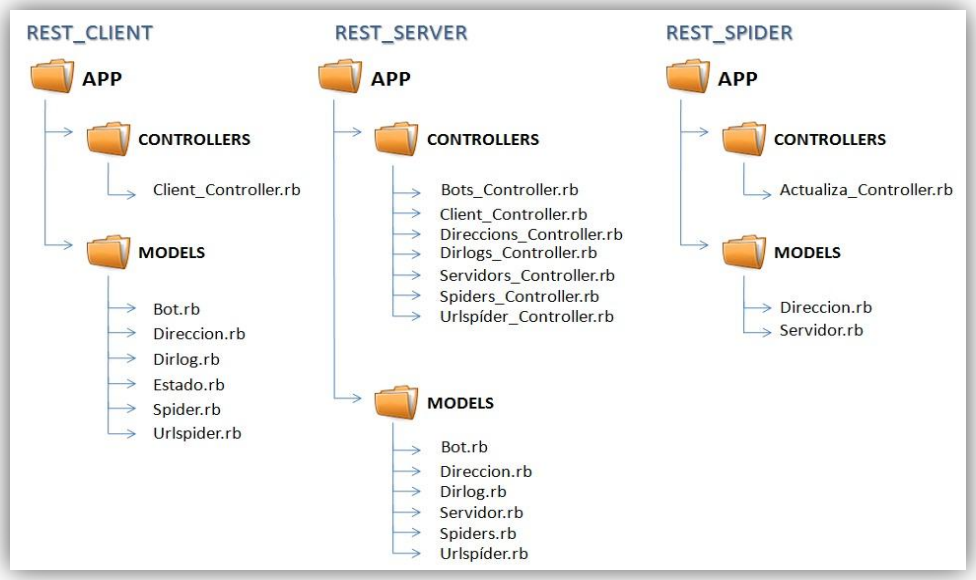



**Figura 21 - Estructura Módulos Cliente – Servidor – Spider**

Cada una de las instancias de los modelos de cada aplicación coinciden ya que es la manera de realizar la comunicación entre los módulos, con respecto al spider tiene en la carpeta de modelos sólo servidor.rb y direccion.rb ya que es lo único que necesita el spider para procesar la información de cada URL.

Ahora se presenta el controlador del rest_client, éste por convención de REST es llamado xxxx_controller, donde xxx representa el nombre del controlador, en este caso client_controller:

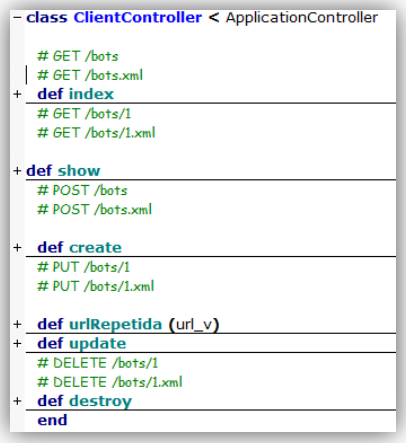



**Figura 22 - Esquema aplicación cliente (rest_client)**

Como se observa el controlador del rest_client se encarga de gestionar las funciones básicas en la base de datos, crear, obtener, actualizar y eliminar (CRUD del original en inglés: Create, Read, Update and Delete) y consiste en dados los datos de nombre del proceso y las URLs, crear un proceso nuevo con sus URLs asociadas, editar las URLs de un proceso dado, mostrar las URLs de un proceso dado y eliminar un proceso o alguna URL asociada a un proceso dado.

A continuación se presentan las funcionalidades CRUD de crear y eliminar:

```
def create (arreglo_url,  nombre_proceso, numero_urls)

 @bot = Bot.create(:name => nombre_proceso)

   while  numero_urls > cont do
      @direccion = Direccion.create(:dir => arreglo_url[cont],:bot_id=>@bot.id)
      cont = cont +1
   end

end
```

**Figura 23 - Método Create (Crear) de rest_client**

```
def destroy

  my_bot = Bot.find(params[:id])
  @url = Direccion.find(:all)

  @url.each{ |dir|
    if dir.bot_id == my_bot.id
      dir.destroy
    end
    }
  my_bot.destroy
end

end
```

**Figura 24 - Método Destroy (Eliminar) de rest_client**

Como se observa, tanto en el método de create como el de destroy, se manejan modelos bot y dirección que se encargan de comunicarse con la base de datos y extraer la información necesaria así como su inclusión, estos modelos son subclases de ActiveRecord para poder realizar de manera correcta la conexión con la base de datos.

Ahora bien, en rest_server se tiene los controladores de cada uno de los modelos generados en el módulo de rest_client, de esta manera, cada vez que se quiera gestionar una funcionalidad CRUD en cualquiera de estos modelos, éste interactúa con su controlador en rest_server y ejecuta el método que se invoque (crear, obtener, actualizar o eliminar). Como se mencionó anteriormente, todos estos controladores son generados con la técnica de andamiaje (scaffold) en Rails, y no necesita ser modificado ya que las CRUD son iguales para todos los modelos.

```
class ServidorsController < ApplicationController
  # GET /servidors
  # GET /servidors.xml
  def index
  # GET /servidors/1
  # GET /servidors/1.xml
  def show
  # GET /servidors/new
  # GET /servidors/new.xml
  def new
  # GET /servidors/1/edit
  def edit
  # POST /servidors
  # POST /servidors.xml
  def create
  # PUT /servidors/1
  # PUT /servidors/1.xml
  def update
  # DELETE /servidors/1
  # DELETE /servidors/1.xml
  def destroy
end
```

**Figura 25 - Operaciones CRUD en controlador Rails**

Por último el controlador de rest_spider, es llamado actualiza_controller en el cual está definido un método llamado update que simula el procesamiento de las URLs y su catalogación en la base de datos, y se realiza un cambio de estado en el que se encuentra actualmente, estos estados pueden ser: ejecutados, sin ejecutar y cancelados.

```
def update
  @servidor = Servidor.find(:all)
  @estados = []
  @servidor.each{ |serv|

     if serv.estado != 1 or serv.estado != 2
          @estados << serv
     end

  }
  @direccion = Direccion.find(:all)
  @tamano_estados = @estados.size
  conta = 0

  while @tamano_estados > conta
        spider = @estados[conta]

        @direccion.each{ |url_actual|
           if url_actual.id == spider.url_id
                if url_actual.estado != spider.estado
                     Direccion.update(url_actual.id, :dir => url_actual.dir, :estado => spider.estado, :bot_id => url_actual.bot_id)
                end
                break
           end
        }
        conta = conta + 1
  end  # End While
end
```

**Figura 26 - Método Update (Actualiza) de rest_spider**

- Pruebas

El desarrollo de las pruebas funcionales se realizó con cada una de las historias de usuario de esta iteración por separado y una última prueba que es la integración de las tres aplicaciones (rest_client, rest_server, rest_spider). A continuación se muestras las pruebas con los resultados obtenidos:

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 5 | 8 | Ingresar por medio de formulario del rest_client información de procesos nuevos y URLs asociadas al proceso creado | La aplicación debe realizar las 4 operaciones CRUD de forma exitosa. | La aplicación mostró resultados satisfactorios al realizar operaciones CRUD sobre procesos y URLs que se ingresan por el formulario de prueba |
| 6 | 9 | Ingresar por formulario rest_server información de procesos y URLs asociadas. | La aplicación debe realizar las 4 operaciones CRUD de forma exitosa. | Al igual que la prueba realizada con el rest_client, se obtuvo resultados exitosos con las operaciones CRUD. |
| 7 | 10 | Ingresar por formulario rest_spider información de procesos y URLs asociadas. | La aplicación debe realizar las 4 operaciones CRUD de forma exitosa. | En el rest_spider se realizan todas las operaciones exitosamente. |

| | | | | |
|---|---|---|---|---|
| 8 | 8, 9, 10 | Integrar las aplicaciones rest_client, rest_server y rest_spider y realizar pruebas a través de un formulario único el cual se puede acceder desde el rest_client | La integración de las aplicaciones no debe generar ningún error y al realizar las operaciones CRUD desde el formulario del rest_cliente debe obtener resultados satisfactorios | La integración se realizó correctamente ya que se comunican a través de la gema de Rails ActiveResource. Las operaciones que se realizan desde el cliente al servidor, y del servidor al spider y viceversa obtuvieron los resultados esperados. |

### 4.2.5 ITERACIÓN 4:

- Gráfico

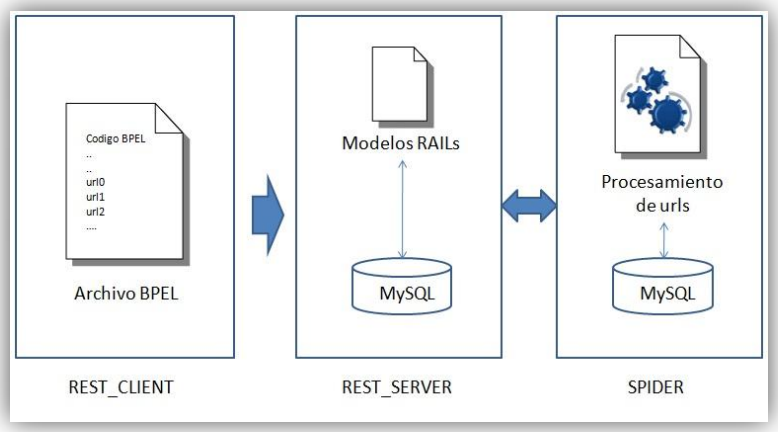



**Figura 27 - Gráfico descriptivo de la iteración 4**

- Planificación

Bajo el esquema de MySQL, se generan las bases de datos para cada aplicación servidor y spider, se verifican las tablas y los campos, y se crea una tabla log para los acontecimientos inesperados producidos durante la ejecución de las aplicaciones.

| **Iteración 4** | | |
|---|---|---|
| **Descripción** | Creación de base de datos para las aplicaciones Servidor y Spider las cuales contienen información de los procesos. Direcciones y spiders disponibles en el sistema | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 11 | Crear una base de datos en el lado del servidor | Nueva |
| 12 | Crear una base de datos en el spider | Nueva |

| | | |
|---|---|---|
| 13 | Modificar base de datos que contengan todos los procesos, direcciones y spider disponibles. | Modificación/ Mejora |
| 14 | Generar una tabla log en la base de datos | Nueva |

- Diseño

  Durante el desarrollo de esta iteración se realiza la creación de bases de datos para cada aplicación Rails que se generó en la iteración anterior. En el caso del cliente (rest_client) no se genera base de datos ya que ésta se encarga de ingresar datos hacia el servidor y obtener respuestas del mismo.

  Por otro lado, tanto el rest_server como el rest_spider trabajan con una base de datos local a cada uno de ellos para manejar la información y procesar los datos según sus sean sus requerimientos.

