



CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL
INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE COMPUTACIÓN

Administrador de recursos compartidos y distribuidos para trabajo colaborativo en la Web

Tesis que presenta

José Luis Ortigosa Flores

para obtener el grado de

Maestro en Ciencias en Computación

Directores de tesis:

Dra. Sonia Guadalupe Mendoza Chapa

Dr. Dominique Decouchant

Ciudad de México

Diciembre de 2017

Resumen

En un principio la Web fue creada con dos propósitos: por un lado la compartición de información y por el otro la interconexión de tal información. Hoy en día la Web es mucho más que eso. Hoy es posible crear aplicaciones web que pueden ejecutarse en cualquier dispositivo: desde celulares, hasta computadoras, pasando por *tablets* y televisiones inteligentes siempre que cuenten con un navegador web. Las aplicaciones web proporcionan a los usuarios de la Web medios para comunicarse con otras personas, medios para publicar información (recursos) en cualquier formato desde simple texto hasta imágenes o videos. Hoy por hoy podemos encontrar aplicaciones Web de cualquier tipo tales como editores de texto, hasta aplicaciones más complejas como lo son las aplicaciones de diseño 3D. La Web ha sido un punto importante para el cómputo ubicuo debido a que no importa donde se encuentre la persona o que dispositivo utilice, esta tendrá acceso a su información y aplicaciones. Lo anterior es posible debido a que la Web cuenta con una gran variedad de tecnologías desarrolladas como lo es XML o JSON para empaquetar datos, JavaScript para desarrollar aplicaciones sobre los navegadores, CSS para dar formatos complejos a los datos y muchos otros orientados a diferentes tópicos tales como la transmisión de audio o la creación de imágenes vectoriales, etc. De modo que se puede decir que la Web se ha convertido en una plataforma homogénea para desarrollar aplicaciones. Aun cuando la evolución de la Web es un aspecto positivo se debe decir que sus recursos están interrelacionados por medio de hipervínculos, donde es difícil determinar si un recurso está siendo referenciado por algún otro y que genera inevitablemente referencias hacia recursos no existentes debido a limitaciones en las URLs. Este problema es conocido como hipervínculos rotos el cual a la fecha de esta tesis no cuenta con una solución. A pesar de lo anterior este problema es solucionado parcialmente en sistemas encargados de la gestión de sitios web, conocidos como administradores de contenido web, sin embargo en estos sistemas se centraliza tanto el manejo como la persistencia de recursos, provocando algunos problemas asociados a cuellos de botella. Para aprovechar que la Web ofrece un entorno idóneo para crear aplicaciones, se pretende crear un *framework* que aproveche las ventajas de la Web para facilitar el desarrollo de aplicaciones web colaborativas. El *framework* usará la técnica empleada en los administradores de contenido web para hacer frente a los hipervínculos rotos en conjunción de una propuesta de técnica para separar la doble función de las URLs con respecto a la ubicación e identificación de recursos, además se tomara a manera de especificación las características que ofrece la plataforma PIÑAS y el editor colaborativo Alliance web, los cuales fomentan el uso de la replicación y el modo de interacción asíncrono para el trabajo colaborativo.

Abstract

At the beginning the Web was created having two goals: the first one was to share information and the second one was to interconnect it. Nowadays the Web is much more than that. Today we can create web application, which can be executed on every device: from cell phones to computers or Smart TVs including tablets, whenever they have a web explorer. Web applications provide means for users to communicate each other's, means for publishing information (resources) in every format from simple text to images or videos. Today we can find web applications for any propose such as text editors to complex applications like 3D design applications. Web has been an important part for ubiquitous computation because no matter where you are or kind of device you use, you will have access to your information and applications. The above is possible because web has a huge variety of developed technologies such as XML or JSON to transmit information, JavaScript for developing applications for browsers, or CSS for complex formatting data, and many others technologies for diverse purpose such as passing audio or for creating vector images, etc. So we can say that Web has been converted in a homogeneous platform for developing applications. Even when Web evolution has been positive it has to be said that its resources are related by means of links where it is difficult to determine if a resource is being referenced by other one, which generate references to no existing resources because of URLs limitations. The problem is known as broken links which until now it's not a solving problem. Despite the above, broken links problem has been partially solved by a type of software known as web content management system, which were created to managed web sites. This kind of systems centralized the management and persistence of resources which generate bottlenecks. So to take advantage of web features for developing applications. It's proposed to create a framework to ease development of collaborative web applications. This framework will utilize the technique used by the web content management systems to face broken links problem in conjunction with a proposal for separating the URLs double function related to location and identification of resources. In addition, it will be taken features from PIÑAS platform and Alliance collaborative web editor as specifications for using replication and asynchronous interaction for providing collaborative work.

Agradecimientos

Agradecimientos especiales al CONACyT al CINVESTAV por permitirme realizar mis estudios de maestría.

También agradecimientos especiales a mis directores de tesis la *Dra. Sonia Mendoza Chapa* y al *Dr. Dominique Decouchant* por todo su apoyo.

Otros agradecimientos especiales a mis revisores el *Dr. Amilcar Meneses Viveros* y al *Dr. José Guadalupe Rodríguez García*.

Y finalmente también agradecimientos para todos mis profesores y al personal de apoyo administrativo.

Índice general

Resumen	I
Abstract	III
Agradecimientos	V
Índice general	VII
1. Introducción	1
1.1. Motivación	1
1.2. Planteamiento del problema	2
1.3. Objetivos	2
1.3.1. Objetivo general	2
1.3.2. Objetivos particulares	2
1.4. Organización del documento	3
2. Marco teórico	5
3. Estado del arte	9
3.1. Hipervínculos rotos	9
3.2. GroupKit	10
3.3. <i>Framework</i> DISCIPLE	11
3.4. <i>Framework</i> ANTS	11
3.5. COPSE-Web	12
3.6. Administrador de contenido web	12
4. Antecedentes	15
4.1. Plataforma PIÑAS	15
4.1.1. Arquitectura Distribuida	15
4.1.2. Extensiones de PIÑAS	18
4.1.3. Sitios de almacenamiento	19
4.2. Alliance	20

5. Análisis y solución al problema de los hipervínculos rotos	25
5.1. Análisis detallado del problema de las hipervínculos rotos	25
5.1.1. Descripción de un escenario común	26
5.1.2. Detalles técnicos del problema	32
5.2. Solución al problema de los hipervínculos rotos	37
5.2.1. Seguimiento de recursos referenciados	43
6. Desarrollo e implementación	45
6.1. Selección de tecnología	45
6.2. Metodología de desarrollar del sistema	47
6.3. Requerimientos del <i>Framework</i>	48
6.4. Arquitectura	49
6.5. Análisis y diseño del <i>framework</i>	51
6.6. Diagrama de clases	59
6.7. Proceso de compartición y edición de documentos	60
6.8. Envío de información	62
6.9. Interfaz	63
7. Pruebas	67
7.1. Características y objetivo de las pruebas	67
7.2. Desarrollo de la aplicación de prueba	68
7.2.1. Registrar colaborador	71
7.2.2. Ingreso a la aplicación de prueba	72
7.2.3. Crear un documento compartido	74
7.2.4. Recepción de solicitudes de colaboración	76
7.2.5. Definición de áreas de trabajo	77
7.2.6. Registrar recursos	80
7.2.7. Procesos de creación de documento compartido en la aplicación de prueba	80
7.3. Distribución de recursos	88
7.4. Múltiples URLs apuntando a un recurso	89
7.5. URLs para acceder a los recursos	91
8. Conclusiones y trabajo futuro	95
8.1. Conclusiones	95
8.2. Trabajo futuro	96
Bibliografía	99

Capítulo 1

Introducción

1.1. Motivación

La Web ofrece un entorno de desarrollo de aplicaciones que homogeniza las diferencias que existen entre sistemas operativos y arquitecturas de máquina. Las aplicaciones web tiene la particularidad de poderse ejecutar en cualquier dispositivo, desde teléfonos inteligentes hasta televisiones, pasando por *tablets* y computadoras de escritorio o portátiles. Otra ventaja que nos ofrece la web es que todos los recursos que son publicados en este medio pueden ser accesibles desde cualquier parte del mundo, esta característica se extiende a las aplicaciones web al estar disponibles sin importar el punto geográfico donde se encuentren sus usuarios. La manera de acceder a recursos y aplicaciones es por medio de URLs, lo que facilita interrelacionar información, recursos y aplicaciones.

La ubicuidad de las aplicaciones y recursos se debe a que la Web se apoya en tecnologías ampliamente usadas y difundidas (e. g., HTTP, HTML, XML, CSS, JavaScript, URL, etc.). La Web ha evolucionado de un lugar para publicar y relacionar información a una fuente de información, recursos y aplicaciones. Todas las características de la Web pueden ser aprovechadas para el desarrollo de aplicaciones colaborativas.

Se han desarrollado aplicaciones colaborativas dependientes del sistema operativo (e.g., [GroupKit, 2016]), o de tecnologías de desarrollo de software que requieren de personal altamente especializado, lo que dificulta su amplio uso, difusión y mantenimiento, por otro lado la Web ofrece un conjunto de tecnologías estándar y ampliamente conocidas lo que facilita su mantenimiento. Sin embargo hay un problema con las URLs, a pesar de ser excelente medio para referenciar recursos estas no presentan mecanismos para preservar la integridad de los enlaces lo que ha desembocado en el problema conocido como hipervínculos rotos (*broken links*), el problema de los hipervínculos rotos radica en que las URLs tienen una doble funcionalidad, por un lado sirven para identificar a un recurso y por el otro sirven para localizarlo, cualquier cambio en una de estas propiedades del recurso termina con la invalidación de URLs y por tanto una referencia rota.

Ninguna aplicación colaborativa se parece a otra y normalmente la creación de las mismas implica un diseño específico para un grupo de trabajo determinado, apesaro de lo anterior las aplicaciones colaborativas comparten determinadas características que han sido encapsuladas en *frameworks* y/o bibliotecas. Existen bibliotecas y *frameworks* que facilitan el desarrollo de aplicaciones web colaborativas [Lopez and Skarmeta, 2003, David and Borges, 2002] pero que han sido desarrolladas usando tecnologías que ya desaparecieron o están en desuso. Por lo que es importante impulsar el desarrollo de *frameworks* o bibliotecas que usen los estándares de la Web para facilitar el desarrollo de aplicaciones web colaborativas.

1.2. Planteamiento del problema

No existe software moderno para el desarrollo de aplicaciones web colaborativas que basen su comunicación totalmente en HTTP y que utilicen tecnologías estándar de la Web¹. Además de que los *frameworks*, existentes en la literatura, no aprovechan el uso de las URLs para referenciar recursos. Aun cuando las URLs ofrecen un medio estándar para hacer referencia a recursos distribuidos en la Web, estas no cuentan con mecanismo que aseguren la integridad de las referencias, lo que lleva al problema conocido como hipervínculos rotos (*broken links*).

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar e implementar un *framework* administrador de contenido distribuido y compartido en la Web, basado en la plataforma PIÑAS, que proporcione solución al problema de los hipervínculos rotos para referenciar recursos usando URLs y que de soporte el trabajo asíncrona.

1.3.2. Objetivos particulares

- Investigar y analizar el problema de los hipervínculos rotos (broken links) para entender sus retos, implicaciones y soluciones actuales.
- Analizar la plataforma PIÑAS y Alliance Web con el fin de extraer las técnicas empleadas para soportar el trabajo colaborativo.
- Diseñar e implementar un *framework* administrador de recursos compartidos y distribuidos en la Web que facilite la creación de aplicaciones web colaborativas.
- Probar el *framework* mediante el desarrollo de una aplicación web que use al software como base su implementación para analizar sus ventajas ante el desarrollo de aplicaciones colaborativas.

¹Los *frameworks* más recientes son de inicios del año 2000 y usan como medio principal de comunicación a CORBA, RMI o JMS. Algunos *frameworks* dan soporte a *Applets*, tecnología ya descontinuada actualmente

1.4. Organización del documento

El documento se encuentra dividido en 8 Capítulos (ver Figura 1.1). En el Capítulo 2 se presentan los conceptos y definiciones que se usan a lo largo de la tesis.

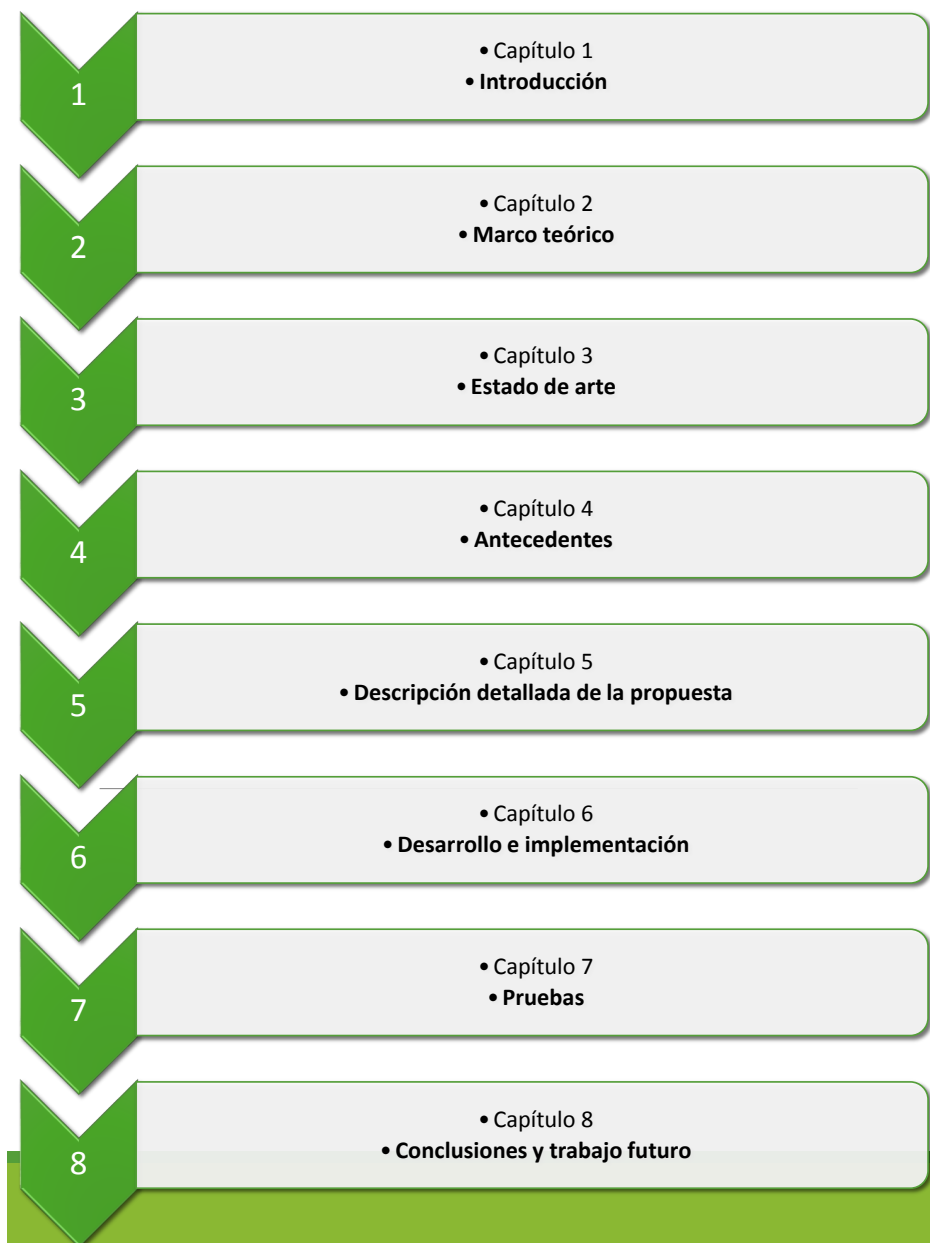


Figura 1.1: Diagrama que muestra la organización del contenido de la tesis.

En el Capítulo 3 se exponen las tecnologías que dan solución al problema de hipervínculos rotos, así como trabajos relacionados que se utilizan de manera profesional para gestionar sitios web de grandes proporciones y con gran cantidad de contenido en el que hay miles de

usuarios involucrados en su gestión y mantenimiento. Además, se presentan soluciones que podrían verse relacionadas con el objetivo de esta tesis, tales como los almacenamientos en la nube proporcionados por algunas empresas.

En el Capítulo 4 se presenta los detalles del trabajo realizado en el grupo de investigación de la directora y codirector de tesis. En este caso se presenta a la plataforma PIÑAS y al editor colaborativo Alliance, los cuales son la base y corazón de esta tesis.

El Capítulo 5 presenta el análisis detallado de los elementos involucrados en el problema de los hipervínculos rotos. Posteriormente se presenta la manera en la que se da solución al problema. Y se finaliza exponiendo las ideas que están detrás de la implementación del *framework*.

En el Capítulo 6 se muestra el proceso para desarrollar el *framework* que permitirá la compartición de recursos que están distribuidos en la Web. El software desarrollado es capaz de ayudar a implementar dos aspectos importantes de todo sistema colaborativo como lo es la colaboración y la coordinación.

En el Capítulo 7 se presenta la creación de una aplicación, la cual usa el *framework* como base para exponer su uso y puntos de oportunidad a la hora de involucrarse en la creación de aplicaciones web colaborativas. La aplicación de prueba se enfoca en mostrar que el *framework* trabaja sobre HTTP/XML y que facilita el desarrollo de aplicaciones colaborativas.

Finalmente en el Capítulo 8 se presentan una serie de observaciones al *framework* creado lo que incluye cuestiones relacionadas a su impacto, sus alcances y restricciones, así como posibles puntos de mejora y comentarios finales.

Capítulo 2

Marco teórico

En este capítulo se presentan algunos conceptos importantes que se manejan a lo largo de la tesis.

Sistema colaborativo

La definición que a continuación se presenta es tomada de [Koch and Gross, 2006]. Las aplicaciones o sistemas colaborativas (*Groupware*) son sistemas encargados de apoyar a grupos de personas en cuatro aspectos importantes que son los siguientes:

- **Comunicación:** la comunicación se refiere a que el sistema debe de permitir que cualquier persona miembro del grupo pueda comunicarse con cualquier otra y en cualquier momento que lo necesite.
- **Colaboración:** la colaboración se refiere a que las personas que colaboran para un fin tienen tareas que contribuyen al objetivo del grupo de trabajo, cada una de las tareas asignadas a los integrantes es importante e impacta en el progreso del trabajo, no hay tareas aisladas.
- **Coordinación:** la coordinación se refiere a que el sistema debe proporcionar mecanismo que coordine los esfuerzos realizados por los integrantes del grupo.
- **Conciencia de grupo:** este término se refiere a que incluso cuando los integrantes se encuentren geográficamente distantes, el sistema debe proporcionar mecanismos que les permita a las personas percibir el progreso de las tareas asignadas a cada uno de los miembros del grupo, así como el progreso general del trabajo grupal.

Cuando un sistema cumple todos los aspectos listados entonces ese sistema se considera colaborativo.

Framework

Para presentar una definición de este término se usa la que se encuentra en [Riehle, 2000]. Un *framework* es una aplicación semi-implementada que modela cualquier dominio, por ejemplo la conversión de entidades del paradigma orientado a objetos, a entidades del modelo relacional, este tipo de herramientas es conocida como *ORM*¹.

Un *framework* encapsula fragmentos de código y diseño que son reusables y generalizables. Normalmente un *framework* define de forma parcial o completa el flujo de ejecución de una aplicación. Los *frameworks* trabajan con objetos abstractos de los cuales se debe heredar para que sean controlados por el objeto controlador. Por ejemplo JPA² es un *framework* de tipo *ORM*. Aquí el objeto que lleva el control de la ejecución de la aplicación es conocida como `EntityManager` este objeto trabaja con un tipo especial de entidades conocidos como `Entities`, los cuales son objetos que son susceptibles de ser almacenados, borrados o actualizados en una base de datos.

Middleware

La definición es tomada de la siguiente fuente [Krakowiak, 2007]. Capa de software que se encuentra por encima de los sistemas operativos y de los protocolos de comunicación para realizar las siguientes funciones:

- Ocultamiento de elementos distribuidos: oculta la interacción con partes que están localizadas en diferentes lugares.
- Ocultamiento de la heterogeneidad: oculta las diferencias existentes entre los sistemas operativos, hardware y protocolos de comunicación.
- Proporcionar uniformidad: proporciona una única interfaz para los desarrolladores con el fin de que se facilite la interoperabilidad.
- Proporcionar servicios comunes: debe proporcionar funciones de propósito general con el fin de evitar la duplicación de esfuerzos.

Un *middleware* debe proporcionar independencia de plataforma, y ocultamiento de detalles de bajo nivel.

REST

La definición de REST es tomada de la siguiente fuente [Sandoval, 2009]. El termino REST viene de la tesis doctoral del Dr. Roy Fielding publicada en el año 2000, y significa

¹Object-Relational Mapping

²Java Persistence API

REpresentational State Transfer. REST es un conjunto de condiciones que cuando se aplican al diseño de un sistema crean un estilo de arquitectura de software. El conjunto de condiciones de REST son el resultado de evaluar todos los recursos de red y tecnologías disponibles para crear aplicaciones distribuidas. El conjunto de condiciones son las siguientes:

- Debe ser un sistema cliente-servidor.
- Debe ser un servicio sin estado.
- La infraestructura de red debe soportar diferentes niveles de cache.
- Cada recurso debe accederse de manera uniforme por medio de una única dirección.
- Debe ser escalable.

La Web cumple con todas las condiciones planteadas por REST. Un recurso RESTful es cualquier cosa que pueda ser asociada a una dirección y que pueden ser transferidos de un servidor a un cliente. En RESTful cada recurso puede tener múltiples representaciones, por ejemplo un recurso puede tener la forma de texto, al mismo que tiene una en la forma de XML o de una imagen. En RESTful una URI o URL es un hipervínculo a un recurso y esta es la única manera de intercambiar representaciones. Bajo RESTful las URLs no pueden cambiar con el paso del tiempo. REST usa el protocolo HTTP para crear, eliminar obtener o actualizar recursos, tales operaciones están relacionadas con los métodos del protocolo de la siguiente manera:

- Un recurso puede ser creado usando el método POST del protocolo HTTP.
- Un recurso poder ser eliminado usando el método DELETE del protocolo HTTP.
- Un recurso puede ser actualizado por medio del método PUT del protocolo HTTP.
- Se puede obtener un recurso por medio del método GET del protocolo HTTP.

Capítulo 3

Estado del arte

En este capítulo se presentan los trabajos más relevantes relacionados con el problema (descrito en la Sección 1.2). La primera sección 3.1 se encarga de describir los trabajos relacionados al problema de los hipervínculos rotos, para posteriormente exponer en las secciones 3.2, 3.3, 3.4 y 3.5 los *frameworks* orientados a facilitar el desarrollo de aplicaciones web colaborativas y se finaliza el capítulo con la sección 3.6 hablando de la manera en la que le hacen frente los desarrolladores de sitios web.

3.1. Hipervínculos rotos

El problema de los hipervínculos rotos (*broken links*) ha sido considerado un problema serio en el contexto de la Web. Hasta el momento se han desarrollado varios sistemas que no son lo suficientemente maduros, algunos sistemas implican el monitorear el estado de los hipervínculos¹. Tales sistemas consisten en registrar las URL pertenecientes a recurso de interés, después checar si el recurso existe o no para después notificarle al interesado [Glass et al., 2001]. Algunos ejemplos de sistemas que verifican que los hipervínculos en las páginas sean correctos son: Xenu's Sleuth o W3C link checker [Martinez-Romo and Araujo, 2012].

Los hipervínculos rotos pueden deberse a que la página a la cual apuntaba ha desaparecido permanentemente, pero en otros muchos casos puede ser que la página ha sido reubicada dentro del mismo sitio o en otro, otras veces la página ha sido actualizada por lo que ha cambiado un poco [Martinez-Romo and Araujo, 2012].

Por otra parte en [Haslhofer and Popitsch, 2009] [Popitsch and Haslhofer, 2010] [Popitsch and Haslhofer, 2011] se indica que el problema de los hipervínculos rotos (*broken links*) es un gran reto y que además es un problema que se debe resolver para dar paso a la Web de los datos ligados (Web en la que información similar y relacionada se encuentre ligada y que está pueda ser leída por personas y máquinas). En [Popitsch and Haslhofer, 2010]

¹Identificar si la URL apunta a un recurso existente

describen el problema de los hipervínculos rotos y proponen un *framework* llamado *DSNotify* que implementa un asistente para reparar hipervínculos rotos mediante el uso de heurísticas.

Un ejemplo de sistema es el presentado en [Morishima et al., 2008] presentan el proyecto WISH (Web Integrity Management by self-Healing), el cual es un sistema que se encarga de monitorear de manera constante los hipervínculos entre páginas, donde su principal objetivo es reparar los hipervínculos rotos a causa de que fueron reubicadas las páginas. Este proyecto propone el concepto de *link authority* que es una página p la cual es apuntada desde otra página q , donde la página q siempre está actualizada. Este sistema se encarga de encontrar los *links authorities* y de explorar más hipervínculos si se detectan hipervínculos rotos, aunque se debe mencionar que no hace una búsqueda exhaustiva, ya que maneja un conjunto seis heurísticas.

Otro ejemplo es el que encontramos en [Martinez-Romo and Araujo, 2012], en este artículo nos presentan la situación en la que uno está navegando y encuentra un link roto pero que se desea ver si aún existe su información en alguna parte del sitio o incluso si existe algún sitio que tenga un contenido similar al que apuntaba el hipervínculo roto. En este trabajo presentan un sistema que propone candidatos de páginas web, para reparar el hipervínculo roto. Este sistema recibe como entrada una página web la cual explora en busca de hipervínculos. Una vez que detecta hipervínculos rotos, el sistema determina si tiene suficiente información como para hacer recomendaciones, si este es el caso el sistema proporciona un conjunto de páginas candidatas para reemplazar al hipervínculo roto. Este sistema se basa en extraer información relevante de la página tal como terminología relevante y clasificación de resultados.

Si el problema de los hipervínculos rotos en el contexto de las páginas web es un gran reto, este lo es más cuando se trata de la nube de LOD (*Linked Open Data*), la cual es un conjunto de bases de datos de varios dominios que han sido trasladados a RDF², y ligados a otros conjuntos de datos por medio de *links RDF*. Si en este contexto se da el problema de los hipervínculos rotos ésto causaría que los datos se dividieran y que llegara a darse el caso de áreas de datos desconectas [Rajabi et al., 2014, Popitsch and Haslhofer, 2010].

3.2. GroupKit

GroupKit es una biblioteca que facilita la creación de aplicaciones colaborativas en tiempo real, algunos ejemplos de aplicaciones que soporta son ([GroupKit, 2016]):

- Herramientas de dibujo con soporte para varios usuarios.
- Soporte para editores de texto.
- Herramientas para conversación, etc.

²RDF significa Resource Description *framework* la cual es una tecnología basada en XML para agregar metadatos a la información así como de proveer la manera de crear relaciones entre los datos.

Esta biblioteca fue desarrollada por la universidad de Calgary la cual se apoya fuertemente de c++/RPC³, donde la última actualización registrada fue del año 2003, en su página se publica que esta biblioteca sólo tiene un valor histórico debido a los cambios tecnológicos.

La biblioteca tiene como principal objeto compartido de datos a lo que ellos llaman *environment* donde su principal característica es que cuenta con gestión de consistencia colaborativa [Lopez and Skarmeta, 2003].

3.3. *Framework* DISCIPLE

DISCIPLE⁴ ofrece solución a dos problemas por un lado proporciona un conjunto de herramientas para desarrollar aplicaciones colaborativas y un *framework* para compartir aplicaciones monousuario, todo en el contexto de la aplicaciones JavaBeans como lo son los Applets [Marsic, 1999].

El principal componente de este *framework* es llamado bus de colaboración el cual involucra componentes para acoplar aplicaciones, componentes para el control de concurrencia, un componente dedicado a la replicación de eventos y un componente encargado de la conciencia de grupo. El *framework* da soporte a las aplicaciones síncronas [Marsic, 1999].

3.4. *Framework* ANTS

ANTS es un *framework* que proporciona servicios genéricos para el trabajo colaborativo, el cual internamente cuenta con un *middleware* llamado JMS que se apoya de una arquitectura centralizada y basada en replicación. El *framework* cuenta con tres capas principales que son: la capa de aplicación, la de CSCW y la capa de tecnología.

ANTS se centra en proporcionar cuatro servicios que son los siguientes: sesiones compartidas, soporte para componentes síncronos y asíncronos, coordinación y conciencia. En donde una sesión es un grupo de objetos asociados a un patrón de comunicación, la cual forma las bases de una interacción compartida en un contexto remoto. Uno de sus componentes más importantes es el de políticas de coordinación, tales como reglas de acceso, de concurrencia, así como de flujos de control. Un último componente destacado es el de la conciencia el cual proporciona información de las actividades los colaboradores.

El *framework* es dirigido a programadores del lenguaje Java, sin embargo, en su trabajo futuro se proponen que el sistema cuente con una interfaz menos dependiente del lenguaje por lo que proponen el uso de XML y Servicios Web.

³*Remote Procedure Call*

⁴*Distributed System for Collaborative Information Processing and Learning*

En la búsqueda por bibliotecas, conjuntos de herramientas o *frameworks* orientados al desarrollo de aplicaciones colaborativas no se encontró trabajo más reciente que el de [Lopez and Skarmeta, 2003], sin embargo, ANTS es un proyecto muy ambicioso que intenta cubrir tres características importantes de las aplicaciones colaborativas que son: la coordinación (objetos síncronos y asíncronos y flujos de trabajo), la colaboración (al proporcionar conciencia de grupo) y por supuesto conciencia de colaboración.

La implementación de este *framework* usa como medio de comunicación *Java Message Service*, el cual es un servicio de notificación basado en el patrón de suscriptor-publicador que ofrece la especificación de Java EE 2⁵.

3.5. COPSE-Web

Framework desarrollado para soportar la interacción síncrona y asíncrona. La interacción síncrona se implementa por medio de *Applets* y la parte asíncrona por medio *JSP*⁶s y un conjunto de *Servlets*. Está basado en la memoria de grupo, la cual almacena cada operación, actividad y artefacto producido por los colaboradores. Esta memoria puede ser compartida y distribuida.

El *framework* se basa en un servicio de directorio para almacenar la información de los usuarios y así proporcionar un único punto de acceso. El framework proporciona conciencia de grupo basado en la memoria de grupo.

Este *framework* es centralizado no se especifican los mecanismos que usan para un entorno de aplicaciones colaborativas distribuidas.

3.6. Administrador de contenido web

En los inicios de la Web, todo aquel que quería publicar su contenido en este medio dependía de los así llamados *webmasters*. Los cuales básicamente se encargaban de la creación, edición y publicación de páginas Web estáticas (lo que significa que, sin importar quién consultara la información está sería la misma para todos) [McKeever, 2003]. Bajo este ambiente, en el que las páginas web consistían de texto estático, de un número limitado de imágenes y de unos cuantos hipervínculos, es donde se empieza a apreciar una serie de problemas muy comunes entre los que encontramos: pobre código HTML, tablas rotas (falta de etiquetas que de cierre de fila o celda), hipervínculos rotos, imágenes faltantes y una baja calidad en el contenido, debido en gran medida a que toda la responsabilidad recaía en las habilidades del *webmasters* [McKeever, 2003].

⁵Java Enterprise Edition 2

⁶Java Server Pages

Al pasar de los años, a finales de los noventas, la complejidad de los sitios web se incrementó. Por un lado aumentó el volumen del contenido a ser publicado, las páginas web evolucionaron para presentar contenido dinámico, el número de visitantes se incrementó, se debía dar soporte a una gran variedad de software (complementos en los navegadores) y hardware (variedad de dispositivos) [McKeever, 2003]. Muy pronto los errores comunes en las páginas web dejaron de ser aceptables [Powel and Gill, 2003, McKeever, 2003].

Los sistemas Administradores de Contenido Web⁷ comenzaron a ganar terreno, debido a que entre sus principales objetivos se encuentra la separación de la creación del contenido del de su formato y publicación [Powel and Gill, 2003, McKeever, 2003], pero también porque ofrecen ayuda con los problemas descritos en la Sección 1.2; ejemplos de este tipo de sistemas son Drupal u OpenCMS [Buytaert, 2016, Alkacon, 2016].

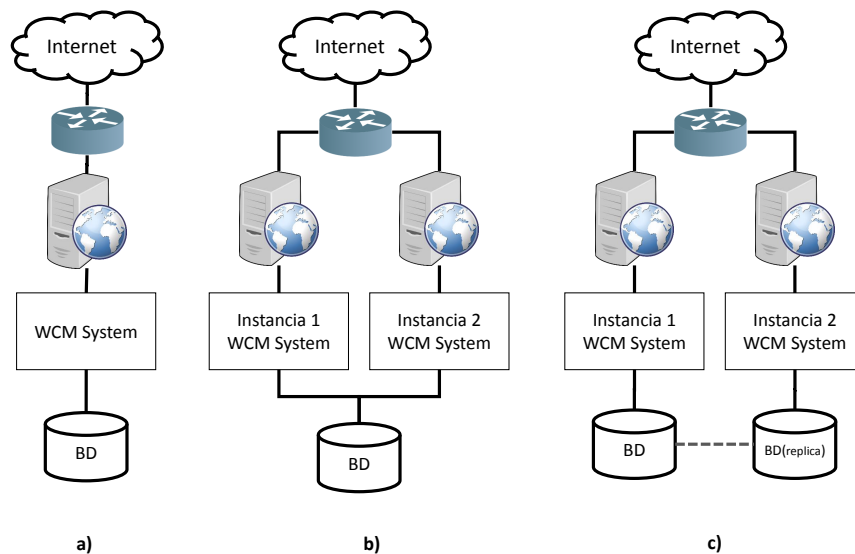


Figura 3.1: a) Esquema que muestra una instalación común de un sistema WCM; b) Esquema que muestra dos instancias del mismo sistema WCM, con el fin de mejorar el rendimiento del servicio, pero con un cuello de botella al dejar que ambas instancias accedan a la misma base de datos; c) Esquema que muestra la situación del inciso b) con el uso de una réplica de la base de datos, con el fin de solucionar problemas de cuello de botella.

La idea general de este tipo de sistemas radica en almacenar todos los recursos (contenido) en una base de datos, junto con información extra tal como: las relaciones que existen entre los recursos, usuarios y permisos, además de datos referentes al formato y distribución del contenido para su posterior presentación [Liliedahl, 2008, Tomlinson, 2010]. Esta información es usada para dar solución a los problemas antes mencionados al incorporar funciones tales como la comprobación automática de los hipervínculos de las páginas web [McKeever, 2003].

⁷ WCM Systems, del inglés para *Web Content Management Systems*

Un ejemplo de este tipo de sistemas es OpenCMS, el cual incluye entre sus varios componentes un Motor de Relaciones de Contenido, el cual se encarga de mantener a los hipervínculos internos intactos ante operaciones de cambio de nombre o cambios en su ubicación. Además se encarga de llevar el registro de las relaciones entre los distintos recursos para que, en el caso de que un usuario intente borrar un recurso, el sistema le reporte los recursos que están dependiendo del mismo, y tome decisiones al respecto [Alkacon, 2016, Liliedahl, 2008].