  Estas dos bases de datos contienen la misma información, sólo que la base de datos del rest_spider en su única tabla posee información de control adicional sobre las URLs que la tabla de la base de datos de la aplicación rest_server.

- Codificación

  Para crear las bases de datos de las dos aplicaciones se toma en cuenta los datos que introduce el usuario a través de la aplicación cliente.

  Para la base de datos de servidor se crean dos tablas esenciales para el funcionamiento correcto de la aplicación, estas tablas son bots (proceso) y direccions, donde proceso tiene el id y el nombre del proceso creado (Figura 28) y direccions tiene un id, una dirección (URL), un estado, y el id del proceso al cual pertenece la URL (Figura 29).

```
mysql> describe bots;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.44 sec)
```

**Figura 28 - Tabla bots de rest_server**

```
mysql> describe direccions;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| dir    | varchar(255) | NO   |     | NULL    |                |
| estado | int(11)      | NO   |     | 0       |                |
| bot_id | int(11)      | NO   |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
4 rows in set (0.08 sec)
```

**Figura 29 - Tabla direccions de rest_server**

Los posibles estados por lo que pasa o puede pasar una URL se muestran en la siguiente figura (Figura 30):

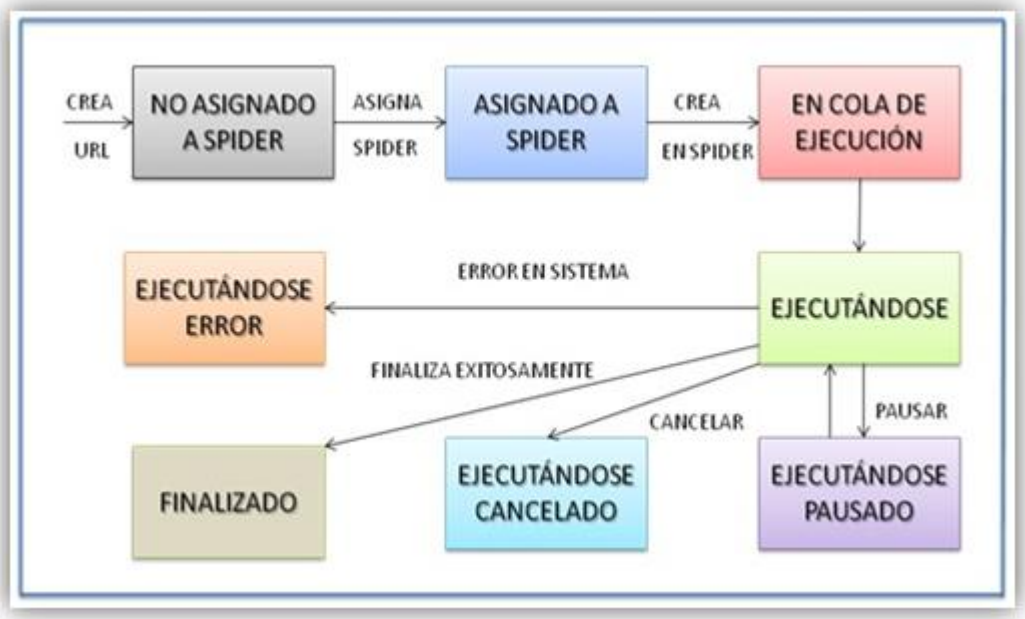



**Figura 30 - Estados posible de una URL**

El campo estado de la tabla direccions es un entero (int) que identifica al estado que se encuentra en una tabla nueva creada, llamada estados. Esta tabla consta de tres campos: id que es el identificador de cada estado, estado que en un entero que se usa al momento de crear o modificar una entrada en la base de datos, esto se utiliza de esta forma para evitar

errores de escritura o desigualdades entre un mismo nombre de estado, y str_estado que indica el nombre del estado actual en la que se encuentra una URL. La figura 31 muestra los campos de la tabla de estados.

```
mysql> describe estados;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| estado     | int(11)      | NO   |     | NULL    |                |
| str_estado | varchar(255) | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

**Figura 31 - Tabla de estados del rest_server**

En la base de datos del rest_spider se crea una tabla denominada sevidors. Esta tabla contiene la misma información que la tabla direccions, sólo que tiene otra información de control de la URL. Esta tabla consta de un id, el id de la URL, el nombre de la URL (dirección URL), el estado en el que se encuentra, el estado anterior que tuvo, el spider asignado a la URL y una hora de inicio y final que indica el tiempo en que duró la ejecución de la URL. La figura 32, muestra la tabla servidors del rest_spider

```
mysql> describe servidors;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| id              | int(11)      | NO   | PRI | NULL    | auto_increment |
| url_id          | int(11)      | NO   |     | NULL    |                |
| direccion_url   | varchar(255) | NO   |     | NULL    |                |
| estado          | int(11)      | NO   |     | NULL    |                |
| estado_anterior | int(11)      | NO   |     | NULL    |                |
| spider_id       | int(11)      | NO   |     | NULL    |                |
| hora_inicio     | datetime     | NO   |     | NULL    |                |
| hora_fin        | datetime     | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)
```

**Figura 32 - Tabla servidors del rest_spider**

Al analizar la arquitectura de las aplicaciones, se procede a modificar la base de datos del rest_server, ya que esta debe tener la información del spider que va a asignar a la URL y un log en donde se registre cada una de las operaciones que se realicen en el servidor y en el spider.

En la figura 33 se muestra la tabla spiders, que contiene un identificador único y la dirección en donde se encuentra el spider.

```
mysql> describe spiders;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| direccion | varchar(255) | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
2 rows in set (0.02 sec)
```

**Figura 33 - Tabla spiders del rest_server**

Por otro lado la tabla dirlogs contiene un id, un módulo que indica en donde se realiza la operación, si en el cliente, en el servidor o en el spider, una operación que indica cual de las 4 operaciones CRUD se está registrando, el log que contiene información detallada del registro obtenido y fecha que nos indica la hora en la que se produjo el evento. En la Figura 34, se muestra la tabla mencionada.

```
mysql> describe dirlogs;
+--------------+----------+------+-----+---------+----------------+
| Field        | Type     | Null | Key | Default | Extra          |
+--------------+----------+------+-----+---------+----------------+
| id           | int(11)  | NO   | PRI | NULL    | auto_increment |
| modulo_id    | int(11)  | NO   |     | NULL    |                |
| operacion_id | int(11)  | NO   |     | NULL    |                |
| log          | text     | NO   |     | NULL    |                |
| fecha        | datetime | NO   |     | NULL    |                |
+--------------+----------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

**Figura 34 - Tabla dirlogs del rest_server**

- Pruebas

Las pruebas realizadas en esta iteración se hicieron con las aplicaciones y las bases de datos explicadas anteriormente. Se realiza la misma prueba para cada una de las aplicaciones y una prueba final con la integración de la aplicación.

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 9 | 11, 12, 13, 14 | Ingresar datos a través de formulario de cada aplicación para verificar que los datos pueden ser almacenados, actualizados, y eliminados | Los datos deber ser guardados, mostrados, actualizados y borrados de forma correcta en cada base de datos | Los datos se manipularon de forma exitosa en cada una de las bases de datos |
| 10 | 11, 12, 13, 14 | Se tiene la integración de las aplicaciones y se procede a realizar prueba en donde se registre todo lo relacionado con las operaciones CRUD | Los datos que genera cada aplicación deber poder registrarse de forma correcta en las dos bases de datos del proyecto | Los datos se guardaron de forma correcta en cada una de las bases de datos con las que trabaja el proyecto |

### 4.2.6 ITERACIÓN 5:

- Gráfico

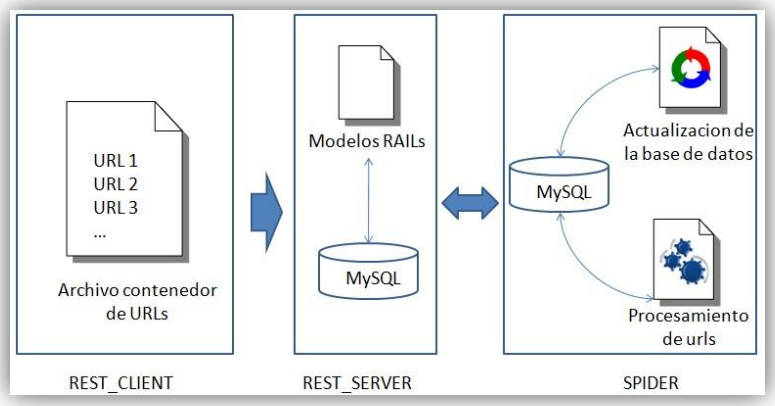



**Figura 35 - Gráfico descriptivo de la iteración 5**

- Planificación

La realización de esta iteración tiene como punto de partida solucionar el inconveniente de hacer que el módulo *spider* (rest_spider) constantemente se encuentre ejecutándose en espera de la generación de nuevos recursos por parte del cliente. Debido a que se utiliza cualquier browser de internet para la ejecución de la aplicación, éstos no permiten la ejecución de ciclos infinitos (debido a que el browser se colapsa), por este inconveniente con el browser, por lo tanto, esta constante revisión debe realizarse desde alguna aplicación que se ejecute paralelamente y de forma infinita.

| **Iteración 5** | | |
|---|---|---|
| **Descripción** | Desarrollo de archivos que actualicen y verifiquen la base de datos del servidor y del spider | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |

| | | |
|---|---|---|
| 15 | Desarrollar código en el spider que actualice el estado de las direcciones en la base de datos | Nueva |
| 16 | Desarrollar código del lado del servidor que verifique la base de datos. | Nueva |

- Diseño

Se desarrolló la idea de crear un módulo externo almacenado en la carpeta*/lib* de la aplicación Rails y escrito en Ruby capaz de ejecutarse infinitamente y que realice una llamada al controlador del spider, el cual tiene como primera tarea verificar los estados en que se encuentra cada recurso y realizar o no su procesamiento dependiendo de su estado. (Figura 36).

Es importante mencionar que en esta carpeta*/lib* se almacenan todas aquellas clases que pueden servir de apoyo o librerías externas que son utilizadas por la aplicación Rails.