A pesar de que estos sistemas dan una solución a los problemas, nos encontramos con ciertas limitaciones al momento de usarlos para dar soporte al trabajo colaborativo. Una de estas limitaciones es que se puede observar que la información está concentrada o centralizada en una base de datos, lo que representa un cuello de botella para el acceso de la información (ver Figura 3.1a. Esta limitación es enfrentada por estos sistemas al incorporar mecanismos de replicación ya sea por medio del Administrador de Bases de Datos con su soporte para replicación ⁸ o bien por medio de componentes adicionales de los Sistemas Administradores de Contenido Web, los cuales permiten la creación de nuevas instancias del Administrador de Contenido y ayudan con la replicación de la base de datos⁹. En cualquier caso, el objetivo de la replicación es el de incrementar el rendimiento del servicio al crear nuevas instancias, tanto del sistema como de la base de datos, para posteriormente dejarle a un dispositivo hardware que se encargue del balanceo de cargas (ver Figura 3.1b y 3.1c).

⁸MySQL y Oracle Database son administradores de bases de datos frecuentemente compatibles con varios WCM Systems, los cuales soportan replicación de bases de datos [Oracle, 2016a, Oracle, 2016b]

⁹En OpenCMS encontramos Alkacon OCEE Cluster Package y en Drupal al módulo Drupal Replication [Alkacon, 2016, Buytaert, 2016]

Capítulo 4

Antecedentes

En este capítulo se presentan dos trabajos importantes y fundamentales que sirven de simientes para el trabajo realizado en esta tesis. Los trabajos descritos son la plataforma PIÑAS [Decouchant et al., 2008] (ver Sección 4.1) y Alliance [Salcedo and Decouchant, 1997] (ver Sección 4.2).

4.1. Plataforma PIÑAS

En esta sección se presentan los aspectos conceptuales más importantes ofrecidos por esta plataforma, resaltando su arquitectura (ver Subsección 4.1.1) y extensiones (ver Subsección 4.1.2).

4.1.1. Arquitectura Distribuida

Una arquitectura distribuida define las maneras en la que los componentes de un sistema colaborativos son distribuidos entre los sitios participantes. Se identifican dos arquitecturas para soportar el trabajo colaborativo, las cuales son:

- La arquitectura totalmente replicada que involucra crear copias de documentos compartidos entre todos los sitios de los coautores. Esta solución es muy adecuada para el trabajo colaborativo autónomo y desconectado.
- La arquitectura híbrida consiste en cargar documentos desde el servidor, para luego hacer modificaciones de manera local y posteriormente regresarlos al servidor.

Estas arquitecturas son las bases del principio de autor multi-sitio propuesto por la Plataforma PIÑAS [Decouchant et al., 2008] para facilitar la colaboración entre usuarios móviles. Ya que lo más importante es facilitar el trabajo en grupo para todo sistema colaborativos, este principio establece una organización de sitios que permite a los colaboradores beneficiarse de una alta disponibilidad de las colecciones compartidas, a pesar de su uso en entornos sujetos a posibles desconexiones.

Para especificar los sitios involucrados en una sesión colaborativa, el principio autor multi-sitio distingue entre sitios de trabajo y sitios de almacenamiento. Un sitio de trabajo ejecuta la aplicación colaborativa PIÑAS que permite a los colaboradores consultar y modificar colecciones. Un sitio de almacenamiento tiene al menos un servidor Web. El principio de autor multi-sitio no agrega restricciones al rol de cada sitio, ya que un sitio de trabajo o almacenamiento puede ser compartido por varios colaboradores y un sitio puede, de forma simultánea actuar como un sitio de trabajo y de almacenamiento (ver Figura 4.1), con el fin de facilitar el trabajo autónomo y desconectado.

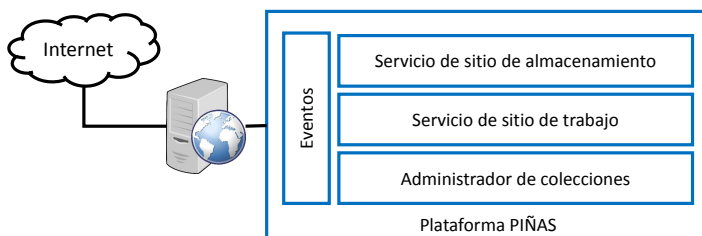


Figura 4.1: Principales componentes que ofrece la plataforma PIÑAS

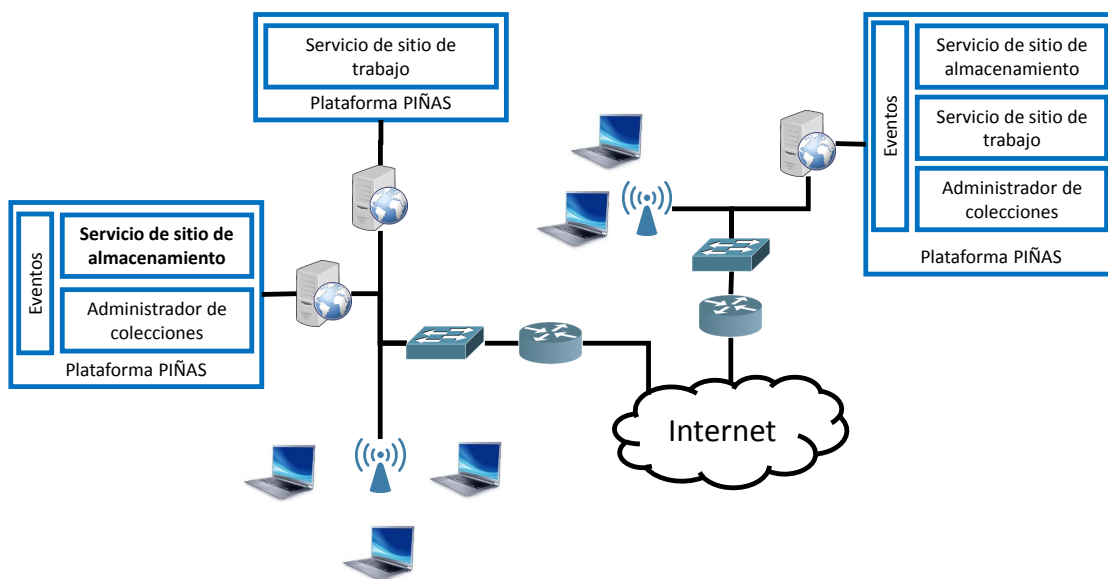


Figura 4.2: La Plataforma PIÑAS actúa como un middleware que proporciona una interfaz homogénea.

Este principio también define asociaciones entre los sitios de trabajo y de almacenamiento que están disponibles para los colaboradores. Estas asociaciones constituyen el primer paso para soportar el trabajo nómada, ya que un colaborador puede establecer una conexión con otros sitios mientras se mueve. Además estas asociaciones permiten llevar a cabo optimizaciones (e.g., acceder a las colecciones desde el sitio más cercano) así como validaciones (e.g.,

identificación de sitios que necesitan ciertas colecciones).

Para ilustrar este principio, supongamos que Margot, Lina y Bart tiene varios sitios (s_0 a s_4) a su disposición (ver Figura 4.3). Bart puede modificar colecciones desde cualquier sitio de trabajo s_0, s_1 o s_3 los cuales pertenecen a dominios de diferentes: s_0 pertenece al *dominio*₀ s_1, s_2 al *dominio*₁ y s_3 y s_4 al *dominio*₂ (ver Figura 4.3) instituciones. Como estos sitios están interconectados por medio de Internet, la base de colecciones¹ de Bart se replica en el sitio de almacenamiento de cada dominio, con el fin de prevenir fallos de red que interrumpan al grupo de trabajo.

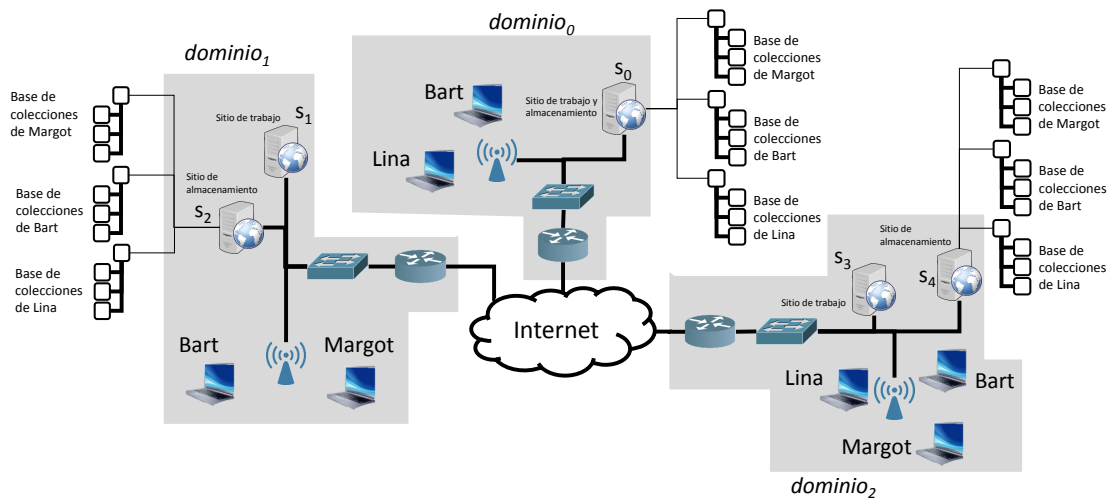


Figura 4.3: En cada dominio diferente en el que los colaboradores participan usando la Plataforma PIÑAS hay al menos un sitio de almacenamiento que contiene una réplica de la base de colecciones de todos los coautores, En el caso del *dominio*₀ un solo servidor actúa como sitio de almacenamiento y de trabajo (s_0), mientras que en los otros dominios (áreas sombreadas) los tienen en sitios separados; *dominio*₁ tiene a s_1 como sitio de trabajo y s_2 como sitio de almacenamiento mientras que el *dominio*₂ tiene a s_3 como sitio de trabajo y s_4 como sitio de almacenamiento.

Cualquier usuario tiene acceso transparente a las colecciones (sin importar su ubicación). Las colecciones son administradas por los sitios de almacenamiento más cercanos, los cuales pertenecen al mismo dominio. Esta organización de sitios les permite a los usuarios moverse de un punto a otro en Internet, de donde puede recuperar su base de colecciones de forma automática. Así un colaborador puede trabajar en un entorno distribuido más confiable.

Los colaboradores cuyos sitios de trabajo están conectados a redes menos susceptibles a desconexiones pueden subir o descargar colecciones compartidas hacia y desde un sitio de

¹Una base de colecciones es un contenedor que agrupa recursos que están relacionados con un documento web

almacenamiento común, por lo que no hay necesidad de crear réplicas de sus bases de colecciones. Por otro lado, un sitio puede actuar al mismo tiempo como un sitio de trabajo y de almacenamiento, así que puede trabajar de forma autónoma porque su sesión es insensible a fallas de la red (ver Figura 4.3).

La plataforma PIÑAS mezcla las arquitecturas totalmente replicada e híbrida para proporcionar una solución escalable para soportar el trabajo colaborativo sobre la Web.

4.1.2. Extensiones de PIÑAS

La frontera entre navegadores estándar y la producción colaborativa es removida por PIÑAS, debido a que son integradas en un sólo ambiente, por lo tanto un documento puede ser consultado por cualquier usuario a través de un navegador Web mientras es editado por un grupo de colaboradores usando una aplicación colaborativa de PIÑAS.

En cada sitio de trabajo, la plataforma PIÑAS proporciona un esquema de nombres compatible con URL que permite a los colaboradores designar y acceder homogéneamente a dos tipos de documentos Web: a) documentos estándar y b) documentos fragmentados, los cuales están específicamente diseñados para soportar la producción colaborativa asíncrona. Es posible establecer hipervínculos entre los documentos estándar y los fragmentados, los cuales son manejados de la misma forma que los hipervínculos entre documentos estándar.

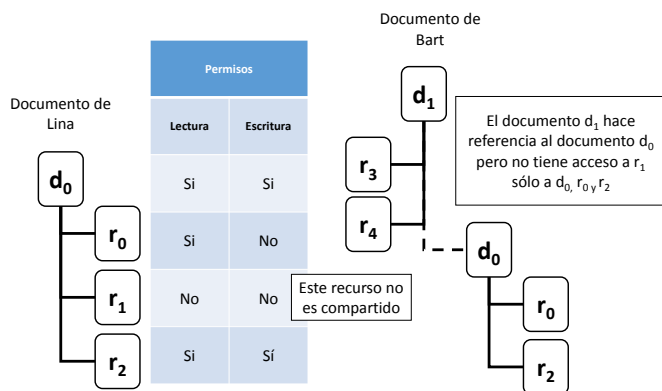


Figura 4.4: Ejemplo en el que un usuario niega el acceso al recurso r_1 , por lo que cualquier otro usuario no podrá tener acceso al mismo, pero si al resto.

En cada sitio de almacenamiento, PIÑAS extiende las funciones de designación para permitirle a los servidores Web identificar el tipo de documento solicitado y el tipo de cliente (navegadores o aplicaciones colaborativas de PIÑAS). La identificación de los tipos de documentos y clientes facilita la creación de respuestas apropiadas para solicitudes de acceso de documentos. Así la presentación del documento es dinámicamente adaptada por cada tipo

de cliente, esto quiere decir que si un navegador pide un documento fragmentado, el servidor Web generará un documento monolítico.

Para proporcionar control de acceso de documentos, PIÑAS depende de los atributos y roles de autores sobre los fragmentos y recursos de un documento compartido. En respuesta a una solicitud de acceso a documento, el servidor Web crea una versión reducida del documento de acuerdo al rol del solicitante. Si un fragmento se declara como confidencial por el grupo de coautores, una solicitud para este fragmento será denegada para toda persona externa. Como resultado, el servidor crea un documento que no incluye partes restringidas (ver Figura 4.4).

4.1.3. Sitios de almacenamiento

Los *sitios de almacenamiento* son las entidades encargadas de dar soporte a la distribución y compartición de recursos web. Estas entidades tienen varias características principales:

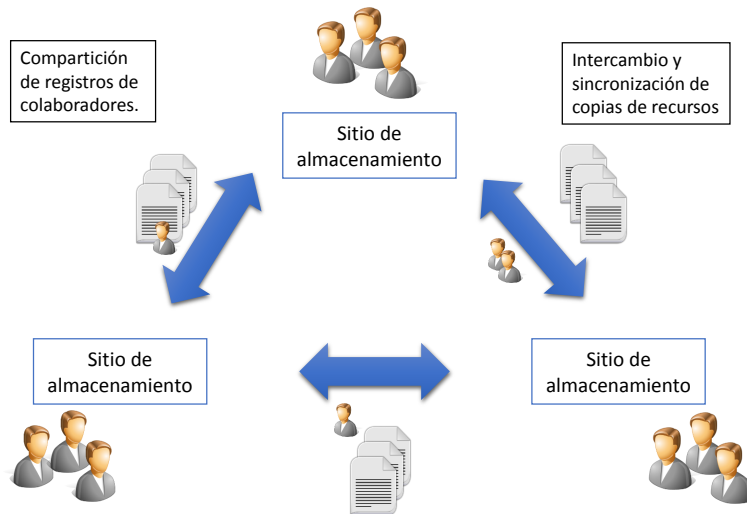


Figura 4.5: Esquema que muestra a sitios de almacenamiento intercambiando registros de colaboradores y recurso.

- Tienen capacidad para aceptar peticiones por recursos (tales como archivos HTML, XML, XLST, PNG, etc) y para responder a tales peticiones enviando los recursos solicitados.
- Los sitios de almacenamiento mantienen registros de colaboradores, además de contar con la capacidad para compartirlos entre otros sitios de almacenamiento, en función de las necesidades de cooperación.
- Tienen la capacidad para distribuir copias de sus recurso hacia otros sitios de almacenamiento, en función de las necesidades de cooperación.

- Tienen la capacidad para mantener sincronizadas las copias distribuidas entre los diversos sitios de almacenamiento.
- Dan soporte al trabajo cooperativo asíncrono.
- Tienen la capacidad para asignar identificadores que son únicos (identificadores globales) entre todos los sitios de almacenamiento.

Cada sitio de almacenamiento S tiene un conjunto U de colaboradores, un conjunto C de colecciones y un conjunto R de recursos, de tal manera que un sitio de almacenamiento está definido por $S = (U, C, R)$. Una colección $c \in C$ es también un recurso ($c \in R$). Una colección c está definida por un conjunto u de colaboradores ($u \subseteq U$) y por un conjunto de recursos γ ($\gamma \subseteq R$), es decir que un colección está definida por $c = (u, \gamma)$.

Colecciones

Una *colección* es un contenedor que agrupa a todos los recursos que conforman un documento web (ver Figura 4.6). Este contenedor además tiene un archivo XML en el que se definen las áreas de trabajo (denominadas como fragmentos), así como los colaboradores asociados a cada área. La suma de las áreas de trabajo conforma el documento web completo.

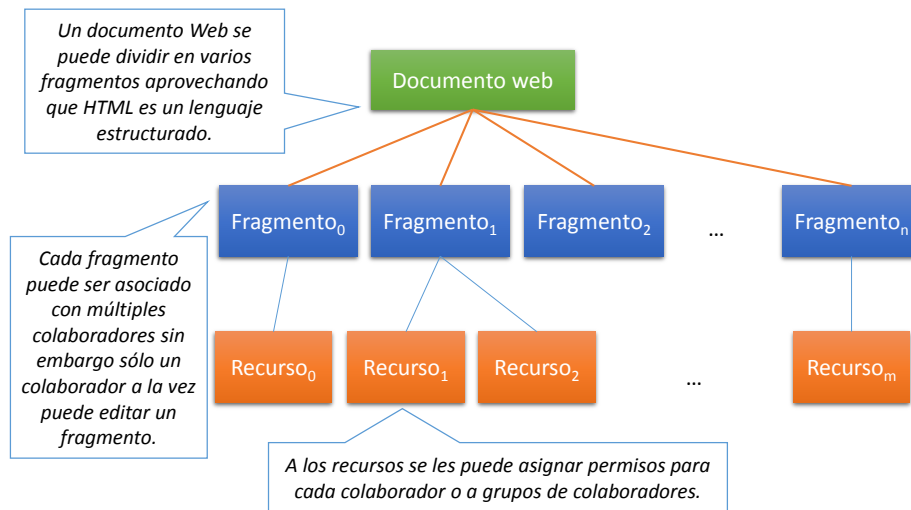


Figura 4.6: División de fragmentos propuesta por PIÑAS.