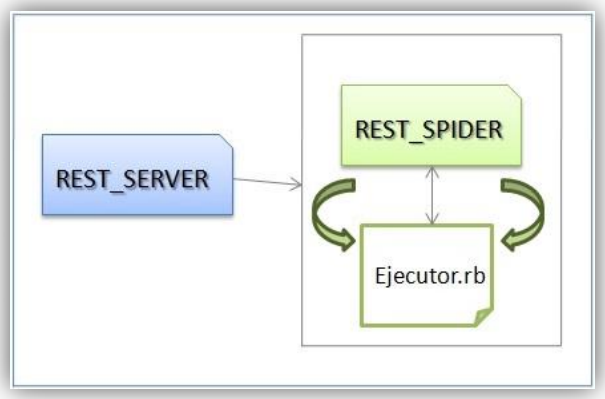



**Figura 36 - Ciclo iterativo de supervisión en el rest_spider**

Se tiene que tomar en cuenta que el código desarrollado en el spider se encarga de supervisar cualquier cambio de estado en la base de datos de las URLs generadas para procesarlas, por lo tanto, en principio, un ente debe generar estos cambios y el spider se percata de los cambios, este ente es un código generado durante la creación de un recurso, el cual verifica el estado de las URLs y la disponibilidad de los spiders y modifica su estado a un estado *asignado a spider.*

- Codificación:

La codificación de esta iteración viene dada por el siguiente código ejecutado en paralelo con las aplicaciones *rest_client, rest_server y rest_spider* de manera de que la ejecución del controlador sea constante. Se puede observar en la siguiente figura el ciclo infinito desarrollado en el archivo *iterador.rb* alojado en la carpeta /*lib* de la aplicación Rails, además se observa la llamada al método *actualiza* que se encuentra en el controlador (*actualiza_controller*).

```
while true

    (0..60).each do |t|
        system "clear"
        puts "cargando.."
        puts "Esperando #{60-t} segs para proxima corrida..."
        sleep 1

    end

    @actualiza = Actualiza.update(:id => 1, :name => "k")

end
```

**Figura 37 - Ciclo infinito iterador.rb**

La llamada al método *update()* del módulo *actualiza* se encarga de verificar los estados de la tabla *servidors* y actualizar los estados de las URLs que se encuentran almacenados en la tabla *direccions* de la base de datos *del rest_server*, ese procedimiento se realiza haciendo la comparación entre el estado anterior y el estado actual del recurso (estos son campos de la tabla *servidors*), en caso de coincidencia no procede a realizar cambios, en caso de discrepancia se realiza la actualización del estado en la tabla *direccions* manteniendo el estado actual.

Por otra parte, para que el spider tenga en su base de datos entradas de las URLs que se han creado, se desarrolló un controlador en el *rest_server* capaz de realizar esta tarea. Este controlador se denomina *cliente_controller*, el cual tiene un método create que realiza el

llenado de la base de datos del *rest_spider* (Figura 38). Este controlador se ejecuta de forma independiente.

```
18
19  def create
20    @direccions = Direccion.find(:all)
21    @array_cola = []
22
23    # Coloco las urls que tienes estado "no_asignado" en un array (array_cola)
24    @direccions.each{ |encolar|
25      if encolar.estado == 0                    # estado no asignado
26            @array_cola << encolar
27      end
28    }
29
30    tamano_cola = @array_cola.size
31
32    cont = 0
33    while cont < tamano_cola
34        # Saco del arreglo el primer elemento p
35        direccion_auxiliar = @array_cola[0]
36
37
38        # Coloco en un array todas las tuplas que tengan estado = 1 (ejecutandose)
39        @servidor = Servidor.find(:all)
40        @array_ejecutandose = []
41
42        @servidor.each{ |serv|
43            if serv.estado == 3     # estado ejecutandose
44                  @array_ejecutandose << serv
45            end
46        }
47
48        if @array_ejecutandose.empty? == true
49              # se realiza una busqueda nueva en la base de datos
50        end
51
52        posibilidad = 0
53        @id_spider_aceptado = 0
54
```

**Figura 38 - Método create del cliente_controller**

El método create revisa cada cierto tiempo la tabla *direccions* verificando la entrada de nuevas URLs a la base de datos, si una URL tiene estado cero (0) significa que es una URL nueva, el código obtiene esa URL, se le asigna a un spider, dependiendo de la disponibilidad, cambia su estado en 1 y manda a crear esa nueva entrada en la base de datos del *rest_spider* para que pueda ser ejecutada y actualiza la información en la tabla *direccions* del *rest_server*.

- Pruebas

Las pruebas que se realizaron en esta iteración fueron de tipo funcional, en donde se crearon diversas entradas en la base de datos del rest_server y se mandó a ejecutar el create del *cliente_controller*, además de realizar cambios en el estado de varias URL en el *rest_spider* para simulación de una ejecución y así demostrar la actualización que realiza en el *rest_server*. Los resultados obtenidos se muestran a continuación:

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 11 | 15 | Realizar cambios en el estado de las URLs de la base de datos del rest_spider de forma de simular la ejecución de las mismas y verificar resultado | Al realizar los cambios en el estado, el código iterador.rb debe verificar y actualizar los estados en la tabla direccions del rest_server de manera correcta | Al realizar un cambio de estado en una URL, se obtuvo la actualización del campo estado de la tabla direccions exitosamente, y el estado de una URL en particular es el acorde con el que indica en el rest_spider. |
| 12 | 16 | Revisar cada cierto tiempo la tabla direccions para verificar si hay una nueva URL creada y enviarla al spider con el estado actualizado para ambas tablas | El método debe revisar la tabla direccions en busca de una nueva URL, asignarle un spider en caso de que exista, y cambiar el estado original de la misma | El resultado fue el esperado. El método seleccionaba a las URL que estaban nuevas en la base de datos, le asignaba un spider, actualizaba el campo estado en direccions y creaba una entrada en |

| | | | | |
|---|---|---|---|---|
| | | (direccions y servidors) | | servidors de rest_spider exitosamente. |

### 4.2.7 ITERACIÓN 6:

- Planificación

Esta iteración cumple un rol esencial durante la ejecución del programa, ya que prevee y evita dar paso a posibles errores capaces de detener y terminar la ejecución de la aplicación, con la incursión de manejadores de excepciones.

| **Iteración 6** | | |
|---|---|---|
| **Descripción** | Realizar validaciones a nivel del sistema para el cliente, el servidor y el spider | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 17 | Realizar validaciones en el cliente | Nueva |
| 18 | Realizar validaciones en el servidor | Nueva |
| 19 | Realizar validaciones en el spider | Nueva |

- Diseño

Esto se realizó estudiando cuáles son los puntos críticos en el código donde hay mayor manipulación de datos de entrada e iteraciones de ciclos dependientes de variables externas, cuyo procesamiento puede incurrir en distintas excepciones. Para tratar una posible excepción, se coloca el segmento de código que puede causar el error dentro de un bloque *begin .. end* el cual se ejecutará hasta que haya una excepción, lo que provoca que el control se transfiera a un bloque con el código de gestión de errores, aquel marcado con rescue.

Es importante acotar que el código interrumpido por la excepción, nunca se ejecuta. Una vez que la excepción es manejada (por el *rescue*), la ejecución continúa inmediatamente después del bloque *begin* fuente del error.

- Codificación

Debido a que principalmente las validaciones se hacen sobre los datos de entrada de un sistema, en este caso, los nombres de procesos y URLs provenientes de usuario y procesados por el *rest_client* fueron validadas sintácticamente, por lo cual el proceso de validación se centra en las excepciones que pueden surgir durante su procesamiento, para poder realizar esto se colocaron los bloques *begin.. rescue.. end* en puntos más vulnerables del código para evitar errores, se presentan a continuación algunas de estas ubicaciones:

```
begin
   @bot = Bot.create(:name => @nombre_proceso)
   @msj_bot = "OK"
rescue Exception => e
   @msj_bot = e.message
   Dirlog.create(:modulo_id => 1, :operacion_id => 1, :log => @msj_bot +" creacion de proceso", :fecha => Time.now)
end

@response_url = []

   @arreglo_url.each{ |url|
    if !urlRepetida(url)
      begin
        Direccion.create(:dir => url,:bot_id=>@bot.id)

        @response_url << "OK"
      rescue Exception => e
        @response_url = e.message
        Dirlog.create(:modulo_id => 1, :operacion_id => 1, :log => @response_url +" creacion de URL", :fecha => Time.now)
      end

    else
      @response_url << "URL REPETIDA"
      Dirlog.create(:modulo_id => 1, :operacion_id => 1, :log => "URL REPETIDA: "+url, :fecha => Time.now)
      cont += 1
    end
   }
```

**Figura 39 - Validación en  rest_client**

Como se puede observar, el segmento más crítico de este código es la generación de las direcciones, además genera un mensaje resultado que puede ser accedido por el invocador para informar el resultado de la operación.

```
begin

while @tamano_estados > conta
      spider = @estados[conta]

      @direccion.each{ |url_actual|
         if url_actual.id == spider.url_id
               if url_actual.estado != spider.estado
                     Direccion.update(url_actual.id, :dir => url_actual.dir, :estado => spider.estado, :bot_id => url_actual.bot_id)
               end
               break
         end
      }

      conta = conta + 1
 end  # End While
end

rescue Exception => e
        @response_url = e.message
        Dirlog.create(:modulo_id => 3, :operacion_id => 2, :log => "error de ejecucion", :fecha => Time.now)
  end
```

**Figura 40 - Validación en actualiza_controller**

Como se puede observar en los dos ejemplos, el manejo de la excepción viene dado por una entrada en la tabla *log* la cual se encarga de almacenar los registros de cualquier eventualidad, con una descripción de la ubicación aproximada de donde surgió la excepción y una descripción de la misma.

- Pruebas

Las pruebas que se desarrollaron en este punto fueron de creación de URLs y procesos de forma correcta y de forma errada, para verificar que el resultado que se obtenga sea el esperado. A continuación se muestran las pruebas realizadas:

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 13 | 17, 18, 19 | Para las tres aplicaciones Rails se crearon entradas validas, es decir, que no | Al procesarse entradas válidas en el sistema, este no presente ningún problema y guarde de forma correcta los datos en la base de datos. | Al crear nuevas URLs al sistema, todo se creó de forma correcta, las |

| | | | | |
|---|---|---|---|---|
| | | generan error para verificar si se realizaba de manera correcta el proceso después de agregar las validaciones. | | validaciones agregada no perjudicaron el procesamiento cuando las entradas no generan error |
| 14 | 17, 18, 19 | Se agregó nuevas URLs que no eran válidas, es decir, que entran en el rescue y ejecutan el error para observar la salida y error generado | Al entrar la URL por el rescue debe manejarse el error y mostrar un mensaje indicando el error y guardarlo en la tabla dirlogs. | Las URLs que entran por el rescue muestran un error por pantalla y lo registran en el la tabla dirlogs. |

En esta etapa del desarrollo del proyecto surgió la necesidad de revisar los alcances y tecnologías involucradas, por lo tanto se generaron nuevas historias de usuario derivadas de la discusión con el cliente. Se planeó un cambio de enfoque, principalmente cambiar de una arquitectura SOA orientada a procesos de negocios a una arquitectura de Cloud Computing. En el apéndice 8.1 se muestra la arquitectura detallada.