4.2. Alliance

Alliance es una aplicación de edición colaborativa que permite a las personas, que están geográficamente distribuidas para trabajar y producir documentos estructurados compartidos [Salcedo and Decouchant, 1997]. Esta aplicación es un sistema colaborativo construido encima de Thot [INRIA and S.A., 1997] el cual usa parte de la tecnología web como apoyo

para lograr tanto la administración de documentos distribuidos, como la conciencia de grupo asíncrona, la comunicación y la cooperación entre autores distribuidos.

Alliance organiza el trabajo de tal manera que cada miembro tiene un rol diferente en cada parte del documento (hay que mencionar que Alliance divide el documento en múltiples partes, las cuales son llamadas fragmentos. Ver Figura 4.6). Esta aplicación define cuatro roles que son lector, escritor, administrador y nulo, en donde el rol de administrador es asignado al creador del documento, quien es responsable de distribuir los fragmentos y los roles a los colaboradores. En esta aplicación, el mismo autor puede tener diferentes roles en diferentes fragmentos, sin embargo, sólo un colaborador puede editar un fragmento en un momento dado.

Alliance proporciona a sus colaboradores conciencia de grupo, informándole a cada colaborador del trabajo realizado por todos los colaboradores, además proporciona un área privada de trabajo, donde cada colaborador puede realizar sus contribuciones para después hacerlas públicas.

Esta aplicación está basada en conciencia de grupo asíncrona, lo que implica que los cambios realizados por algún colaborador no serán mostrados a los otros colaboradores hasta que el autor de los cambios, lo decida. Otro aspecto importante de Alliance, como parte de su conciencia asíncrona, es que los colaboradores que tienen el rol de lector en algún fragmento pueden solicitar la última versión de quien tenga el rol de escritor, quien es el que tiene la última versión del fragmento [Salcedo and Decouchant, 1997]. Es importante hacer notar que el colaborador con el rol de escritor puede decidir bajarse a un de rol con menos derechos, i.e., a un simple lector. Esta operación hace que a los demás colaboradores se les avise que ellos pueden tomar el rol de escritor.

Alliance ha desarrollado sofisticados mecanismos de administración de documentos usando el protocolo HTTP, el esquema de nombre de las URLs y del uso de *scripts* para CGI, todo con el fin de permitir la cooperación de documentos distribuidos.

En Alliance los documentos compartidos se encuentra representado por medio de un conjunto de archivos que contienen: fragmentos de documento, roles de usuario para cada fragmento, el orden de los fragmentos, así como el estado de cada fragmento. Con el fin de que los documentos puedan ser compartidos Alliance crea copias de los documentos junto con todos sus metadatos que permiten su administración, con el fin de que cada usuario pueda trabajar de manera local. Se debe señalar que las copias de los documentos y de sus metadatos tienen que ser actualizadas cada vez que un usuario remoto las modifica. Aquí es en donde Alliance toma ventaja de la cooperación asíncrona la cual permite que las actualizaciones no tengan que ser distribuidas en tiempo real.

En Alliance la consistencia se mantiene por medio de un principio simple, el cual consiste en siempre mantener una copia maestra de cada fragmento, la cual tiene la característica de

servir como referencia para otras copias del fragmento llamadas copias esclavas, cada una de estas copias (maestra y esclava) permiten diferentes roles. En este caso la copia maestra permite a los usuarios tomar cualquier rol (escritor, administrador, lector o nulo). El punto clave es que sólo existe una copia maestra. Se debe mencionar que sólo un usuario puede tomar uno de los roles para cada fragmento. Por otra parte la copia esclava sólo permite los roles de lector o nulo. El conjunto de todas las copias maestras conforma el estado actual del documento compartido (Ver Figura 4.7).

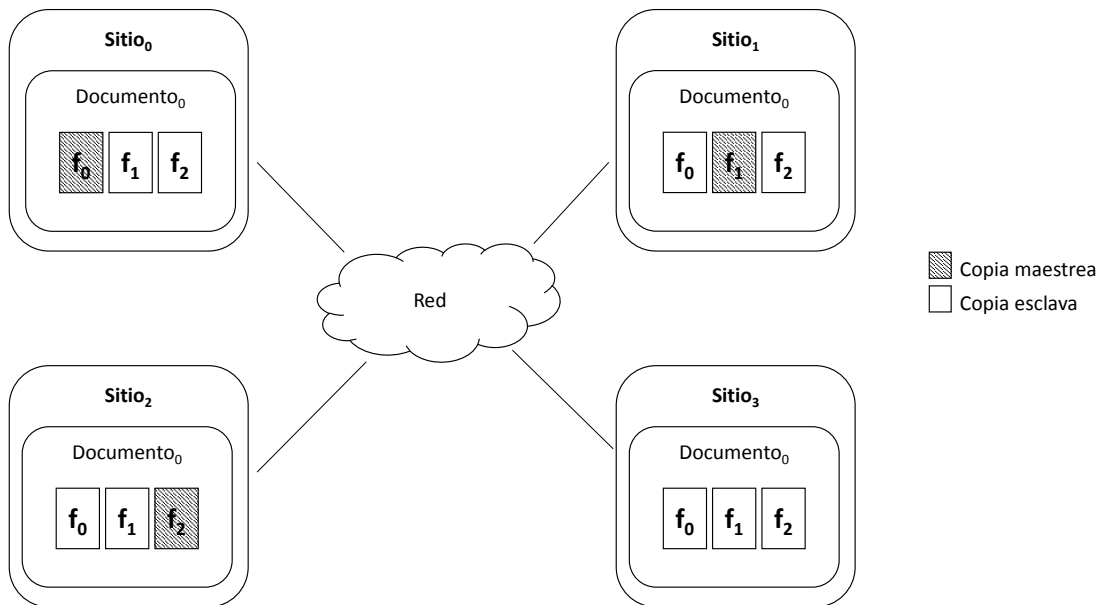


Figura 4.7: Esquema que muestra a cuatro sitios en los que se encuentra un documento (*Documento₀*) fragmentado en tres partes (f_0 , f_1 y f_2). La copia maestra del fragmento f_0 se encuentra en *Sitio₀* los fragmentos f_0 de los sitios *Sitio₁*, *Sitio₂* y *Sitio₃* son copias esclavas. La suma de los fragmentos f_0 de *Sitio₀* más el fragmento f_1 de *Sitio₁* más el fragmento f_2 de *Sitio₂* conforman el estado actual del documento compartido.

Los usuarios de Alliance pueden solicitar el rol de escritor sobre un fragmento en este caso el sistema inicia un proceso denominado migración del fragmento maestro, esta operación inicia un proceso de transacción con el fin de evitar pérdidas o duplicados de copias maestras. Antes de mover un fragmento de un sitio a otro se evalúan ciertas condiciones, entre las más importantes se encuentran las siguientes: la copia maestra es alcanzable y no ha sido bloqueada por otro colaborador a causa de que este está usando el rol de escritor; otra condición importante es que el usuario con el rol de administrador o escritor acepte ceder sus derechos al colaborador solicitante, si todas las condiciones se cumplen el proceso se realiza si una de ellas no sucede entonces no se realiza nada.

Con respecto a las actualizaciones realizadas en los fragmentos no se envían a los sitios

involucrados de manera automática, sino que aquí se opta por enviar un mensaje corto a manera de notificación avisando que existen cambios en los fragmentos, con el fin de que la actualización sea solicitada cuando se requiera.

Con respecto a la arquitectura de Alliance se puede decir lo siguiente, la aplicación cuenta con un editor local (Thot [INRIA and S.A., 1997]) es decir una aplicación de escritorio. Esta aplicación de escritorio se conecta por medio de una capa de software encargada de la comunicación denominada asistente, la cual tiene como base un cliente que maneja el protocolo HTTP. Otra capa de software importante, es la llamada capa de servidor de documento la cual es un servidor web que se apoya de la tecnología CGI. Esta última capa es la más importante ya que tiene varios objetivos entre los que se destacan la gestión de distribución de documentos, migración de copias, control de concurrencia, candados, etc. (ver Figura 4.8).

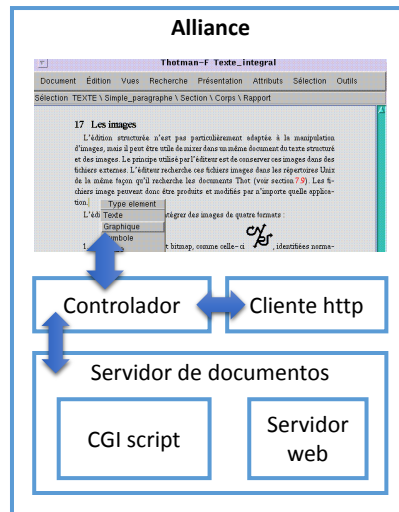


Figura 4.8: Esquema que muestra las capas de software involucradas en la aplicación colaborativa Alliance.

La idea de este editor colaborativo es tomar un documento HTML y dividirlo en fragmentos, donde cada fragmento puede ser editado por un grupo de colaboradores. Sin embargo, la edición no es síncrona, lo que significa que no todos los miembros de un grupo de colaboradores pueden editar un mismo fragmento al mismo tiempo, aquí en Alliance la edición es asíncrona, es decir sólo uno a la vez puede editar el fragmento, si alguien más quiere editar el fragmento entonces quien tenga el rol de editor debe ser su rol de escritor. Un aspecto que se debe hacer notar es que la versión más actual del documento completo es la suma de los fragmentos (estas copias son denominadas copias maestras) que tienen los colaboradores con el rol de escritores.

La aplicación colaborativa Alliance trabaja de la siguiente manera, un colaborador edita un fragmento y decide hacer públicos los cambios entre los miembros de trabajo. En este

momento los datos que se hayan agregado al fragmento son almacenados en un archivo con ayuda del editor Thot. Lo siguiente que hace Alliance es avisar a todos los colaboradores, tanto a los que están colaborando en el mismo fragmento como a los que están trabajando en otros fragmentos que hay una nueva versión de tal fragmento. Esta notificación se hace por medio de un cliente HTTP. Una vez que los colaboradores fueron notificados de la existencia de esta nueva versión de fragmento, estos pueden solicitar que se le envíe la información para lo que la aplicación crea un proceso nuevo quien se encarga de comunicarse con el servidor HTTP en donde se encuentra el fragmento nuevo y solicita que se le envíe el recurso. Cuando el proceso encargado recibe la información la almacena en el archivo y le notifica al editor que hay cambios en el archivo (ver Figura 4.9).

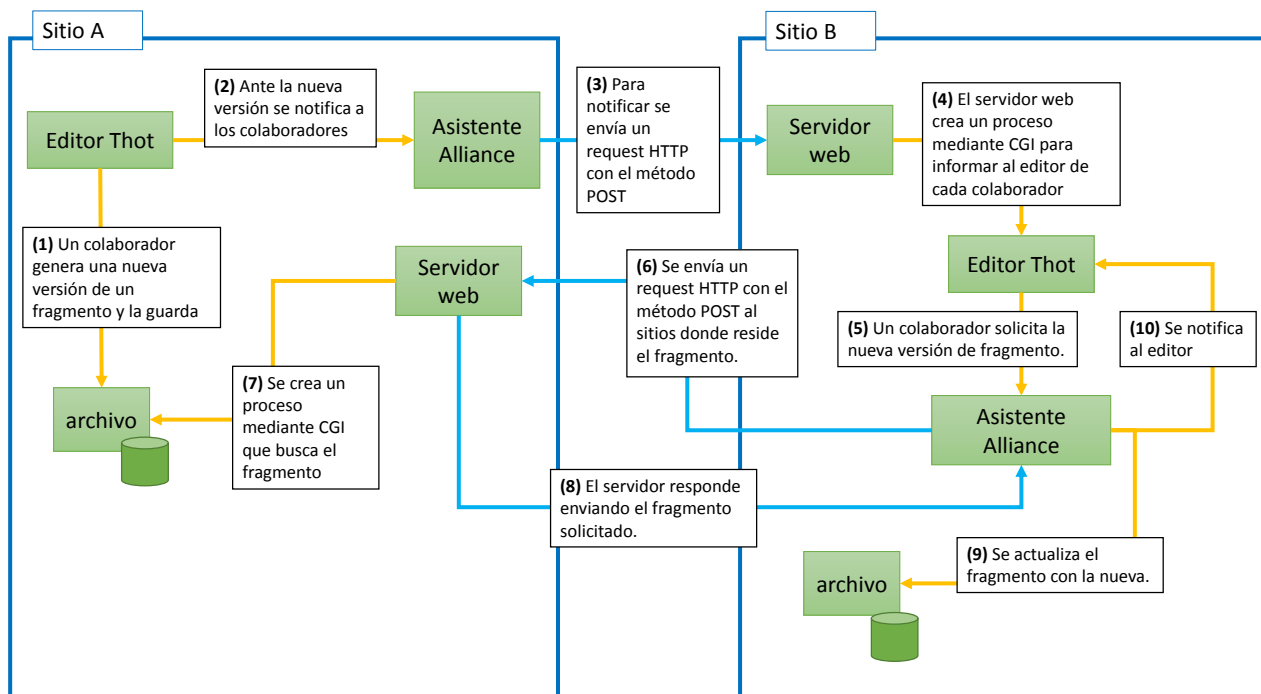


Figura 4.9: Funcionamiento general de Alliance.

Capítulo 5

Análisis y solución al problema de los hipervínculos rotos

El capítulo inicia con el análisis del problema de los hipervínculos rotos (ver Sección 5.1), y se finaliza con la propuesta de solución al problema de los hipervínculos rotos basado en la plataforma PIÑAS y el uso la integridad de referencia proporcionado por los manejadores de bases de datos (ver Sección 5.2).

5.1. Análisis detallado del problema de las hipervínculos rotos

En esta sección se desglosa a detalle cada uno de los factores involucrados en el problema de los hipervínculos rotos. Primero se plantea un escenario común en el que el problema se presenta (ver Subsección 5.1.1). Para después analizar los detalles técnicos del problema (ver Subsección 5.1.2).

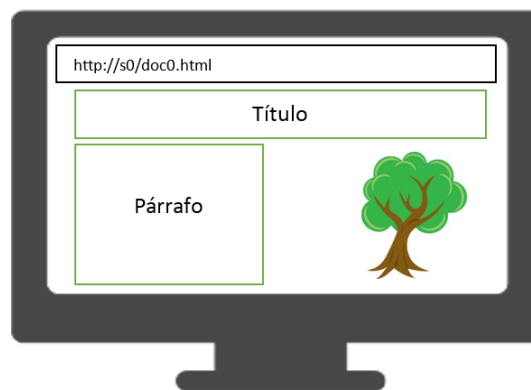


Figura 5.1: Página web producida por el archivo *doc0.html*.

5.1.1. Descripción de un escenario común

```

<html>
  <head>
    <link rel="stylesheet" href="css/style.css"/>
  </head>
  <body>
    <h1>Título</h1>
    <p>Párrafo</p>
    
  </body>
</html>

```

Figura 5.2: Contenido del archivo *doc0.html*.

Una situación común en la Web es la siguiente: un navegador carga una página web llamada *doc0.html* del servidor web llamado s_0 (ver Figura 5.1).

A pesar de que el navegador muestra una página web, la página no está conformada por un único elemento que contiene texto y una imagen. Si se pone atención al servidor web, en específico al archivo *doc0.html*, se puede ver que la página web está definida por un documento de texto que contiene etiquetas HTML (ver Figura 5.2).

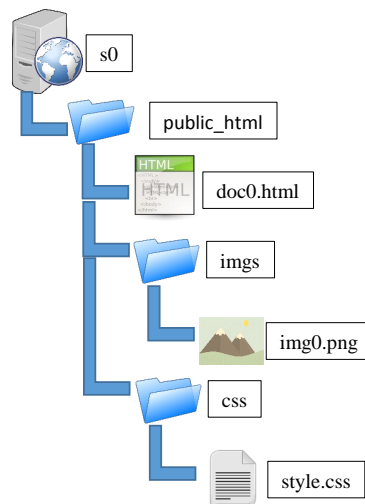


Figura 5.3: Estructura de directorios en el servidor s_0 .

Este documento a su vez tiene ligas hacia diversos recursos. En el ejemplo de la figura 5.2 se aprecia el contenido del documento *doc0.html* y se observa que tiene dos ligas: una hacia un archivo que define el formato (*style.css*), y otra liga hacia un archivo de imagen PNG (*img0.png*). Las ligas se muestran en rojo en la Figura 5.2. Tales recursos son locales, i.e.,

5.1. ANÁLISIS DETALLADO DEL PROBLEMA DE LAS HIPERVÍNCULOS ROTOS 27

se encuentran dentro de la estructura de directorios del servidor web s_0 . En la Figura 5.3 se muestra la estructura de directorios para la página web.

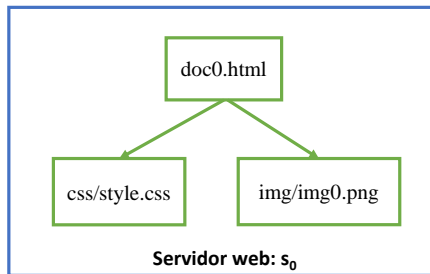


Figura 5.4: Representación de referencias a archivos locales dentro del archivo *doc0.html*.