En la siguiente figura se observa la nueva arquitectura del sistema siguiendo el nuevo enfoque:

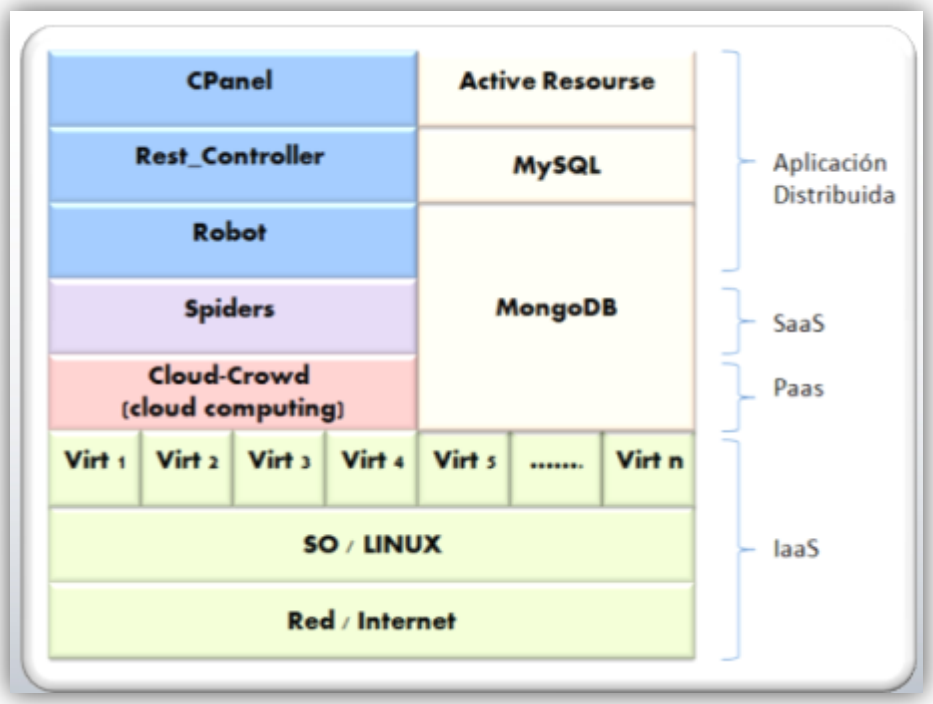



**Figura 41 - Arquitectura general en capas**

A continuación se listan dichas historias de usuario y su correspondientes iteraciones (de la 7 a la 11):

| **Número: 20** | **Nombre:** Unificación de las base de datos |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 3 días |
| **Descripción:** Crear una base de datos que contenga la unificación de la base de datos de la aplicación usuario, servidor y la del spider. | |

| **Número: 21** | **Nombre:** Realizar la integración de las aplicaciones cliente servidor y spider con los otros módulos del proyecto |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 7 días |
| **Descripción:** Revisar, modificar y actualizar las aplicaciones cliente, servidor y spider con los otros módulos del proyecto: CPanel, Servidor, etc. | |

| **Número: 22** | **Nombre:** Desarrollar, incorporar y adaptar funcionalidades desarrolladas para clientes. |
|---|---|
| **Tipo:** Mejora / Modificación | **Tiempo Estimado:** 4 días |
| **Descripción:** Modificar y adaptar las funcionalidades realizadas en la aplicación cliente en el módulo Cpanel. | |

| **Número: 23** | **Nombre:** Realizar algoritmo para la validación de URI válidos para el sistema |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Realizar algoritmo que verifique si una uri es válida o no. Debe arrojar un booleano de salida, donde verdadero es una uri absoluta y falso una uri mal formada o relativa. | |

| **Número: 24** | **Nombre:** Realizar diccionario de datos de la base de datos de control (ITICVE) |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 2 días |
| **Descripción:** Realizar un diccionario de datos, donde indique función que tienen los campos de las tablas de la base de datos de control. | |

| **Número: 25** | **Nombre:** Desarrollar modelos ActiveRecord para cada tabla de la base de datos de control |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Desarrollar modelos ActiveRecord para cada tabla de la base de datos y agregar las asociaciones entre tablas y validaciones en modelos. | |

| **Número: 26** | **Nombre:** Desarrollar Controllers para la aplicación |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 9 días |
| **Descripción:** Desarrollar controladores para los procesos, direcciones, spider, que permitan dar servicio a clientes ActiveRecord. | |

| **Número: 27** | **Nombre:** Realizar una clase en Ruby para el acceso a la base de datos MongoDB |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 6 días |
| **Descripción:** Realizar una clase Writer escrita en Ruby para facilitar, mediante encapsulamiento, el acceso a la base de datos para la caché MongoDB | |

| **Número: 28** | **Nombre:** Generar algoritmo para el análisis de documentos web. |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 6 días |
| **Descripción:** Generar algoritmo que tenga como entrada un arreglo de uris, las cuales serán analizadas para obtener como salida un arreglo de objetos tipo URI (validos) o documentos (imágenes, audio, video, etc.). | |

| **Número: 29** | **Nombre:** Desarrollar algoritmo para exportar páginas en archivo a PDF |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 4 días |
| **Descripción:** Crear un algoritmo que reciba un conjunto de URLs, las procese y genere como salida un archivo PDF que el usuario final podrá descargar del portal. El PDF contiene la imagen de la URL o URLs que el usuario seleccionó a través del portal y fueron procesaron. | |

| **Número: 30** | **Nombre:** Realizar algoritmo worker cuya ejecución en segundo plano analice el arreglo de salida del spider |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 8 días |
| **Descripción:** Realizar algoritmo que analice en segundo plano archivo de salida generado por el spider para filtrar las uris redireccionadas, archivos multimedia y documentos. | |

| **Número: 31** | **Nombre:** Desarrollo de workers de workling para los diferentes trabajos en segundo plano |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 7 días |
| **Descripción:** Desarrollar workers de workling para Rails para trabajar en segundo plano con diferentes tareas | |

| **Número: 32** | **Nombre:** Integración de trabajos en la nube |
|---|---|
| **Tipo:** Nueva | **Tiempo Estimado:** 8 días |
| **Descripción:** Integrar trabajos realizados en segundo plano en la nube para un procesamiento más eficiente. | |

### 4.2.8 ITERACIÓN 7:

- Gráfico

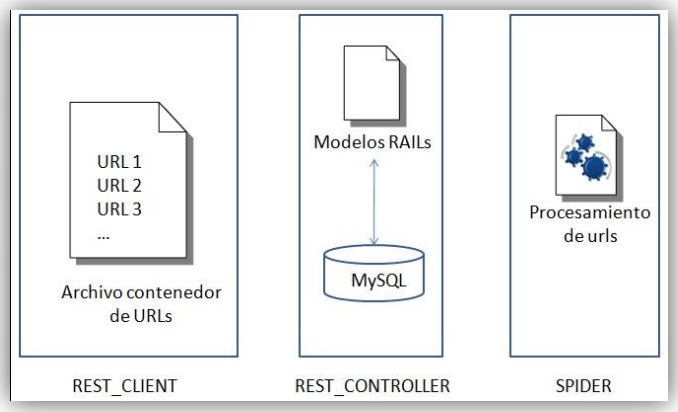



**Figura 42 - Gráfico descriptivo de la iteración 7**

- Planificación

Durante esta iteración se realizó la unificación de las bases de datos, como se observa cada módulo cuenta con su propia base de datos, y además existe comunicación entre los módulos, por lo tanto, para realizar un equilibrio entre las funcionalidades de cada módulo se creó una recopilación de las bases de datos existentes en el módulo de *rest_server*.

| **Iteración 7** | | |
|---|---|---|
| **Descripción** | Unificación e integración de las base de datos y aplicaciones cliente, servidor y spider con los otros módulos del proyecto | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 20 | Unificación de las base de datos | Nueva |
| 21 | Realizar la integración de las aplicaciones cliente servidor y spider con los otros módulos del proyecto | Nueva |

- Codificación

Generación de una base de datos llamada *ITICVE* en la cual se importaron todas las tablas de las bases de datos externas, por tanto, aquellas tablas con nombres que coincidían fueron editadas, dando como resultado la siguiente lista de tablas de la base de datos:


**Figura 43 - Base de datos ITICVE**

Además de unificar las bases de datos, también se realizaron todas las modificaciones en los módulos que realizaban llamadas a tablas a las cuales se les cambio el nombre por coincidir con el nombre de otras tablas.

Debido al cambio de estructura, para acceder a una tabla desde un módulo distinto al de *rest_server* la conexión se realiza orientada a recursos, es decir, se debe generar un modelo en el módulo que haga la solicitud de acceso a la base de datos, que se comunique con un modelo (del mismo nombre) en el módulo del *rest_server* y de esta manera interactúan los datos de las tablas. Es importante mencionar que para acceder a cada tabla, el modelo tiene que tener el mismo nombre de la tabla pluralizado. En la siguiente figura se explica cómo se accede a los datos de una tabla *direccions* desde el módulo de rest_client:

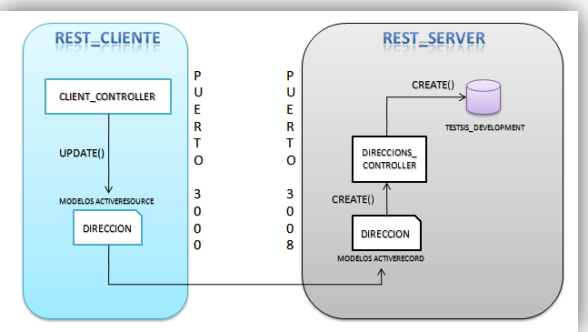



**Figura 44 - Acceso desde rest_client a rest_server**

- Pruebas

Para esta iteración se desarrollaron dos pruebas sencillas, la primera para verificar que todos los accesos a la base de datos se hayan cambiado en los archivos y guarde correctamente los datos, y una segunda prueba a las aplicaciones desde el cliente para verificar que funcione correctamente.

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 13 | 20 | Se realizan operaciones CRUD a la base de datos para verificar resultados | Al realizar las consultas a la base de datos debe poder guardar, modificar, mostrar y borrar los datos de la base de datos. | Se realizaron consultas a la base de datos a través de las aplicaciones y realiza la operación de forma satisfactoria. |
| 14 | 21 | Se realiza prueba general de cliente y servidor | Funcionamiento correcto de las aplicaciones | Al realizar las pruebas el cliente y el servidor se comunicar con éxito. |

### 4.2.9 ITERACIÓN 8:

- Gráfico

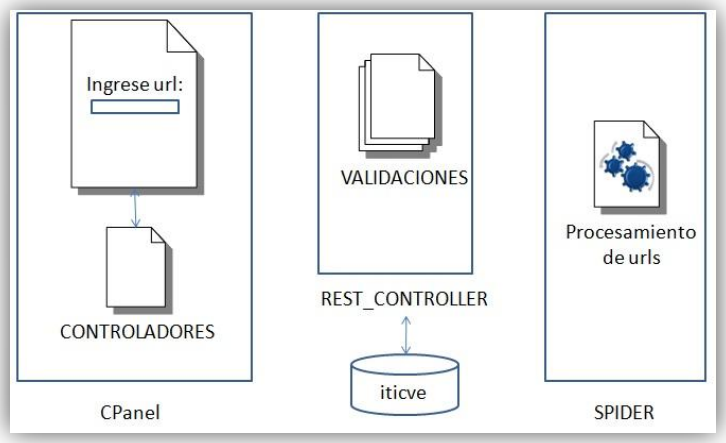



**Figura 45 - Gráfico descriptivo de la iteración 8**

- Planificación:

  Durante el proceso de unificación de códigos el módulo de rest_client se combina con el cliente (ente externo al proyecto al que tiene acceso el usuario), de manera de establecer tres grandes grupos: panel de control (cpanel), spider y controlador, este último es el evaluado en este trabajo. Por lo tanto, muchas de las funcionalidades, llamadas a rutinas y conexiones deben adaptarse al nuevo entorno en el cpanel.