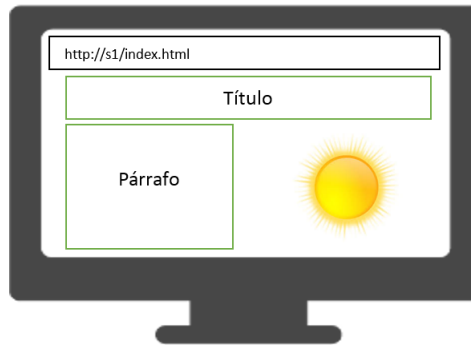


Figura 5.5: Página web creada a partir del archivo *index.html*.

De manera esquemática se podría decir que el documento HTML llamado *doc0.html* tiene referencias a dos archivos locales uno llamado *img0.png* y otro llamado *style.css* y se representaría como se muestra en la Figura 5.4.

Ahora suponga que en otro servidor web llamado s_1 se tiene un documento HTML llamada *index.html*. Este documento es interpretado por el navegador y muestra la página web de la Figura 5.5. El contenido de *index.html* se muestra en la Figura 5.6.

```
<html>
<head>
  <link rel="stylesheet" href="css/style1.css"/>
</head>
<body>
  <h1>Título</h1>
  <p>Párrafo</p>
  
</body>
</html>
```

Figura 5.6: Contenido del archivo *index.html*.

28CAPÍTULO 5. ANÁLISIS Y SOLUCIÓN AL PROBLEMA DE LOS HIPERVÍNCULOS ROTOS

El archivo *index.html* a su vez está definido dentro de la estructura de directorios de la Figura 5.7.

El esquema que muestra las ligas hacia los archivos locales se muestra en la Figura 5.8.

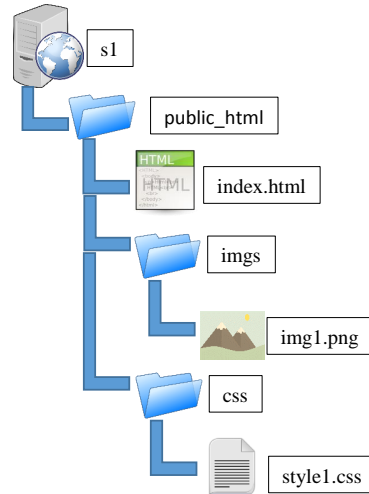


Figura 5.7: Estructura de directorios en el servidor s_1 .

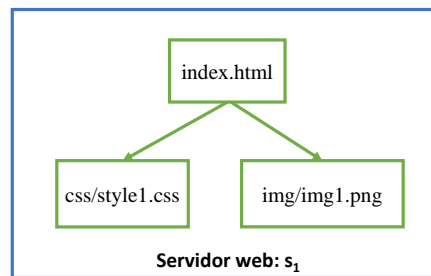


Figura 5.8: Referencias a archivos locales del archivo *index.html*.

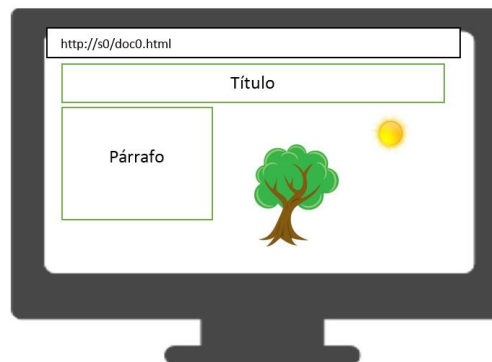


Figura 5.9: Página web creada a partir de archivo *doc0.html*.

5.1. ANÁLISIS DETALLADO DEL PROBLEMA DE LAS HIPERVÍNCULOS ROTOS 29

La página web mostrada en la Figura 5.1 (descrita por el archivo HTML *doc0.html*) también hace referencia hacia dos archivos que se encuentran dentro del mismo servidor web s_1 , tal como se muestra en la Figura 5.8. Ahora en la página web definida por el documento *doc0.html* se quiere usar un recurso que está en otro servidor el cual es la imagen *img1.png* que está en el servidor web s_1 , con el fin de que el resultado se vea como el de la Figura 5.9. En este caso lo que se debe de hacer es modificar el documento *doc0.html*, tal como se muestra en la Figura 5.10 (texto en verde).

```
<html>
  <head>
    <link rel="stylesheet" href="css/style.css"/>
  </head>
  <body>
    <h1>Título</h1>
    <p>Párrafo</p>
    
    
  </body>
</html>
```

Figura 5.10: Contenido del archivo *doc0.html*.

Con la inserción de la línea de texto en verde se logra que se vea en la página web definida por *doc0.html* la imagen que se encuentra en el servidor s_1 . Para modificar el tamaño y/o la posición de la imagen *img1.png* se debe modificar el archivo *style.css*. Esquemáticamente se vería como se muestra en la Figura 5.11.

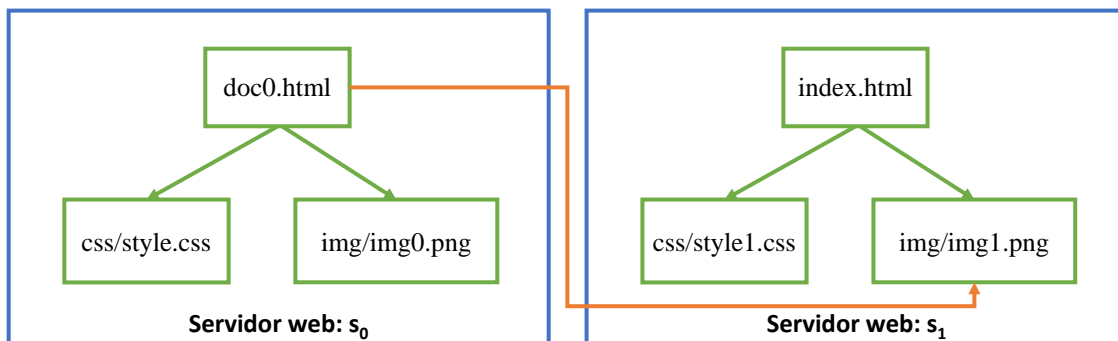


Figura 5.11: Las referencias locales y remotas de los archivos *doc0.html* e *index.html*.

En este momento el documento *doc0.html* tendría una referencia o liga hacia un recurso remoto (en este caso la imagen *img1.png* que se encuentra en s_1).

Los problemas surgen cuando por alguna razón en el servidor s_1 se decide que ya no se usará a la imagen *img1.png* dentro del documento *index.html*. Bajo estas circunstancias, una

posibilidad es que se elimine la imagen *img1.png* y se coloque otra en su lugar para que se muestre un resultado con en la Figura 5.12.

Para lograr tal fin, el documento *index.html* se modifica tal como se muestra en la Figura 5.13. Se elimina *img1.png* de la estructura de directorios y se agrega a la nueva imagen quedando la estructura de directorios del servidor s_1 que se muestra en la Figura 5.14.

Como se ve en la Figura 5.14 la imagen *img0.png* ha desaparecido y en su lugar pusieron a uno nuevo llamado *img2.png* (recuadro punteado de la imagen 5.14).

El servidor s_1 no tiene manera de avisarle al sitio s_0 de que se ha cambiado la imagen *img1.png* por otra llamada *img2.png* y que además el contenido ya no es el mismo.

Ante esta situación, cuando el navegador intenta mostrar la página definida por *doc0.html* el resultado es el que se muestra en la Figura 5.15.

Aún cuando el documento *doc0.html* no fue modificado, el resultado de la página web ya no es el esperado. Esquemáticamente se tiene lo que se muestra en la Figura 5.16.

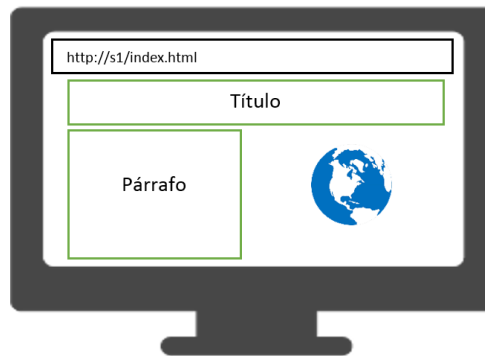


Figura 5.12: Página web creada a partir del archivo *index.html*, con una *img1.png*.

```
<html>
  <head>
    <link rel="stylesheet" href="css/style1.css"/>
  </head>
  <body>
    <h1>Título</h1>
    <p>Párrafo</p>
    
  </body>
</html>
```

Figura 5.13: Contenido del archivo *index.html*, después de haber cambiado la imagen.

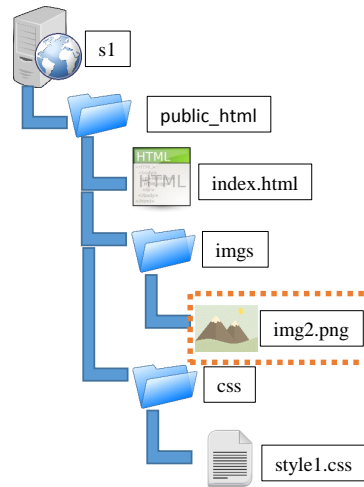


Figura 5.14: Estructura de directorios del servidor s_1 , en el que se muestra al nuevo recurso.

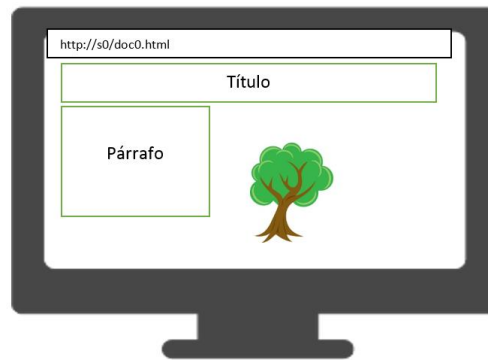


Figura 5.15: Página web creada a partir del documento *doc0.html* después de haber agregado *img2.png*



Figura 5.16: Referencias locales y remotas de los archivos *doc0.html* e *index.html*, en el que además se aprecia que *doc0.html* tiene una referencia hacia un recurso que no existe (link roto).

Tal como se ve en la Figura 5.16 el documento *doc0.html* tiene una referencia hacia un archivo que ya no existe en el servidor s_1 , lo que provoca un error en la visualización del documento *doc0.html*. Otra situación común que puede ocurrir es que en el sitio s_1 podría eliminarse la imagen *img1.png* que contiene un “sol” y podría insertarse una imagen que contiene al planeta “tierra”, pero por alguna razón se deja el mismo nombre que tenía la imagen anterior es i.e., *img1.png*. Bajo esta situación el esquema que muestra las ligas entre recursos se vería tal como se muestra en la Figura 5.17.

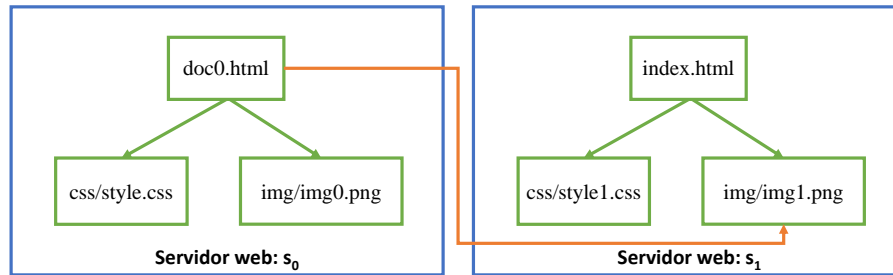


Figura 5.17: Las referencias locales y remotas de los archivos *doc0.html* e *index.html*.

La página no muestra un mensaje de error, sin embargo cuando se visualiza la página definida por *doc0.html* se obtiene lo que se muestra en la Figura 5.18.

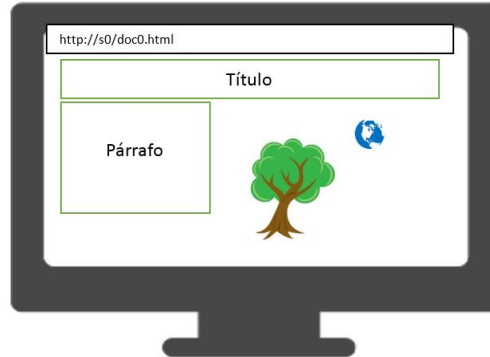


Figura 5.18: Página creada a partir del archivo *doc0.html*, en el que aparece una inconsistencia visual debido a que el contenido ha cambiado.

Tal como se ve en la Figura 5.18 no se obtiene el resultado esperado para visualizar el documento *doc0.html*, lo que implica otro error de visualización.

5.1.2. Detalles técnicos del problema

En el *doc0.html* tenemos lo que se muestra en la Figura 5.19. En el que se aprecian referencias hacia recursos en el servidor local y a uno remoto, ahora si nos concentramos en la referencia hacia un recurso remoto se puede ver lo que se muestra en la Figura 5.20.

```

<html>
  <head>
    <link rel="stylesheet" href="css/style.css"/>
  </head>
  <body>
    <h1>Título</h1>
    <p>Párrafo</p>
    
    
  </body>
</html>

```

Figura 5.19: Contenido del archivo *doc0.html*, en el que se hace referencia a un recurso local y a uno remoto.

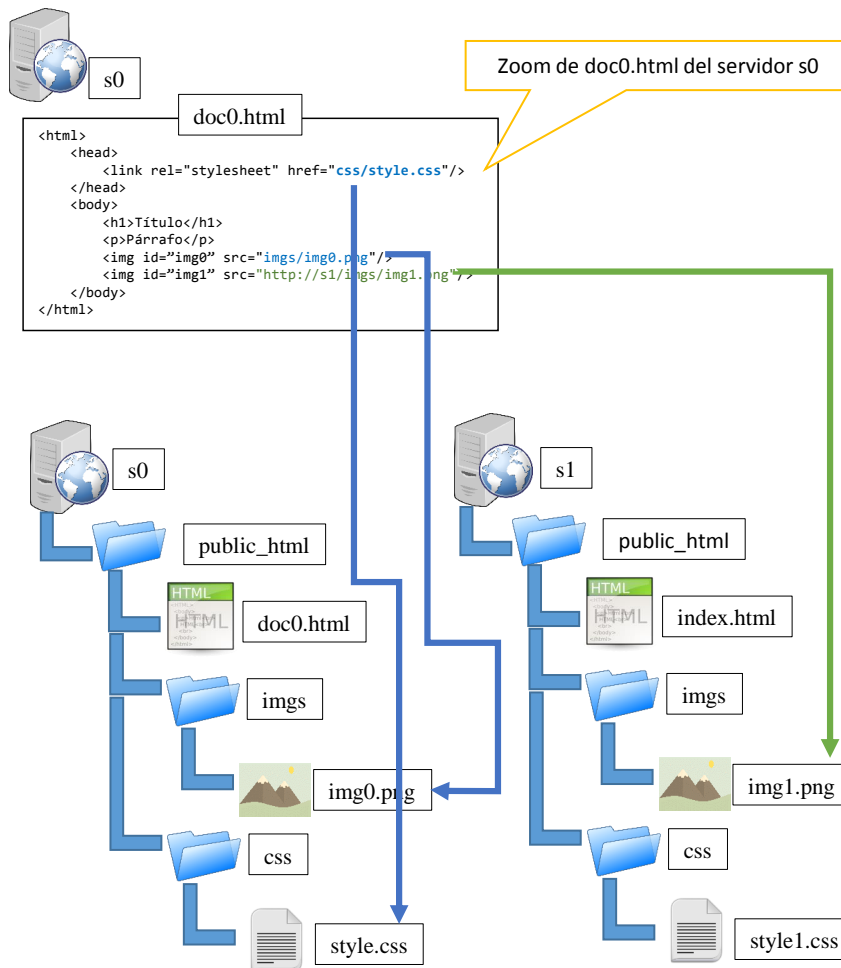


Figura 5.20: Esquema en el que se muestran las referencias que mantiene el archivo *doc0.html*. Una hacia un recurso local y otra hacia un recurso remoto.

En el diagrama de la Figura 5.20 se muestra en flechas azules las referencias hacia recurso locales mientras que en verde una referencia hacia un recurso remoto.

Aún cuando aparentemente se ve en el diagrama que, de algún modo, el servidor s_0 (por medio del *doc0.html*) tiene referencias hacia los recursos locales y remotos, La verdad es que no es trabajo del servidor o servidores resolver estas referencias. Lo que resuelve esas referencias es el software encargado de mostrar la página web que (normalmente) es un navegador.

Lo que verdaderamente ocurre es lo siguiente. En el servidor s_0 tenemos la estructura de directorios que se muestra en la Figura 5.21.

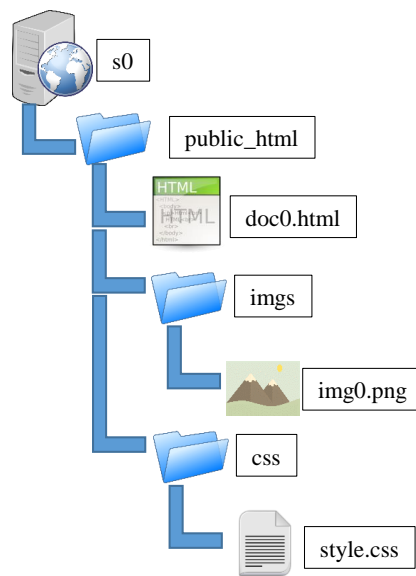


Figura 5.21: Estructura de directorios del servidor web s_0 , para el documento web generado por *doc0.html*.

Un cliente (un navegador por ejemplo) manda una solicitud (*request*) por el documento *doc0.html*, esto se hace al escribir en el navegador la URL: *http://s0/doc0.html*. El navegador manda una solicitud (*request*) del recurso *doc0.html* al servidor s_0 . Esta solicitud se envía usando el protocolo HTTP. Dicha petición luciría como se muestra en la Figura 5.22.

```

GET /doc0.html HTTP/1.1
User-Agent: curl/7.23.1 (x86_64-apple-darwin11.2.0) libcurl/7.23.1 zlib/1.2.5
Host: s0
Accept: text/html
...
  
```

Figura 5.22: Contenido de la solicitud para obtener *doc0.html*.

5.1. ANÁLISIS DETALLADO DEL PROBLEMA DE LAS HIPERVÍNCULOS ROTOS 35

Como respuesta a tal petición, el servidor s_0 encuentra el recurso solicitado (*doc0.html*) y envía la respuesta mostrada en la Figura 5.23.

El navegador lee el contenido para reconocer las etiquetas HTML (en la Figura 5.23 texto marcado con (1)). En algún punto de la interpretación del HTML se llega a la siguiente etiqueta:

```
<link rel='stylesheet' href='css/style.css' />
```

Cuando se alcanza el atributo `href` de la etiqueta `link` el navegador crea una nueva solicitud al servidor s_0 (solicitud que se muestra en la Figura 5.24).

El navegador recibe como respuesta un mensaje similar a lo mostrado en la Figura 5.25.

```
HTTP/1.1 200 OK
Date: Sat, 17 Nov 2012 17:42:15 GMT
Server: Apache/2.2.3 (Red Hat)
X-Powered-By: PHP/5.2.17
Vary: Accept-Encoding,User-Agent
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

<!doctype html>
<html lang="es">
<head>
  <link rel="stylesheet" href="css/style.css"/>
</head>
<body>
  <h1>Título</h1>
  <p>Párrafo</p>
  
  
</body>
</html>
```

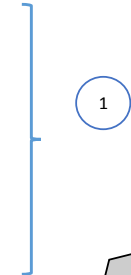


Figura 5.23: Contenido de la respuesta (*response*) como resultado de la solicitud (*request*) esquematizada en la Figura 5.22.

```
GET /css/style.css HTTP/1.1
User-Agent: curl/7.23.1 (x86_64-apple-darwin11.2.0) libcurl/7.23.1 zlib/1.2.5
Host: s0
Accept: text/css
...
```

Figura 5.24: Contenido de la solicitud del recurso *style.css*.

```

HTTP/1.1 200 OK
Date: Sat, 17 Nov 2012 17:50:15 GMT
Server: Apache/2.2.3 (Red Hat)
X-Powered-By: PHP/5.2.17
Vary: Accept-Encoding,User-Agent
Transfer-Encoding: chunked
Content-Type: text/css; charset=UTF-8

body {
  font-family: Arial, Arial, 'Lucida Sans Unicode', Helvetica, sans-serif;
  background-position: top left;
  background-repeat: no-repeat;
  background-image: url(../img/fondo.png);
  background-attachment: fixed;
}

header {
  ...

```

Figura 5.25: Contenido de la respuesta a la solicitud esquematizada en la Figura 5.24.

Y un proceso similar se lleva a cabo cuando se interpreta la etiqueta `` siguiente:

```
<img id='img0' src='imgs/img0.png'>
```

Es decir vuelve a crear una solicitud al servidor s_0 y este le regresa la imagen *img0.png* al navegador para que al final el navegador la muestre.

En el caso de la etiqueta `` siguiente que es en la que se hace referencia a un recurso que no está en el servidor s_0 , tenemos también algo similar:

```
<img id='img1' src='http://s1/imgs/img1.png'>
```

La solicitud que haría el navegador ya no sería hacia el servidor s_0 , sino ahora hacia el servidor s_1 . Dicha petición se vería como se muestra en la Figura 5.26.

```

GET /imgs/img1.png HTTP/1.1
User-Agent: curl/7.23.1 (x86_64-apple-darwin11.2.0) libcurl/7.23.1 zlib/1.2.5
Host: s1
Accept: image/png
...

```

Figura 5.26: Contenido de la solicitud por el archivo *img1.png*.

De manera resumida el proceso se ilustra en el diagrama de secuencias de la Figura 5.27.

En el caso de que *img1.png* ya no se localice, pasa lo siguiente: el navegador hace la petición al servidor s_1 el cual responde con un mensaje de recurso no encontrado (*404 Not found*) (Ver Figura 5.28).

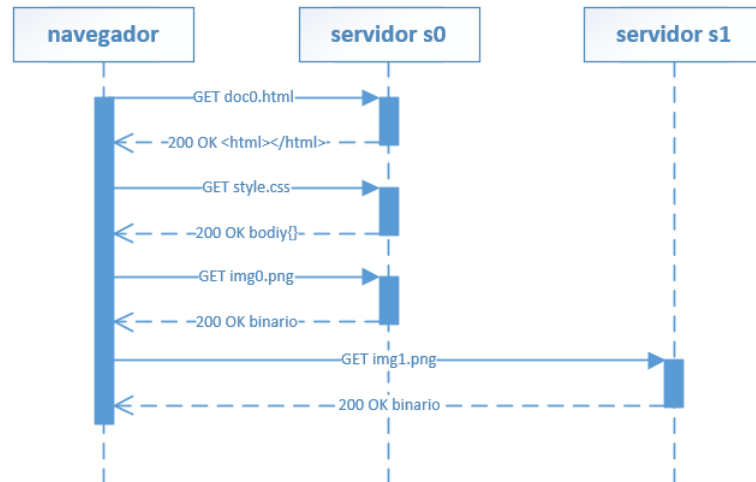


Figura 5.27: Diagrama de secuencias de la interacción entre el navegador y los servidores s_0 y s_1 .

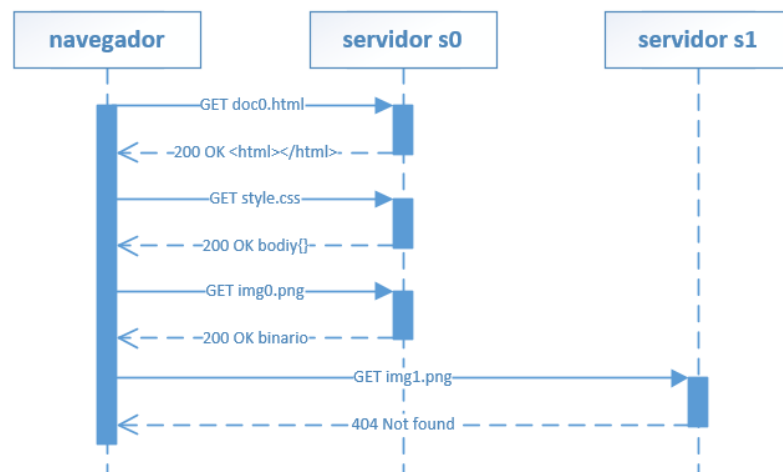


Figura 5.28: Diagrama de secuencias de la interacción entre el navegador con los servidores s_0 y s_1 con una respuesta de recurso no encontrado.

5.2. Solución al problema de los hipervínculos rotos

El problema que tienen las URLs es que son usadas con dos fines: por un lado son usadas para localizar un recurso y por el otro lado son usadas para identificar a un recurso. Esto

quiere decir que si hay un cambio en la ubicación del recurso la URL se invalida, y lo mismo pasa si hay un cambio en el nombre de un recurso.

Para dar solución al problema se usa los tres niveles de nombre de la plataforma PIÑAS esto es un *identificador global de recursos* (denominado *identificador virtual*), donde su principal característica es que es único con respecto al tiempo y al espacio. El identificador global de recursos será el atributo principal del recurso. A cada recurso se le podrán asociar múltiples nombres y ubicaciones (aquí denominados *nombres lógicos*) según las necesidades de los colaboradores. También se propone la existencia de un *identificador físico* en el caso de que se opte por una implementación basada en algún sistema de archivos. Bajo esta situación se propone una organización de recursos (archivos) estática, en la que cada colaborador tendrá un contenedor (directorio) llamado *base* y dentro de este un contenedor más por cada documento (llamado *colección*), el cual agrupará todo tipo de recursos relacionados a un único documento (ver Figura 5.29). Es importante señalar el potencial uso de un sistema de archivos facilita la implementación de la solución, pero también se aumenta la dependencia con algún sistema operativo, reduciendo las ventajas que ofrece la Web. Al mismo tiempo que se abre la puerta para dejar al sistema inconsistente, ya que es posible eliminar archivos y/o directorios sin la necesidad de pasar por el sistema, (es decir usando el mismo sistema de archivos).

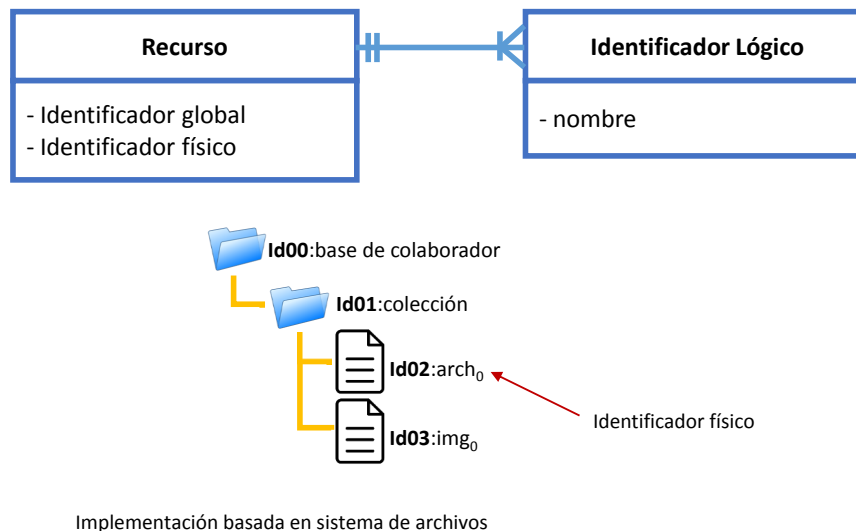


Figura 5.29: Esquema que muestra la relación de uno a muchos entre recurso e identificador lógico, así como el uso del *identificador físico* para conectar con el recurso en el sistema de archivos.

Se propone que el *framework* maneje un formato fijo para las URLs (como parte de la solución al problema de los hipervínculos rotos) el cual será el siguiente:

`http://app/colaborador/colección/recurso`

Cuando el *framework* reciba una URL con el formato anterior este se encargara de regresar el recurso perteneciente al colaborador y colección indicados. Por ejemplo la siguiente URL regresa el recurso llamado `tree` que se localiza en la colección `doc0` y que pertenece a `ortigosa`.

```
http://app/ortigosa/doc0/tree
```

Ante una URL con el siguiente formato el *framework* regresara todos los recursos asociados a la colección de un colaborador.

```
http://app/colaborador/colección/
```

Por ejemplo la siguiente URL regresa todos los recursos de la colección `doc0` del colaborador `ortigosa`.

```
http://app/ortigosa/doc0/
```

Por otra parte si el *framework* recibe una URL con el siguiente formato se regresa todas las colecciones pertenecientes al colaborador indicado.

```
http://app/colaborador/
```

Por ejemplo ante la siguiente URL el *framework* regresa todas las colecciones pertenecientes al colaborador `ortigosa`.

```
http://app/ortigosa/
```

Hay otros formatos de URL como el siguiente que regresa todos los recursos existentes en el sistema este puede ser mejorado para construir una especie de repositorio para que los usuarios seleccionen un recurso previamente cargado por alguien más o incluso dar la opción de cargar un nuevo recurso.

```
http://app/recursos/
```

También existe el formato siguiente el cual regresa una lista con los colaboradores registrados en un sitio de almacenamiento.

```
http://app/colaboradores/
```

La otra parte que ayuda a solucionar el problema de los hipervínculos rotos es usar la integridad de referencias de los manejadores de bases de datos el cual evita que se puedan eliminar recursos que están siendo referenciados por otras tablas.

Para lograr lo anterior se crean tres tablas correspondientes a `Colaborador`, `Colección` y `Recurso`. La relación que existe entre `Colaborador` y `Colección` es una de muchos a muchos

la cual será mapeada con una tabla intermedia donde sus atributos serán las llaves primarias de las entidades *Colaborador* y *Colección*. De igual manera la relación entre las entidades *Colección* y *Recurso* es una de muchos a muchos la cual será mapeada al modelo relacional con una tabla intermedia donde sus atributos serán las llaves primarias de las entidades más un tercer atributo correspondiente al nombre lógico, es atributo será clave para dar soporte a múltiples URLs apuntando a un mismo recurso (ver Figura 5.30).

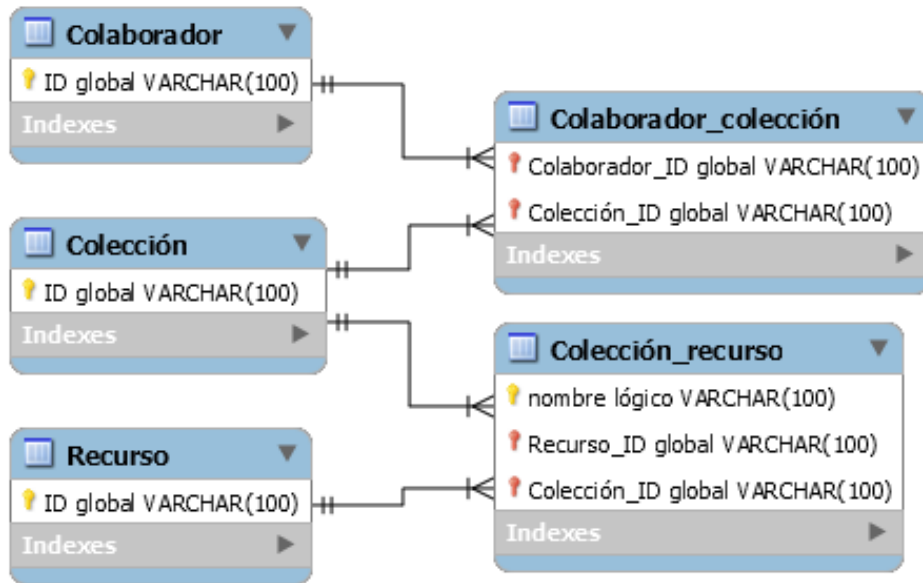


Figura 5.30: Esquema de base de datos para soportar múltiples URLs que apuntan a un mismo recurso y para usar la integridad de referencia del manejador de base de datos.

En la Figura 5.31 se muestran datos de ejemplo para llenar las tablas del esquema de base de datos, los cuales se leen de la siguiente manera: existen dos colaboradores registrados que son (*ortigosa* y *luis*) el colaborador *ortigosa* tiene una colección llamada *documento* mientras que *luis* tiene una colección llamada *reporte*, Ambas colecciones contiene un solo recurso el cual tiene el identificador r_0 , y además se aprecia que para *ortigosa* hace referencia a tal recurso r_0 por medio del nombre *martes* mientras que el colaborador *luis* los hace con el nombre *planeta* (Ver Figura 5.32).

Dado que el recurso r_0 está siendo referenciado tanto por la colección *documento* como por la colección *reporte*, el manejador de base de datos impide que se elimine el recurso (gracias a la integridad de referencias), por lo tanto se evita que suceda un hipervínculo roto a causa de la eliminación de un recurso.

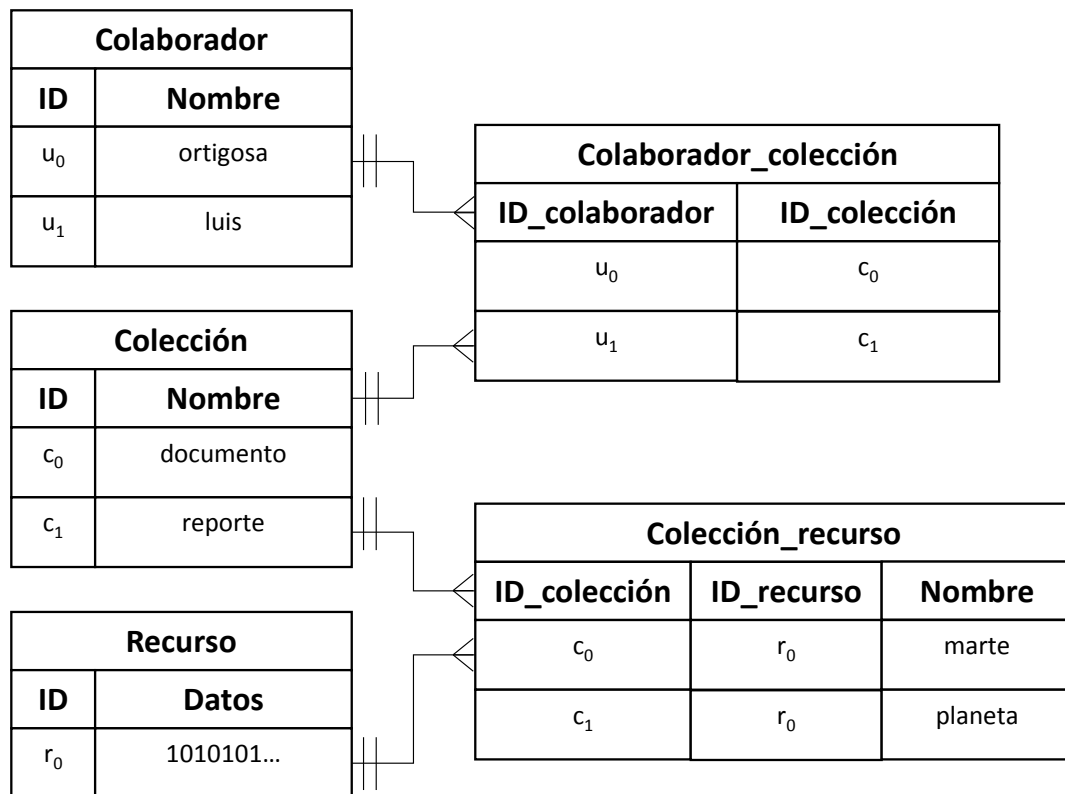


Figura 5.31: Datos de ejemplo para el esquema de bases de datos que almacena las relaciones entre colaborador, colección y recurso.