  Además, se desarrolla un código externo capaz de verificar sintácticamente la correctitud de una URL, de manera de evitar errores durante su procesamiento por el spider.

  Y por último realizar una compilación en la base de datos de las características lógicas y puntuales de los datos que se van a utilizar en el sistema.

| **Iteración 8** | | |
|---|---|---|
| **Descripción** | Adaptación de funcionalidades para aplicación cliente – Cpanel, desarrollo de algoritmo de validación y generación de diccionario de datos | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 22 | Desarrollar, incorporar y adaptar funcionalidades desarrolladas para clientes. | Nueva |
| 23 | Realizar algoritmo para la validación de URI válidos para el sistema | Nueva |
| 24 | Realizar diccionario de datos de la base de datos de control (ITICVE) | Nueva |

- Diseño

  Ciertas funcionalidades como conexión con el rest_server, conexión con el rest_spider, proceso de creación de procesos y URLs fueron adaptadas al cliente y despreciadas en este proyecto.

  Se desarrolló un módulo que se encarga de verificar la validez sintáctica de una URL, verificando que se encuentre bien estructurada para ser enviada al spider, debido a la innumerable cantidad de schemas (parte de la URL que define el protocolo) se utiliza un archivo externo como compilación de la mayoría de los schemas más utilizados y se compara con el schema de la URL dada, y de esta manera se pasa la primera fase de la prueba de validez de la URL, luego se verifica que contenga un host, y de ser una dirección ip, se verifica que los números sean validos.

  Y por último se desarrolló un diccionario de datos en donde se encuentra la lista de todos los elementos que forman parte del flujo de datos de todo el sistema. Los elementos más

importantes son flujos de datos, almacenes de datos y procesos. El diccionario de datos guarda los detalles y descripción de todos estos elementos.

- Codificación

Durante esta iteración se desarrolla un método llamado *validar()* el cual se encuentra en el archivo *validar_URL.rb* y al cual se realiza la llamada pasando por parámetros la URL que se quiere verificar, este método está escrito en Ruby, y la estructura está definida en la siguiente figura:

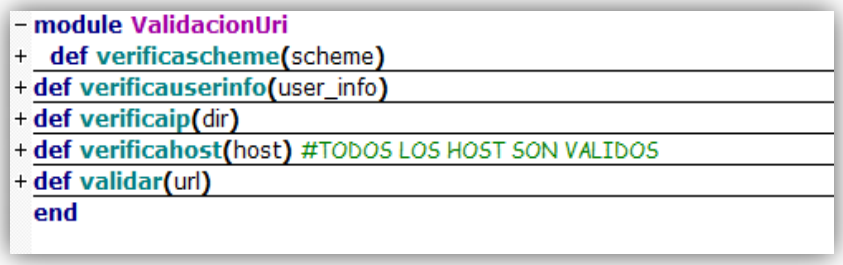



**Figura 46 - Estructura de validar()**

Donde los métodos verificaschema(), verificainfo(), verificahost(), se encargan de validar las tres partes principales de una URL válida, estas partes de la URL son extraídas usando URI.split, el cual se encarga de subdividir todas aquellas partes que nos interesan de una URL. A continuación se muestra en la siguiente figura su utilización:

```
begin
   uri = URI.split(url)
   uri_valida = true
 rescue
  return(false)
 end

 print("scheme: ")
 puts (uri[0]) # scheme
 print("userinfo: ")
 puts (uri[1]) # userinfo
 print("Host: ")
 puts (uri[2]) # Host
 print("Port: ")
 puts (uri[3]) # Port
 print("Registry: ")
 puts (uri[4]) # Registry
 print("Path: ")
 puts (uri[5]) # Path
 print("Opaque: ")
 puts (uri[6]) # Opaque
 print("Query: ")
 puts (uri[7]) # Query
 print("Fragment: ")
 puts (uri[8]) # Fragment
 puts ("--------------------")
```

**Figura 47 - Utilización Uri.split()**

Como se puede observar, se genera un arreglo el cual almacena en cada una de sus posiciones cada parte de la URL procesada.

Por último se tiene el método *verificarip()* el cual se invoca al verificarse que la URL es una dirección ip, este método se encarga de validar cada uno de los octetos que conforman la URL de manera de que se encuentren en el rango válido y no se encuentren caracteres no numéricos, estas validaciones se pueden observar en la siguiente figura:

```
if ( (dir.split('.').length )== 4 and
   ( dir.split('.')[0].to_s.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false : true ) and
   ( dir.split('.')[1].to_s.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false : true ) and
   ( dir.split('.')[2].to_s.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false : true ) and
   ( dir.split('.')[3].to_s.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false : true ) )

   if ( ( (0 < dir.split('.')[0].to_i) and (dir.split('.')[0].to_i) <= 255 ) and
      ( (0 <= dir.split('.')[1].to_i) and (dir.split('.')[1].to_i) <= 255 ) and
      ( (0 <= dir.split('.')[2].to_i) and (dir.split('.')[2].to_i) <= 255 ) and
      ( (0 <= dir.split('.')[3].to_i) and (dir.split('.')[3].to_i) <= 255 )    )
        puts "es ip"
        return true
   else
        return false
   end
```

**Figura 48 - Validaciones URL**

De esta manera, se concluye que al verificar la validez de cada una de las partes de la URL, se asume la validez de la URL completa.

- Pruebas

Durante el desarrollo de esta iteración se realizan pruebas sobre el archivo de validar uri, son tres pruebas las que se realiza, una con una uri valida, otra con una uri no valida y una última con la dirección ip. Estas pruebas se muestran a continuación.

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 15 | 23 | Ingresar como entrada una uri válida para verificar salida | Al ser la entrada una uri válida debe arrojar como salida un booleano true que indica que es válida la uri introducida | La uri de entrada obtuvo como salida true, es decir, que era una uri válida |
| 16 | 23 | Ingresar como entrada una uri no válida o mal formada para verificar salida | Al ser la entrada una uri no válida debe arrojar como salida un booleano false que indica que la uri introducida no cumple con el proceso de validez | La uri de entrada obtuvo como salida false, es decir, que era una uri no válida o mal formada |
| 17 | 23 | Ingresar como entrada una IP para verificar salida | Al ser la entrada una IP debe verificar que sea un esquema IP y arrojar como salida un booleano true que indica que la IP introducida cumple con el proceso de validez | La IP de entrada obtuvo como salida true, es decir, que cumple con el esquema de una IP, y por tanto es válido |

### 4.2.10 ITERACIÓN 9:

- Gráfico

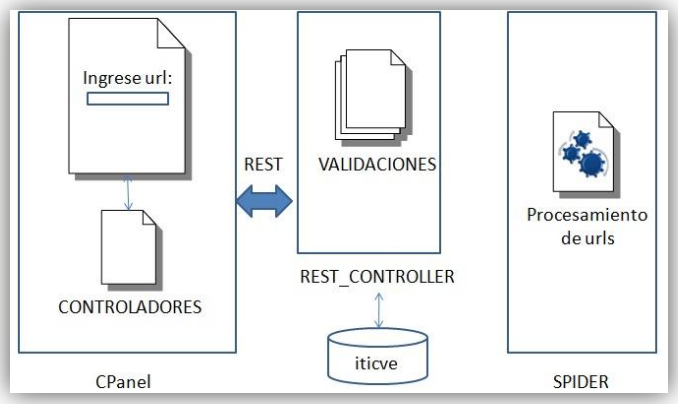



**Figura 49 - Gráfico descriptivo de la iteración 9**

- Planificación

Durante el desarrollo de esta iteración se crearon modelos ActiveRecord necesarios para realizar las consultas a la base de datos, además de controladores para cada uno de los modelos ActiveRecord que permiten la comunicación con otros modelos y otras tablas de la base de datos.

| **Iteración 9** | | |
|---|---|---|
| **Descripción** | Desarrollo de modelos y controladores para la aplicación Servidor y Cpanel | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |
| 25 | Desarrollar modelos ActiveRecord para cada tabla de la BD de control | Nueva |

| | | |
|---|---|---|
| 26 | Desarrollar Controllers para la aplicación que permitan dar servicio | Nueva |

- Diseño

  En la iteración anterior se adaptó el rest_client y cliente CPanel, para formar una sola aplicación llamada Cpanel. Al desarrollar funcionalidades, adaptarlas y/o modificarlas se debe de crear más controladores, modelos, vistas, etc., para que la aplicación funcione correctamente. Al unificar las bases de datos de rest_client, rest_server, rest_spider y Cpanel se deben de crear modelos ActiveRecord por cada tabla de la base de datos de control ITICVE, hay que definirlas y agregar las asociaciones entre las tablas que estén relacionadas y validaciones de base de datos.

  Además de crear los modelos ActiveRecord por cada tabla, hay que desarrollar controladores para los procesos, URLs, módulos, etc., que permitan dar servicio a los clientes ActiveRecord.

- Codificación

  Para crear los modelos ActiveRecord de las tablas de la base de datos de control, primero hay que tener en cuenta cuales tablas fueron agregadas a la base de datos. En la figura 37 se muestran las tablas que tiene ITICVE.

  Los modelos ActiveRecord tienen el mismo nombre que la tabla pero en singular. En el modelo se define las relaciones entre tablas (uno a uno, uno a muchos) con las sentencias *has_many :tabla relación y belongs_to :tabla relación.* En la figura 50 se muestra un ejemplo de esto.

proceso.rb

```
1  class Proceso < ActiveRecord::Base
2    has_many :direcciones
3
4  end
5
```

direccion.rb

```
1  class Direccion < ActiveRecord::Base
2  belongs_to :proceso
3
4  class << self
5    def crear_imagen(aux)
```

**Figura 50 - Relación entre tabla en modelo ActiveRecord**

Adicionalmente en los modelos se puede crear métodos que hagan validaciones de datos, por ejemplo la clave de ingreso de un usuario se debe de verificar que en la base de datos tenga el mismo valor para poder acceder al sistema de administración. Un ejemplo de estas validaciones en un modelo ActiveRecord se muestra en la figura 51.

```
1  class Usuario < ActiveRecord::Base
2
3    validates_uniqueness_of :login, :message => "el usuario ya existe"
4    validates_presence_of :login, :password, :correo, :nombre, :apellido, :perfil, :message => "el campo no puede estar vacÃƒÂ-o"
5    #validates_size_of :clave, :in => 8..16, :message => "la clave debe estar entre 8 ÃƒÂ³ 16 caracteres"
6    validates_format_of :correo, :with => /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})$/i, :message => "correo invÃƒÂ¡lido"
7    validates_format_of :login, :with => /^\w+$/i, :message => "solo puede contener letras y numeros."
8    validates_confirmation_of :password
9    #/\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i
10
11
12   def encriptar_clave(clave)
13
14   end
15
16 end
17
```

**Figura 51 - Ejemplo Validaciones en modelo ActiveRecord**

Además de crear los modelos ActiveRecord, se deben de generar controladores asociados a estos modelos para que se pueda realizar operaciones de consulta en el sistema hacia la base de datos, desde el cliente. Si el tipo de consulta es básico, es decir, no tiene validaciones ni código agregado, se puede crear el controlador con el método de andamiaje (scaffold), el cual contiene las operaciones básicas de un recurso (crear, actualizar, obtener y eliminar), si se desea agregar código adicional, agregar métodos, etc., se puede realizar sin problema alguno en el controlador creado con el scaffold. A continuación en la figura 52 se muestra un ejemplo de un controlador necesario en el proyecto creado con scaffold.

```ruby
class EstadosController < ApplicationController
  # GET /estados
  # GET /estados.xml
  def index
    @estados = Estado.all

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

  # GET /estados/1
  # GET /estados/1.xml
  def show
    @estado = Estado.find(params[:id])

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

  # GET /estados/new
  # GET /estados/new.xml
  def new
    @estado = Estado.new

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

  # GET /estados/1/edit
  def edit
    @estado = Estado.find(params[:id])
  end

  # POST /estados
  # POST /estados.xml
  def create
    @estado = Estado.new(params[:estado])

    respond_to do |format|
      if @estado.save
        format.html { redirect_to(@estado, :notice => 'Estado was successfully created.') }
```

**Figura 52 - Controlador Estado_controller generado con Scaffold**

- Pruebas

  En esta iteración se desarrolla una sola prueba, la cual se realiza a nivel general del sistema, para verificar que el sistema siga funcionando correctamente, como lo hacía antes de agregarle los modelos y los controladores.

| No. | H.U. involucradas | Descripción del Caso de Prueba | Resultado Esperado | Resultado Obtenido |
|---|---|---|---|---|
| 18 | 25, 26 | Se introducen datos a través de la aplicación Cpanel para | Al procesar una operación del Cpanel, se debe de guardar y/o modificar en la base de datos y realizar todo el proceso. | Al ingresar datos por el Cpanel, estos se procesan de forma correcta obteniendo |

| | | | | |
|---|---|---|---|---|
| | | verificar que funcione de forma correcta | | resultados satisfactorios |

### 4.2.11 ITERACIÓN 10:

- Gráfico

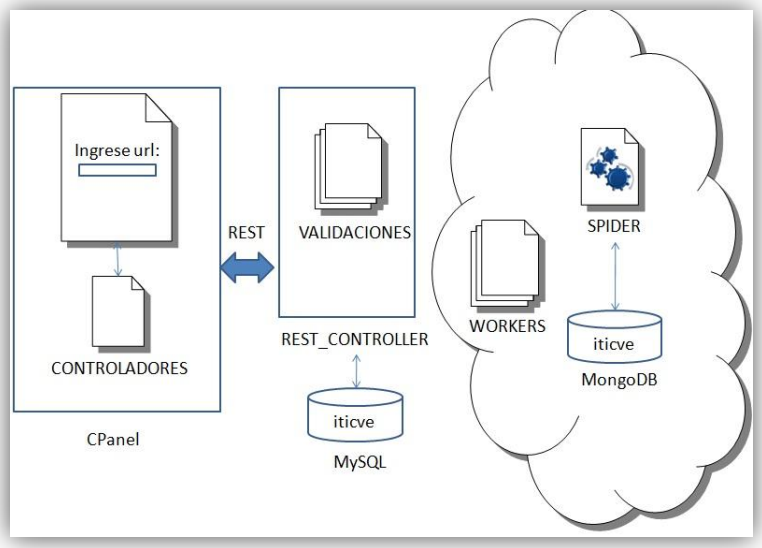



**Figura 53 - Gráfico descriptivo de la iteración 10**

- Planificación

Durante el desarrollo de esta iteración se realizaron algoritmos para acceder y guardar datos en la base de datos MongoDB, para analizar los documentos que serán enviados a MongoDB y crear archivos en formato PDF que contiene la imagen o imágenes de una URL o URLs que se encuentran almacenadas en MongoDB

| **Iteración 10** | |
|---|---|
| **Descripción** | Creación y Desarrollo de algoritmos en Ruby para el acceso a MongoDB, para exportar páginas en archivos PDF, y analizar salida de documentos Web. |
| **Fecha Inicio / Fecha Fin** | |

| Número | Historia | Tipo |
|---|---|---|
| 27 | Realizar una clase en Ruby para el acceso a MongoDB | Nueva |
| 28 | Generar algoritmo para el análisis de documentos Web. | Nueva |
| 29 | Desarrollar algoritmo para exportar páginas en archivo a PDF | Nueva |

- Diseño

Durante esta iteración se desarrollaron algoritmos que están relacionados unos con otros. Estos algoritmos básicamente obtienen documentos (archivos, imágenes, audio, video, etc.), verifican la salida que se genera y acceden a la base de datos MongoDB para almacenar dichos datos de una URL.

El primer algoritmo a realizar es para acceder a la base de datos MongoDB, para así poder realizar las operaciones básicas sobre los datos. Estas operaciones son: guardar, obtener, actualizar o eliminar documentos de una URL.

El algoritmo que genera la salida para ser almacenada en MongoDB lo realiza el Spider. Este spider tiene como entrada un grupo de URLs que han sido creadas en la base de datos de control *(ITICVE)*. Este módulo por cada URL de entrada verifica el tipo MIME del archivo, dependiendo de la extensión que tenga la URL (por los momentos no se toman en cuenta las URLs que no tienen extensión).

Para verificar el tipo MIME del archivo de entrada, éste se compara con una lista MIME almacenada en un archivo de texto plano, así como también la extensión de la URL. De esta manera se verifica si la entrada es o no una URL, ya que se puede tener como entrada documentos o URLs dentro de una URL.

Si se trata de un documento, se crea un objeto que va a contener el documento, el tipo MIME y la URL donde pertenece el documento, en caso contrario se almacenan todas las URLs que están dentro de la URL que está siendo procesada. Este proceso se estará

realizando hasta que en la salida sólo haya documentos y no URLs. Estos documentos serán almacenados en la base de datos a través de un objeto llamado Item.

Con estos datos almacenados en la base de datos MongoDB, se crea un archivo PDF que contiene la visualización de la URL o URLs de entrada, este algoritmo se genera con el fin de que si un usuario necesita obtener información u obtener ciertas estimaciones de algunas páginas web, solo las seleccione y se genera el archivo PDF. Este algoritmo estará disponible para el portal de usuario, el cual no está en el alcance de este proyecto.

### 4.2.12 ITERACIÓN 11:

- Gráfico

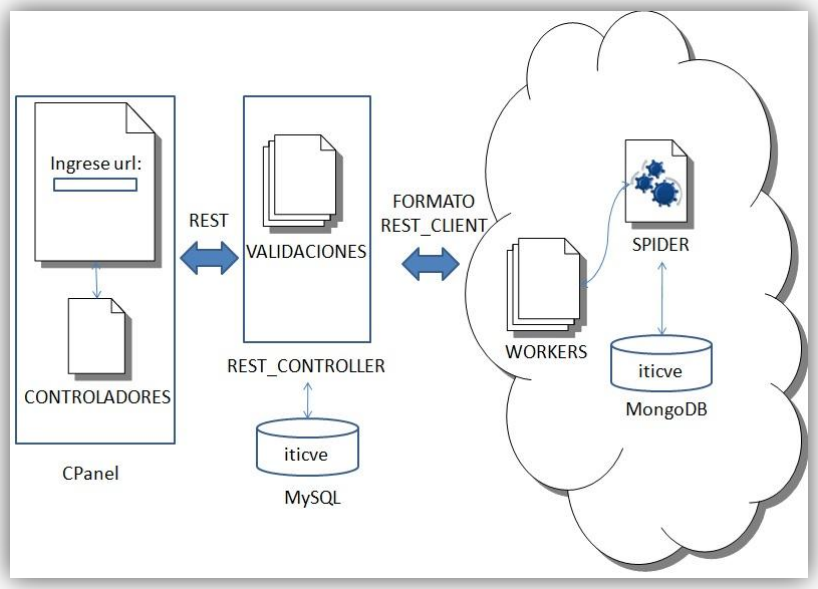



**Figura 54 - Gráfico descriptivo de la iteración 11**

- Planificación

Durante esta iteración se crean en la *nube* los archivos y el código necesario para implementar la ejecución en segundo plano del procesamiento de las URLs y almacenamiento del código HTML y multimedia que conforman la página Web.

| **Iteración 11** | | |
|---|---|---|
| **Descripción** | Desarrollo de workers de workling, algoritmo worker e integración de tareas a la nube | |
| **Fecha Inicio / Fecha Fin** | | |
| **Número** | **Historia** | **Tipo** |

| | | |
|---|---|---|
| 30 | Realizar algoritmo worker cuya ejecución en segundo plano analice el arreglo de salida del spider | Nueva |
| 31 | Desarrollo de workers de workling para los diferentes trabajos en segundo plano | Nueva |
| 32 | Integración de trabajos en la nube | Nueva |

- Diseño

En primera instancia es importante acotar que todos los códigos desarrollados en esta fase están escritos en Ruby.