Figura 5.32: URLs apuntando a un mismo recurso.

Los parámetros de las URLs serán usados para hacer consultas en la base de datos, para el caso de URLs con tres parámetros (formato colaborador/colección/recurso) la consulta que se realiza al manejador de base de datos es la mostrada a continuación:

Join

$$R = \text{Colaborador} \bowtie_{ID_colaborador} \text{Colaborador_colección} \bowtie_{ID_colección} \text{Colección} \bowtie_{ID_colección} \text{Colección_recurso} \bowtie_{ID_recurso} \text{Recurso}$$

Selección

$$S = \sigma_{\text{Colaborador.nombre}=\text{colaborador} \wedge \text{Colección.nombre}=\text{colección} \wedge \text{Recurso.nombre}=\text{recurso}}(R)$$

Proyección

$$\Pi_{\text{Recurso.datos}}(S)$$

La consulta para el caso de URL con dos parámetros (formato colaborador/colección) será la siguiente:

Join

$$R = \text{Colaborador} \bowtie_{ID_colaborador} \text{Colaborador_colección} \bowtie_{ID_colección} \text{Colección}$$

Selección

$$S = \sigma_{\text{Colaborador.nombre}=\text{colaborador} \wedge \text{Colección.nombre}=\text{colección}}(R)$$

Proyección

$$\Pi_{\text{Colección}}(S)$$

Y finalmente para obtener las colecciones de un colaborador se realiza la consulta de la del inciso a), para el caso de querer obtener todos los colaboradores de un sitio de almacenamiento se usa la consulta del inciso b) y finalmente para obtener una lista de todos los recursos de usa la consulta del inciso c).

a)

Selección

$$S = \sigma_{\text{Colaborador.nombre}=\text{colaborador}}(\text{Colaborador})$$

b)

Selección

$$S = \sigma_*(\text{Colaborador})$$

c)

Selección

$$S = \sigma_*(\text{Recurso})$$

5.2.1. Seguimiento de recursos referenciados

La Web puede verse como un grafo dirigido (ver Figura 5.33) en el que los vértices son los documentos web o recursos y las aristas son las URLs que están contenidas (frecuentemente) en documentos web que apuntan a diversos recursos. Sin embargo, no hay un estándar web que lleve el registro de todas las aristas que salen de un vértice (es decir que lleve el registro de todos los hipervínculos que contiene un documento web), ni tampoco uno que lleve el registro de las aristas que convergen en un vértice (es decir un registro de todas las colecciones que hacen referencia hacia un recurso). Lo que provoca problemas relacionados a hipervínculos rotos (broken links) o errores en la visualización del contenido.

Para solucionar este problema (ver Figura 5.33) se propone que para cada vértice (recurso) r_k se lleve el registro de todos los vértices r_i que estén conectados mediante una arista (una URL) hacia r_k (una liga en r_i que apunta a r_k), así como otro registro con todos los vértices r_j que estén relacionados con r_k mediante una liga que está en r_k y que apunta a un recurso r_j . Los vértices r_i representan a todos los recursos (normalmente documentos web) que tienen una URL hacia el recurso r_k . Los vértices r_j representan a todos los recursos a los que apunta una URL contenida en el recurso r_k (ver Figura 5.33).

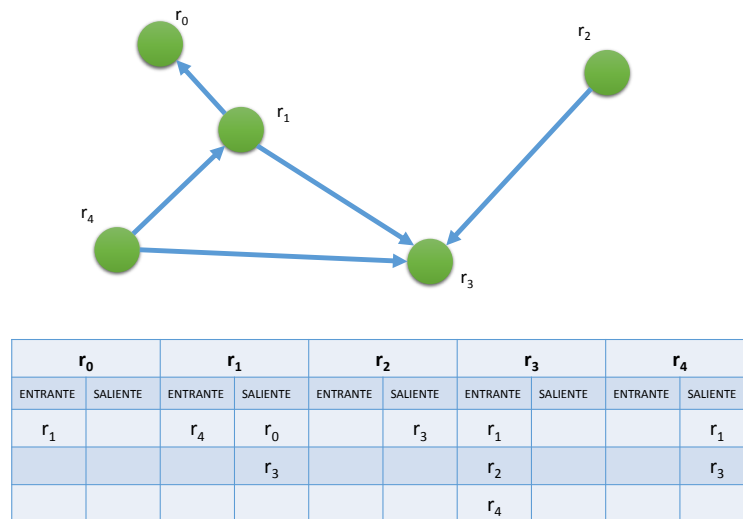


Figura 5.33: Esquema que muestra a la Web representada por un grafo dirigido, en el que se registran para cada recurso (un recurso es cualquier documento web que contenga URLs) r_k , todos los recursos r_i que contienen una URL apuntando a r_k , así como aquellos recursos r_j que son los recursos apuntados por medio de URL desde el recurso r_k .

Se propone que las relaciones (r_k, r_i) y (r_j, r_k) sean almacenadas en una base de datos, con el fin de llevar el seguimiento de todos los recursos r_i que tienen una URL apuntando al recurso r_k , así como de aquellos recursos r_j que tienen una URL que apunta al recurso r_k .

Al tener almacenadas las dependencias que hay entre los recursos es posible hacer frente a la operación de borrado de un recurso r_k , operación que afecta directamente a todos los recursos r_i , debido a que cada recurso r_i tiene al menos una liga o hipervínculo apuntando hacia el recurso r_k , lo que bajo una situación normal dejaría a todos los recurso r_i con links rotos. Al llevar el registro de los recursos r_i afectados se pueden tomar medidas en función de las necesidades de cooperación y/o permisos bajo un ambiente colaborativo.

Las operaciones que pueden terminar en hipervínculos rotos son la eliminación del recurso r_k por parte de un colaborador propietario del mismo, o por la reubicación o cambio de nombre del recurso r_k . En el primer caso se tiene la integridad de dependencias y para el caso de reubicar un recurso esta operación sólo involucra eliminar y agregar un registro a la tabla `Colección_recurso`. Operación que no afectaría al resto de URLs que apuntan al recurso.

Capítulo 6

Desarrollo e implementación

En este capítulo se describe el proceso que se siguió para desarrollar el (*framework*) desde la selección de la tecnología (ver Sección 6.1), pasando por la metodología para el desarrollo (ver Sección 6.2), hasta mostrar partes destacadas del proceso de ingeniería de software tales como: la síntesis de los principales requerimientos del *Framework* (ver Sección 6.3) o el análisis y diseño orientado a objetos (ver Sección 6.5), donde se presentan los principales casos de uso involucrados en el *Framework*. También se presentan detalles de partes destacadas del proceso de desarrollo del *Framework* como lo es el diagrama de clases del modelo de datos (ver Sección 6.6), así como procesos tales como el usado para la compartición de documentos (ver Sección 6.7) y el envío de información (ver Sección 6.8).

6.1. Selección de tecnología

Entre los lenguajes de programación orientados a la creación de aplicaciones web se encuentran los siguientes:

- **Java** [Oracle, 2017]: lenguaje que proporciona un conjunto de *frameworks*¹ y bibliotecas orientados a la creación de aplicaciones web entre las que encontramos: Java Server Pages y Servlets, los cuales proporcionan un marco adecuado para la creación de páginas web dinámicas. Otra tecnología que ofrece Java es Java Server Faces, la cual es un *framework* que permite crear aplicaciones web basadas en el patrón de diseño MVC2. Además Java cuenta con otras tecnologías muy útiles, tales como JPA², la cual es un *framework* que implementa el mapeo entre el paradigma orientado a objetos y el modelo relacional.
- **ASP.NET** [Microsoft, 2017]: entre las tecnologías más conocidas están Web Forms,

¹Un *framework* es un conjunto de bibliotecas y/o herramientas que encapsulan patrones de diseño, componentes de software genéricos y que por lo general proporcionan un comportamiento predefinido, el cual puede ser fácilmente extendido

²Java Persistence API

la cual tiene encapsulados componentes HTML, para un rápido desarrollo de formularios web. ASP MVC es un *framework* similar a Java Server Pages.

- **PHP [Group, 2017]:** es un lenguaje originalmente creado para el desarrollo de páginas web dinámicas. Ahora un lenguaje de propósito general, el cual maneja múltiples paradigmas tales como el procedural es o el orientado a objetos.

El lenguaje seleccionado para la implementación del *Framework* es Java debido a que cuenta con un amplio conjunto de bibliotecas y *frameworks* estables que facilitan el desarrollo de aplicaciones web. En específico se seleccionó la plataforma Java EE, la cual proporciona una API³ y entornos de ejecución para diseñar y ejecutar aplicaciones con las siguientes características: multi-capa, escalables, confiables y seguras. Además de que Java es un lenguaje orientado a objetos, cuyos programas se compilan una vez y son capaces de ejecutarse en cualquier plataforma, lo que proporciona un entorno homogéneo, con respecto a las diferencias entre sistemas operativos.

Para aprovechar las características de la Web, Java EE ofrece la API JAX-RS, la cual usa HTTP para obtener (GET), enviar (POST), actualizar (PUT) o borrar (DELETE) datos. Esta tecnología es conocida como servicios web (*Web Services*) es una tecnología para intercomunicar ya sea computadoras, celulares, electrodomésticos inteligentes o cualquier dispositivo capaz de entender servicios web. Existen dos tipos de Servicios Web: por un lado esta SOAP⁴ y por el otro se tiene a RESTful⁵. En términos generales Web Services es similar a RPC⁶ o a Java RMI⁷ y más comúnmente comparado con CORBA, lo que significa que Servicios web fue creado con el fin de invocar métodos o procedimientos que están en computadoras remotas.

Originalmente SOAP, WSDL⁸ y UDDI⁹ forman parte de la especificación original de los servicios web. Sin embargo, SOAP no requiere de WSDL y UDDI para implementar un servicio web. SOAP es un protocolo de comunicación de aplicaciones, el cual contempla un formato para enviar y recibir mensajes sobre HTTP basado en XML. Otra característica importante de esta tecnología es su independencia con respecto al sistema operativo y al lenguaje de programación. SOAP es similar a CORBA ya que, además de definir el formato de los mensajes (un documento XML), también define la manera en la que los tipos de datos serán codificados, definiendo desde cadenas de texto hasta números de punto flotante de doble precisión, pasando por boléanos. Aunque SOAP está orientado a usar HTTP como protocolo de transporte, no tiene por qué ser así, ya que es posible usar algún otro protocolo

³Es la palabra que en inglés significa *Application Program Interface* que es un conjunto de bibliotecas, protocolos y herramientas para construir aplicaciones software.

⁴*Simple Object Access Protocol*

⁵*Representational State Transfer*

⁶*Remote Procces Call*

⁷*Remote Method Invocation*

⁸palabra que en inglés significa *Web Service Description Language*, el cual sirve para describir los servicios disponibles en un servidor.

⁹palabra en inglés para designar *Universal Description, Discovery, and Integration*, la cual es una especificación para un registro distribuido de servicios web el cual se apoya de WSDL.

de transporte tal como SMTP¹⁰ o FTP¹¹.

Por otro lado, REST o RESTful es una serie de restricciones que proponen una arquitectura tipo SOA¹². La idea general detrás de la implementación de esta arquitectura es de trabajar sobre HTTP, donde cada URI corresponde a un único recurso, el cual puede tener varias representaciones. Se puede interactuar con los recursos por medio de los métodos del protocolo HTTP, i.e., por ejemplo usar GET para obtener el contenido del recurso, eliminar un recurso con DELETE, crear un recurso con POST o actualizar un recurso con PUT. Una de las características principales de los servicios web es que reduce el acoplamiento entre el cliente y el servidor. Las representaciones de los recursos pueden ser en XML, en texto plano, o algún otro tipo. A diferencia de SOAP, aquí no hay un archivo XML que defina la estructura de los mensajes, ni tampoco los tipos de datos, simplemente se puede enviar y/o recibir texto plano, XML o bien JSON. Para aprovechar las ventajas que ofrece la Web, se utiliza RESTful para implementar la comunicación. En específico la implementación denominada JAX-RS, la cual forma parte de la plataforma Java EE. Se seleccionó esta tecnología gracias a su simplicidad y a que RESTful no requiere enviar datos adicionales a los necesarios.

6.2. Metodología de desarrollar del sistema

La metodología seleccionada para desarrollar la aplicación es el enfoque unificado (*Unified Approach*) el cual contempla una iteración de tres fases importantes que son las siguientes:

- **Análisis:** en esta etapa se extraen las necesidades y todo aquello que el *Framework* debe de hacer para satisfacer los requerimientos de los usuarios. Al mismo tiempo que se entiende el dominio del problema, poniendo atención en describir lo que el *Framework* debe de hacer. Los principales pasos de esta etapa son los siguientes:
 - Identificación de autores.
 - Desarrollar un modelo simple del proceso de negocio.
 - Desarrollar los casos de uso.
 - Desarrollar diagramas de interacción.
 - Identificación de clases.
- **Diseño:** se deben aplicar los axiomas del paradigma orientado a objetos para generar lo siguiente:
 - Diseño de clases.
 - Diseño de las capas de acceso.

¹⁰Simple Mail Transfer Protocol

¹¹File Transfer Protocol

¹²Service-Oriented Architecture

- Diseño y prototipo de interfaces.
- **Desarrollo y prototipo incremental:** iterar en los pasos anteriores, refinando el diseño y prototipos.

6.3. Requerimientos del *Framework*

A manera de síntesis se enlistan los requerimientos que el *Framework* debe de cumplir, todos estos provienen de las especificaciones de PIÑAS (ver Sección 4.1) y Alliance (ver Sección 4.2):

ID	Descripción
R001	El <i>framework</i> debe dar soporte al trabajo colaborativo distribuido.
R002	El <i>framework</i> debe permitir una interacción asíncrona entre los colaboradores.
R003	El <i>framework</i> debe de permitir la definición de documentos formados por N fragmentos (áreas de trabajo).
R004	El <i>framework</i> debe dar soporte para definir áreas de trabajo, a los cuales se les puede asignar colaboradores con roles de escritor, lector, nulo y administrador.
R005	El <i>framework</i> debe considerar que el área de trabajo a_j de un documento d_i se encuentra distribuida en m sitios diferentes.
R006	El <i>framework</i> debe considerar que sólo una de las m áreas de trabajo es el área maestra y el resto de las $m - 1$ áreas son esclavas.
R007	El <i>framework</i> identifica como área de trabajo maestra a aquella que está asociada con el colaborador con rol de escritor. Sólo un colaborador en un momento dado puede tener el rol de escritor.
R008	El <i>framework</i> debe permitir que cada área de trabajo a_j tenga asociado r número de colaboradores, con el rol de lector, s número de colaboradores con el rol de nulo y uno sólo con el rol de administrador y de igual manera con el de escritor.
R009	El <i>framework</i> debe proporcionar mecanismos para distribuir las áreas de trabajo maestras, al mismo tiempo que reemplaza las $m - 1$ esclavas.
R010	El <i>framework</i> debe permitir la inserción de recursos en las áreas de trabajo tales como texto, audio, imágenes, video o documentos.

R011	El <i>framework</i> debe encargarse de la distribución de los recursos pertenecientes a un documento, entre todos los colaboradores involucrados en el mismo.
R012	El <i>framework</i> debe dar soporte al trabajo ante el evento de desconexión.
R013	El <i>framework</i> debe dar soporte a operaciones sobre los recursos tales como cambios en sus propiedades y eliminación de los mismos, siempre manteniendo la consistencia de los documentos. La realización de tales operaciones no debe afectar a otros colaboradores y/o documentos.

Los requerimientos técnicos son los siguientes:

ID	Descripción
R014	El <i>framework</i> deberá funcionar sobre la Web.
R015	El <i>framework</i> deberá permitir la creación de sistemas distribuidos.
R017	El <i>framework</i> deberá emplear tecnologías independientes del sistema operativo.
R018	El <i>framework</i> deberá usar estándares de la industria en su implementación.
R019	El <i>framework</i> deberá contar con una arquitectura escalable.
R020	El <i>framework</i> deberá proporcionar una API simple de entender y de usar.

6.4. Arquitectura

Los principales objetos que conforman el *framework* son los siguientes (ver Figura 6.1):

- Objetos del modelo de datos: conjunto de objetos que modela las entidades principales de la plataforma PIÑAS como lo son las definiciones de autor, las colecciones, los recursos, etc. Estos objetos son creados con la tecnología Entity Beans de Java.
- Objeto de acceso a datos: este objeto se encarga de todo lo relacionado con la transferencia de datos desde y hacia la base de datos. Objetos desarrollados con el *framework* JPA.
- Objetos de comunicación: estos objetos se encargan de ofrecer servicios web para distribuir y recibir objetos del modelo de datos. Objetos creados con la tecnología RESTful.
- Objeto encargados del algoritmo para distribuir los objetos del modelo.

- Objeto encargado de la conversión de objetos Java a archivos XML. Objeto creado con la tecnología JAXB.
- Objetos encargados del proceso de compartición, distribución y edición de documentos compartidos. Objeto creado con los Stateless Session Bean.
- Objeto controlador: objeto principal de interacción con el *framework*. Objeto creado con la tecnología Stateless Session Beans.

El *framework* ayuda a implementar los sitios de almacenamientos definidos en la plataforma PIÑAS en la forma de EJBs por lo que los sitios de almacenamiento estarían ubicados en la capa de negocio del modelo de aplicaciones multicapa distribuida de JavaEE. Por lo que las aplicaciones web desarrolladas con el *framework* de esta tesis se implementarían bajo este contexto (ver Figura 6.2).