Para la ejecución en segundo plano del procesamiento de la URL se generó el archivo robot_worker.rb el cual debe estar almacenado en la capeta workers en nuestra aplicación como se observa a continuación:

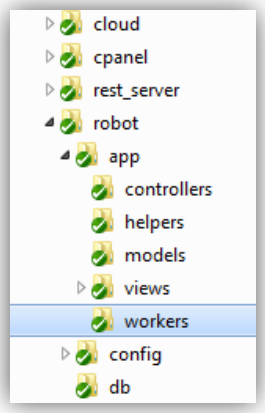



**Figura 55 - Ubicación de la carpeta worker**

En este archivo se escribe el código para invocar la rutina *start()*, en la siguiente figura se muestra la invocación:

```
class RobotWorker < Workling::Base
  def start(options = {})
  puts "EN WORKER"
    Robot.crawl()

  end
end
```

**Figura 56 - Código de llamada a worker**

Este código genera una instancia al modelo robot, el cual se encuentra en el archivo en la carpeta /models, y en el cual se encuentra el código para procesar todas las URLs de entrada.

```
require 'rubygems'
#require 'anemone'
#require '../../lib/nettools.rb'

class Robot

      def self.crawl(opciones={})
end
```

**Figura 57 -Modelo del Workling**

# CONCLUSIÓN

Al concluir la realización de este Trabajo Especial de Grado, se obtuvo la construcción de una arquitectura compuesta por dos aplicaciones denominadas rest_controller y spider las cuales forman parte del Prototipo de Archivo Web que se está desarrollando en el Centro de Computación Paralela y Distribuida de la Escuela de Computación (CCPD), y cuyas funciones son verificar, procesar y almacenar la información suministrada por el administrador desde la aplicación CPanel y desarrollar una extracción de toda la información de alguna página dada para ser archivado en la base de datos MongoDB.

El desarrollo de la aplicación presentó cierto nivel de dificultad debido a que el prototipo aún estaba en fase de planificación y los muchos lineamientos con respecto al cómo llegar al objetivo principal aún estaban en discusión, pero la mejor manera de verificar los alcances de la estructura establecida era poner en práctica el esquema planteado, es por esto que el desarrollo de este Trabajo Especial de Grado fue dividido en dos grandes fases de desarrollo: *Prueba de Concepto* y *Puesta Punto del Prototipo*. Durante la primera fase se fueron inspeccionando diversas soluciones para el desarrollo del controlador principal de la aplicación, pasando por cambios de estructuras como cambios de ambientes, y en la segunda fase se constituye una solución eficiente la cual se adapta a la necesidad presentada.

Es importante resaltar que el desarrollo de la fase de Prueba de Concepto fue de gran ayuda para ubicarnos en una situación de evaluación de las soluciones más adecuadas para llegar a al resultado deseado, tomando en cuenta el tipo de arquitectura utilizada, los resultados que se esperaban obtener y las repercusiones de los nuevos cambios en la arquitectura original.

La elección del método XP para el desarrollo de la aplicación fue clave para el éxito del mismo gracias a las prácticas de proceso en el diseño, en la codificación y en las pruebas, que soportaron el desarrollo rápido de una aplicación expuesta a cambios durante sus diferentes etapas, y promoviendo la comunicación y trabajo en equipo de los programadores comprometidos en el cumplimiento de las labores en cada iteración. Con este método se puede devolver a cualquier etapa sin que esto afecte al proyecto planteado y así modificarlo a conveniencia para que se obtengan los resultados esperados.

El uso de éste método nos fue muy útil ya que pudimos organizar todos los requerimientos y eventos que el cliente necesitaba para el proyecto, pudimos trabajar en pareja sin problema alguno, y siguiendo éste método se realizaron reuniones diarias con el cliente para verificar el funcionamiento del mismo y realizar cambios de ser necesario. No se siguió por completo la definición del método, ya que sólo se hicieron pruebas unitarias no formales y como historias del usuario pudimos tomar no sólo los requerimientos funcionales, sino también los no funcionales, eventos y necesidades.

El uso de Ruby on Rails agilizó considerablemente el proceso de desarrollo, ya que se aprovecharon las diversas estructuras y métodos predefinidos que tiene, de manera que si se desarrollaran con otros lenguajes, en nuestra experiencia, requerirían de un mayor esfuerzo y tiempo de codificación.

Al realizar las tareas en segundo plano permitió que la aplicación pueda realizar diversas tareas en paralelo, lo cual mejora significativamente el rendimiento y eficiencia de todo el proyecto.

El módulo de control realizado es de gran importancia para el proyecto, ya gestiona, verifica y clasifica la entrada generada por la aplicación de administración CPanel, además de guardar la información necesaria de la entrada obtenida y dar servicio de consulta a dicha aplicación Cpanel.

Para validar una URI, se creó una librería capaz de verificar si una URI es absoluta y está bien formada, para así proceder a incluirla en la aplicación, si por el contrario, no es válida, ésta se descarta del grupo de urls. Esto es de gran ayuda ya que la aplicación sólo guarda y procesa URIS válidas.

El módulo del Spider realiza su procedimiento de forma adecuada, ya que al recibir un grupo de URIS, la procesa una por una y obtiene de ella su contenido, su tipo MIME y las URLs asociadas a la URI que está siendo procesada. La salida obtenida es analizada por el robot y éste almacena los documentos en la base de datos MongoDB y genera una nueva entrada al spider con las URIs que son resultado del procesamiento anterior.

El desarrollo de este trabajo nos dio la oportunidad de conocer la bondades de utilizar la virtualización, ofreciendo la posibilidad de tener varias máquinas virtuales dentro de un mismo servidor físico funcionando concurrentemente, este punto es de suma importancia ya que permite ahorrar costos de almacenamiento físico de las máquinas, mantenimiento, y sobre todo, hardware, y así hacer posible el funcionamiento de este esquema con los recursos disponibles en el laboratorio del Centro de Computación Paralela y Distribuida.

Para poder trabajar en equipo, se usó una herramienta muy útil para el manejo de versiones de los distintos módulos realizados. Esta herramienta nos permitió un mejor trabajo en equipo, considerando que el código desarrollado era compartido y modificado por los dos integrantes del proyecto, además del cliente, quienes realizaban actualizaciones y pruebas sobre una misma versión del sistema detectando así si se afectaba el funcionamiento de piezas de código ya consolidadas.

Debido a lo afirmado en los párrafos anteriores podemos concluir que se cumplieron los objetivos planteados al comienzo de este documento, y atribuimos en gran parte el logro de estos a la planificación y organización de requerimientos al momento de realizar cambios imprevistos, y actividades necesarias para la creación de la aplicación.

# RECOMENDACIONES

El presente Trabajo Especial de Grado está basado en el desarrollo de múltiples pruebas de ensayo y error para así obtener los resultados esperados. Para conseguir dichos resultados se efectuaron cambios de esquemas, cambios de arquitectura, todo esto dependiendo de las mejoras que se consiguieron durante la búsqueda de la implementación de un módulo de control centralizado para el procesamiento de las URLs de entrada.

Para implementar las mejoras de este prototipo es necesario un previo estudio de las librerías y gemas disponibles en la actualidad y las mejoras de las utilizadas en la aplicación desarrollada para generar soluciones efectivas.

En este prototipo el procesamiento de las URLs se encuentra restringido para un subconjunto de URLs absolutas bien formadas, por lo cual se recomienda la generación de código capaz de manipular todo tipo de URLs, bien sea absolutas como relativas.

Actualmente, no se encuentra un registro histórico de las páginas cuyo tópico sea un tema relacionado con nuestro país, por lo cual aconsejamos el desarrollo de módulos capaces de distinguir y recopilar todas aquellas páginas de carácter venezolano, de manera de poder contar con nuestro propio compendio de páginas venezolanas.

# BIBLIOGRAFÍA


1. Es.wikipedia.org. Preservación Digital. Tomado 02 de mayo de 2011 de la página Web http://es.wikipedia.org/wiki/Preservaci%C3%B3n_digital

2. Preservación Digital. Tomado el 03 de mayo de 2011 de la página http://www.apuntespreservaciondigital.blogspot.com/

3. Maestros del Web. Internet Archive. Tomado el 28 de abril de 2011 de la página http://www.maestrosdelweb.com/editorial/internetarchive/

4. WayBack Machine. Tomado el 05 de mayo de 2011 de la página http://www.archive.org/

5. Sobrerailes.com. SobreRaíles. En marcha con Ruby on Rails. Tomado el 03 de marzo de 2011 de la página Web http://sobrerailes.com/pages/en_marcha_con_rails

6. Carlson, Lucas y Richardson Leonard (2007) Curso de Ruby, Procedimientos para la creación de secuencias de comandos orientadas a objetos. Madrid - España. Ediciones Anaya Multimedia.

7. Es.wikipedia.org. Ruby on Rails. Tomado el 15 de febrero de 2011 de la página Web http://es.wikipedia.org/wiki/Action_Mailer

8. Thomas D., Heinemeier D, Breedt L., Clark M., Duncan J.,Gehtland J., y Andreas Schwarz (2006). Agile Web Development with Rails: Second Edition, 2nd Edition. The Pragmatic Bookshelf, Estados Unidos

9. Tate, Bruce A. y Hibbs, Curt (2007). Ruby on Rails. Madrid - España. Ediciones Anaya Multimedia.

10. Manual RubyGems (2011). Tomado el 27 de marzo de 2011 de la página http://docs.rubygems.org/read/book/7

11. RubyGems. Tomado el 28 de marzo de 2011 de la página http://en.wikipedia.org/wiki/RubyGems

12. Introducción a los servicios web REST. Tomado el 23 de marzo de 2011 de la página http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html

13. Brando (2008). Ruby on Rails 2.1. ¿Qué hay de nuevo? Tomado el 01 de diciembre de 2010 desde http://www.improveit.com.br/en/company/tapajos

14. ActiveReource. Tomado el 10 de diciembre de 2010 de la página http://apidock.com/rails/ActiveResource/Base

15. Introducción a MongoDB. Tomado el 19 de marzo de 2011 de la página http://www.genbetadev.com/bases-de-datos/una-introduccion-a-mongodb

16. M. Ambrust, et al., "Above the Clouds: A Berkeley View of Cloud Computing," Electrical Engineering and Computer Sciences, Univers ity of California at Berkeley, California, Technical Report UCB/EECS-2009-28, 2009.

17. S. Bennett, M. Bhuller, and R. Covington, "Architectural Strategies for Cloud Computing," Oracle Corporation, 2009.

18. S. Kajeepeta, "Cloud Computing: From Metaphor to Mainstream," Software Magazine, vol. 27, no. 6, pp. 10-13, Nov. 2008.

19. G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud Computing," IBM Corporation, 2007.

20. D. Quan, "From Cloud Computing to the New Enterprise Data Center," IBM Corporation, 2008.

21. D. Thomas, "Cloud Computing - Benefits and Challenges!," Journal of Object Technology, vol. 8, no. 3, pp. 37-41, 2009.

22. Sun Microsystems, "Introduction to Cloud Computing Architecture," Sun Microsystems White Paper, 2009.

23. J. D. Lasica, Identity in the Age of Cloud Computing. United States of America: The Aspen Institute, 2009.

24. Cisco, "Private Cloud Computing for Enterprises," Cisco White Paper, 2009.

25. A. Dver, "Enemy of SaaS?," Software Magazine, vol. 27, no. 6, p. 24, Nov. 2008.

26. Cisco Systems, Inc., "The Cisco Powered Network Cloud: An Exciting Managed Services Opportunity," Cisco Systems, Inc., 2009.

27. B. Ohlman, A. Eriksson, and R. Rembarz, "What Networking of Information Can Do for Cloud Computing," in 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009, pp. 78-83.

28. A. Kamaraju and P. Nicolas, "Cloud Storage," Storage Networking Industry Association, 2009.

29. A. Mendoza, Utility Computing Technologies, Standards, and Strategies. United States of America: Artech House, Inc., 2007.

30. Workling. Tomado el 02 de mayo de 2011 de la página https://github.com/purzelrakete/workling

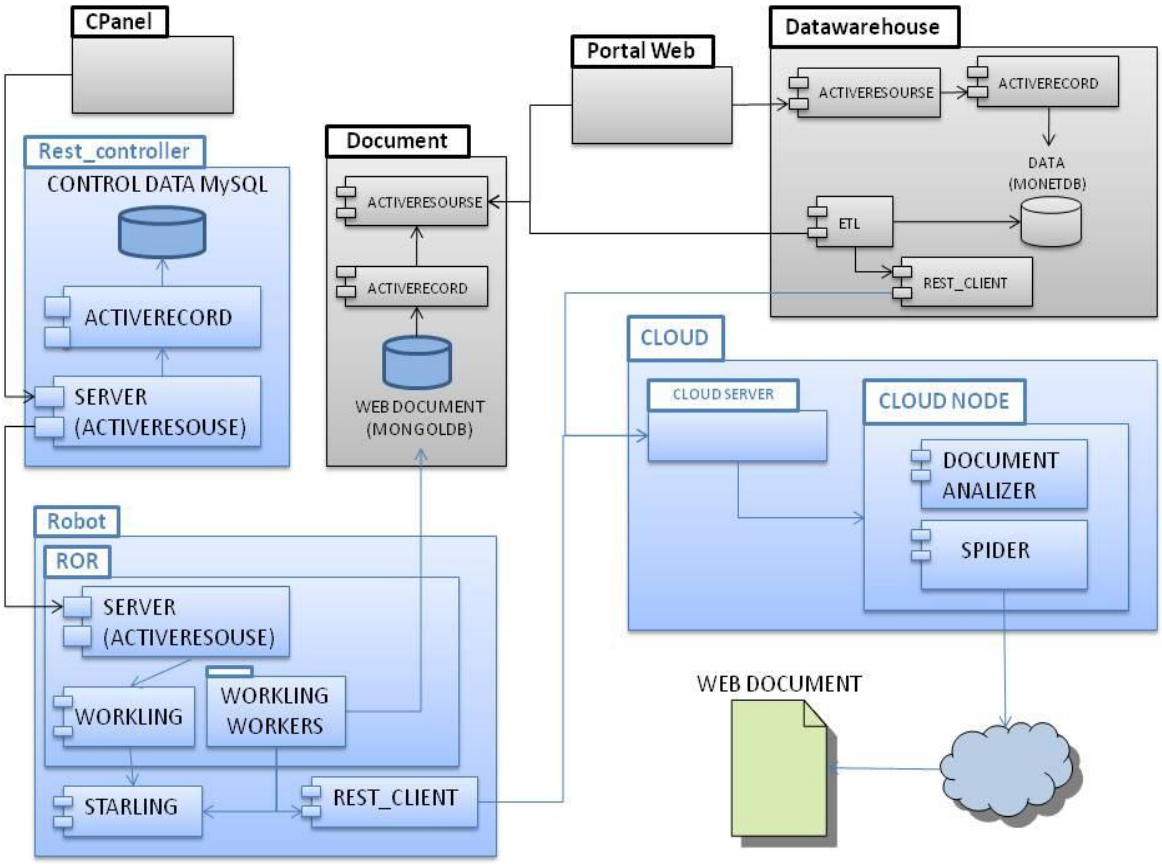
31. Pressman, R. (2007) Ingeniería del Software. Un enfoque práctico. Sexta Edición. Editorial Mc Graw Hill.

32. Jeffries, Anderson, y Hendrickson, Programación Extrema, 2000.

33. Hurtado, Julio Ariel, Bastiarrica Cecilia. Proyecto SIMEP-SW Mayo 08 de 2005, Modelo de Procesos, Calidad y Mejoramiento: CMM, TSP, PSP,ISO, IEEE, SPICE, etc

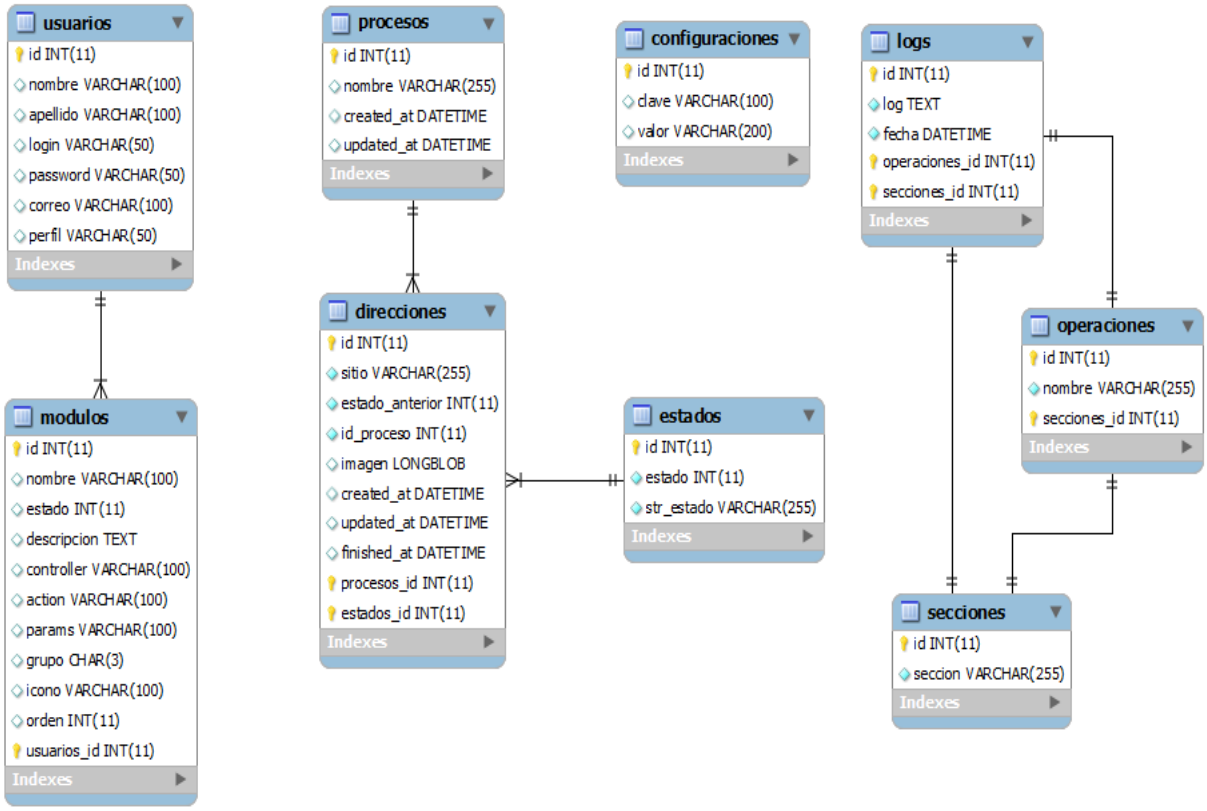
# APENDICE

## 8.1 Arquitectura General del Proyecto




En éste Trabajo Especial de Grado trabajamos en los módulos que están coloreados con azul, es decir, Rest_Controller, Robot, y Cloud y en la base de datos MongoDB en Document.

## 8.2 Modelo de la base de datos ITICVE



## 8.3 Diccionario de Datos de ITICVE

| TABLA CONFIGURACIONES | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de configuraciones |
| clave | Nombre de la configuración a agregar. Por ejemplo: ruta_imagenes |
| valor | Determina la configuración a ser utilizada. Por ejemplo: ruta del directorio de las imágenes |

| TABLA ESTADOS | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Estados |
| estado | Número asociado a cada uno de los estados |
| str_estado | Nombre de cada estado asignado a un identificador |

| TABLA DIRECCIONES | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Direcciones |
| sitio | Dirección de una url |
| estado | Id del estado que una url puede tener |
| estado_anterior | Id del estado que una url tuvo anteriormente |
| id_proceso | Identificador del proceso al que pertenece la url |
| imagen | Imagen de la vista previa de la url |
| Created_at | Fecha de creación de la url en la Base de Datos |
| Update_at | Fecha de las actualizaciones de la url en la Base de Datos |
| Finished_at | Fecha de finalización de la ejecucion de la url |

| TABLA LOGS | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| id | Identificador único de la tabla de Logs |
| operación_id | Id de la operación en donde se registró el evento asociado a la url y/o al proceso |
| sección_id | Id de la sección o modulo del proyecto en donde se registró el evento asociado a la url y/o al proceso |
| log | Evento referente a una url o a un proceso |
| fecha | Fecha en la que se registra un evento de una url o proceso |

| TABLA OPERACIONES | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Operaciones |
| Nombre | Nombre de la operación realizada (create, show, update, destroy) |

| TABLA PROCESOS | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Procesos |
| Nombre | Nombre del proceso |
| Created_at | Fecha de creación del proceso en la Base de Datos |

| TABLA SECCIONES | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Secciones |
| seccion | Nombre de una sección o módulo en el proyecto (Cpanel, Controller, Spider, etc) |

| TABLA USUARIOS | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Usuarios |
| Nombre | Nombre del administrador de Cpanel |
| Apellido | Apellido del administrador de Cpanel |
| Login | Login del administrador de Cpanel |
| Password | Clave de acceso del administrador de Cpanel |
| Correo | Correo electrónico del administrador del Cpanel |
| Perfil | Perfil de usuario Ej. SUPERUSUARIO |

| TABLA MODULOS | |
|---|---|
| ***CAMPOS*** | ***DESCRIPCIÓN*** |
| Id | Identificador único de la tabla de Módulos |
| Nombre | Nombre del módulo administrador |
| Estado | Estado en la que se encuentra un proceso |
| Descripción | Descripción de cada uno de los módulos que se encuentran en el panel del administración |
| Controller | Controlador al cual apunte el módulo para acceder directamente |
| Action | Acción del controlador para acceder a la funcionalidad requerida |
| Params | Parámetros requeridos por el controlador para las distintas acciones a procesar |
| Grupo | Grupo al cual pertenece el módulo dentro de las categorías: GDD, PRO, ADM |
| Icono | Imagen que representa la funcionalidad de un módulo determinado |
| orden | Orden en la que aparece el módulo en la categoría que pertenece |