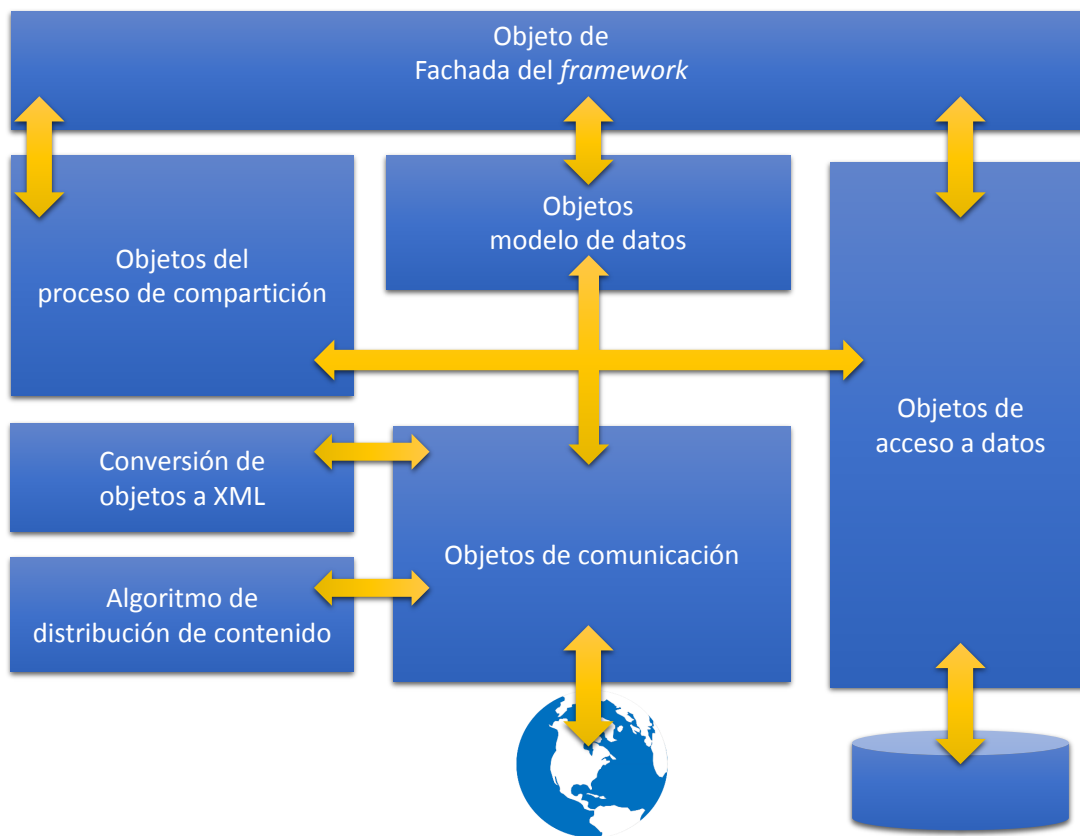


Figura 6.1: Objetos principales del *framework*.

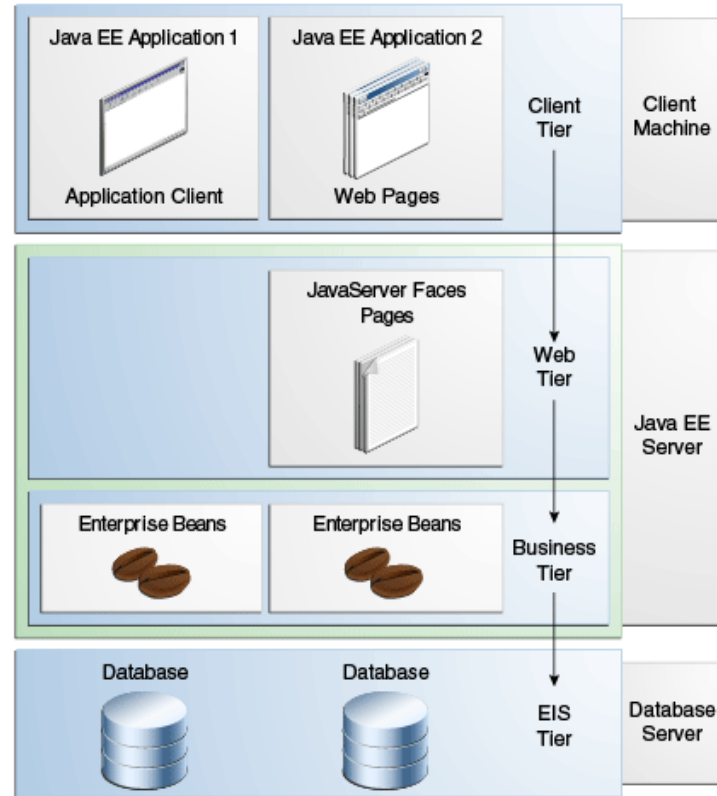


Figura 6.2: Modelo de aplicaciones multicapa distribuidas.

6.5. Análisis y diseño del *framework*

Siguiendo la metodología del enfoque unificado, se procede a realizar la etapa de análisis, para lo cual se identifican los principales casos de uso que ofrecerá el *Framework*. Se contempla que el *Framework* de soporte los siguientes casos de uso (ver Figura 6.3):

- Registro de colaboradores.
- Registro de documento compartido.
- Definición de fragmentos (áreas de trabajo).
- Distribución de documento compartido.
- Distribución de recurso.
- Eliminación de recurso.

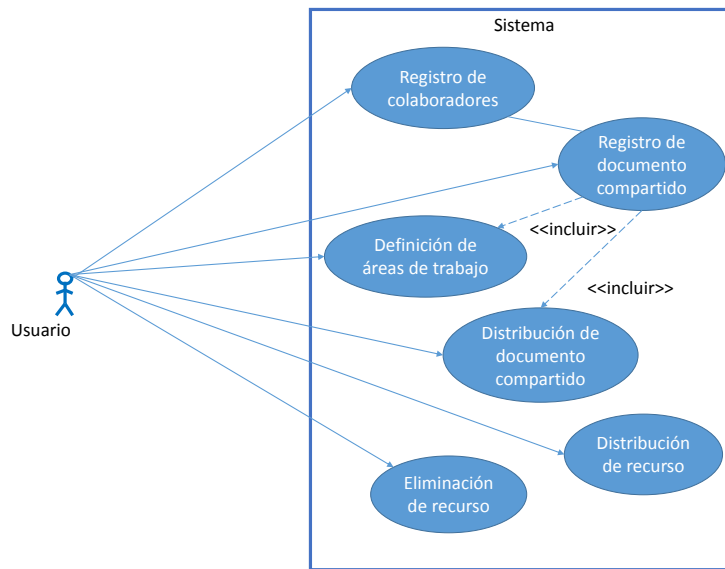


Figura 6.3: Diagrama general de casos de uso del *Framework*.

Tal como se puede apreciar en el diagrama de casos de uso de la Figura 6.3 el autor principal de los casos de uso es un desarrollador de aplicaciones, quien es el que interactuara directamente con los métodos del *framework* creado.

Caso de uso para el registro de colaboradores

Nombre	Registro de colaboradores	
Descripción	Caso de uso encargado del registro de colaboradores.	
Requerimientos		
Casos de uso relacionados		
Precondiciones		
Actores	<i>Framework</i> Desarrollador	
Secuencia principal		
Núm.	Actor	Descripción
1	<i>framework</i>	El <i>Framework</i> solicita al usuario un nombre de colaborador y una lista con al menos una dirección de sitio de almacenamiento.
2	Usuario	Proporciona un nombre y una lista con a lo menos una dirección de sitio de almacenamiento.

3	<i>Framework</i>	El <i>Framework</i> crea un nuevo colaborador al cual le asigna un ID único entre todos los sitios de almacenamiento, además del nombre y lista de sitios designados por el usuario.
4	<i>Framework</i>	El <i>Framework</i> almacena al nuevo colaborador.
5	<i>Framework</i>	El <i>Framework</i> le regresa al usuario el ID del colaborador creado.
Secuencia alternativa 1		
2.1	Usuario	El usuario proporciona un nombre vacío y/o una lista vacía.
2.2	<i>Framework</i>	El <i>Framework</i> no crea a ningún colaborador.
2.3	<i>Framework</i>	El <i>Framework</i> no regresa un ID.
Postcondiciones		
Comentarios		

Caso de uso para el registro de documentos compartidos

Nombre	Registro de documento compartido	
Descripción	Caso de uso encargado del registro de un documento compartido.	
Requerimientos		
Casos de uso relacionados	Definición de fragmentos (áreas de trabajo) Distribución de documento compartido	
Precondiciones	Un colaborador ha iniciado sesión o está activo.	
Actores	<i>Framework</i> Usuario	
Secuencia principal		
Núm.	Actor	Descripción
1	<i>Framework</i>	El <i>Framework</i> solicita un colaborador, un nombre de documento, así como una descripción del mismo.
2	Usuario	El usuario proporciona, el colaborador, además del nombre de documento y una descripción del mismo.
3	<i>Framework</i>	El <i>Framework</i> crea un nuevo documento con el nombre y la descripción.

4	<i>Framework</i>	El <i>Framework</i> solicita al usuario que especifique el nombre de un equipo ¹³ asociado al documento, así como una lista de colaboradores, donde los datos principales son el nombre del colaborador su ID y una dirección de sitio de almacenamiento.
5	Usuario	El usuario ingresa un nombre de equipo, y una lista con los datos de al menos un colaborador.
6	<i>Framework</i>	El <i>Framework</i> crea un equipo con el nombre y la lista de colaboradores y lo asocia al documento.
7	Usuario	El usuario indica al <i>Framework</i> que envíe las solicitudes de colaboración a los miembros del equipo.
8	<i>Framework</i>	El <i>Framework</i> verifica que el nombre de documento y equipo no sean cadenas vacías y almacena el documento junto con el equipo y las solicitudes.
9	<i>Framework</i>	El <i>Framework</i> envía solicitudes de colaboración a los miembros del equipo. La información contenida en la solicitud es la siguiente: ID de solicitud, colaborador que ha creado el documento, nombre del documento, una descripción del documento. Al enviar las solicitudes el <i>Framework</i> se queda en espera de que estas sean aceptadas.
10	<i>Framework</i>	El <i>Framework</i> muestra al usuario el estatus de las solicitudes, donde el dato más importante es si fue o no aceptada.
11	<i>Framework</i>	El <i>Framework</i> le muestra al usuario que los miembros del equipo han aceptado la solicitud de colaboración.
12	Usuario	El usuario procede a definir áreas de trabajo.
13	<i>Framework</i>	El <i>Framework</i> llama al caso de uso <i>Definir áreas de trabajo</i> .
14	Usuario	El usuario procede a definir las áreas de trabajo.
15	Usuario	El <i>Framework</i> llama al caso de uso <i>Distribución de documento compartido</i> .
Secuencia alternativa 1		
2.1	Usuario	El usuario ingresa una cadena de caracteres no vacía de documento.
2.2	<i>Framework</i>	El <i>Framework</i> no crea ningún documento y termina.
Secuencia alternativa 2		

¹³Conjunto de colaboradores involucrados en un documento.

4.1	Usuario	El usuario ingresa un cadena de caracteres vacía y/o una lista vacía de colaboradores.
4.2	<i>Framework</i>	El <i>Framework</i> no crea ningún equipo y termina.
Postcondiciones		El <i>Framework</i> ha creado un documento compartido el cual es susceptible de ediciones por parte de los colaboradores.
Comentarios		

Caso de uso para la definición de áreas de trabajo

Nombre	Definición de áreas de trabajo	
Descripción	Caso de uso encargado de definir las áreas de trabajo (fragmentos)	
Requerimientos		
Casos de uso relacionados	Registro de documento compartido	
Precondiciones	El caso de uso Registrar documento compartido se ha iniciado y se ha creado un documento, además de que al menos una de las solicitudes de colaboración fue aceptada.	
Actores	<i>Framework</i> Usuario	
Secuencia principal		
Núm.	Actor	Descripción
1	<i>Framework</i>	El <i>Framework</i> solicita al usuario que defina la cantidad de áreas de trabajo (N).
2	Usuario	El usuario define N número de áreas de trabajo (fragmentos).
3	<i>Framework</i>	El <i>Framework</i> crea las áreas de trabajo, donde a cada una le asigna un identificador. Y se las muestra al colaborador.
4	Usuario	El usuario selecciona una de las áreas de trabajo.
5	Usuario	El usuario define una dupla que consta de un colaborador más un rol (el cual puede ser ESCRITOR, LECTOR, NULO o ADMINISTRADOR) y la agregar al área de trabajo seleccionada
6	<i>Framework</i>	Sí no hay dupla con el rol de ESCRITOR la agrega a la lista, en el caso de que ya exista, el <i>Framework</i> no agrega nada.
7	Usuario	Si el usuario quiere agregar más duplas ir a siguiente paso, en caso de que no ir al paso número 9.

8	<i>Framework</i>	El <i>Framework</i> se va al paso número 4.
9	<i>Framework</i>	El <i>Framework</i> almacena las áreas de trabajo junto con su lista de duplas.
Secuencia alternativa 1		
2.1	Usuario	El usuario ingresa un número N negativo.
2.2	<i>Framework</i>	permanece en el paso 2.
Postcondiciones		Al terminar este caso de uso se cuenta con un documento dividido en N áreas de trabajo (fragmentos) donde cada una tiene asociado un colaborador con el rol de ESCRITOR, un colaborador con el rol de ADMINISTRADOR, r cantidad de colaboradores con el rol de LECTOR y s cantidad de colaboradores con el rol de NULO.
Comentarios		

Caso de uso para la distribución de documentos compartidos

Nombre	Distribución de documento compartido.	
Descripción	Caso de uso encargado de distribuir el documento entre todos los miembros del equipo.	
Requerimientos		
Casos de uso relacionados	Registro de documento compartido Definición de áreas de trabajo	
Precondiciones	El caso de uso Registrar documento compartido se ha iniciado y se ha creado un documento, además de que al menos una de las solicitudes de colaboración fue aceptada. También ya se han definido áreas de trabajo y se les han asociado colaboradores y roles.	
Actores	<i>Framework</i> Usuario	
Secuencia principal		
Núm.	Actor	Descripción
1	<i>Framework</i>	El <i>Framework</i> procede a identificar los sitios de almacenamiento de los colaboradores.
2	<i>Framework</i>	El <i>Framework</i> recupera las definiciones de autor ¹⁴ de los colaboradores que están en el equipo del documento.

¹⁴una definición de autor está formado por un identificador de colaborador más un conjunto de direcciones a sitios de almacenamiento

3	<i>Framework</i>	El <i>Framework</i> envía a los sitios de los colaboradores del equipo asociado al documento a los colaboradores del documento más el mismo documento.
4	<i>Framework</i>	El <i>Framework</i> espera la confirmación de la entrega del documento y de los colaboradores, por parte de los sitios de los colaboradores.
5	<i>Framework</i>	En caso de error al enviar los colaboradores y documento, el <i>Framework</i> espera un tiempo aleatorio en milisegundos de entre 1000 y 10000 para volver a intentar el envío.
6	<i>Framework</i>	El caso de uso termina al momento de confirmar la recepción de los colaboradores y documento por parte de los sitios de almacenamiento.
Postcondiciones		Al terminar este caso de uso el <i>Framework</i> habrá enviado el documento junto con la lista de colaboradores que participan en la edición del mismo.
Comentarios		

Caso de uso para la distribución de recursos

Nombre	Distribución de recurso.	
Descripción	Caso de uso encargado de distribuir recursos. Este caso de uso entra en juego cuando un colaborador edita un área de trabajo o agrega un recurso al documento.	
Requerimientos		
Casos de uso relacionados		
Precondiciones	Existe un documento definido, el cual tiene un equipo asociado.	
Actores	<i>Framework</i> Usuario	
Secuencia principal		
Núm.	Actor	Descripción
1	<i>Framework</i>	El <i>Framework</i> solicita al usuario que seleccione una colección y el recurso que se desea distribuir.
2	Usuario	El usuario proporciona el recurso y la colección.
3	<i>Framework</i>	El <i>Framework</i> checa la lista de colaboradores del equipo de la colección y obtiene las direcciones de los sitios.

4	<i>Framework</i>	El <i>Framework</i> envía el recurso a cada uno de los sitios de cada uno de los colaborador.
5	<i>Framework</i>	El <i>Framework</i> permanece en espera de confirmar la entrega de los recursos enviados.
6	<i>Framework</i>	Si hay algún error en el envío del recurso entonces regresa al paso 4 para intentar enviarlo de nuevo.
7		Si todos los recursos son enviados el caso de uso termina.
Postcondiciones		Al terminar este caso de uso el recurso se encuentra en todos los sitios de los colaboradores.
Comentarios		

Caso de uso para la eliminación de recursos

Nombre		Eliminación de recurso
Descripción		Caso de uso encargado de eliminar recursos
Requerimientos		
Casos de uso relacionados		
Precondiciones		El recurso a eliminar se encuentra almacenado en el <i>Framework</i> .
Actores		<i>Framework</i> Colaborador
Secuencia principal		
Núm.	Actor	Descripción
1	<i>Framework</i>	El <i>Framework</i> solicita al usuario que indique el recurso y la colección a la cual pertenece el recurso.
2	Usuario	El usuario proporciona el recurso y la colección.
3	<i>Framework</i>	El <i>Framework</i> elimina la referencia hacia el recurso.
4	<i>Framework</i>	El caso de uso termina.
Postcondiciones		Al finalizar este caso de uso el recurso deja de estar asociado a la colección.
Comentarios		

6.6. Diagrama de clases

El modelo de datos se encuentra formado por el siguiente conjunto de clases. En el que la idea principal es modelar a un documento, el cual estará conformado por múltiples fragmentos (áreas de trabajo) y recursos. Una colección corresponde a un documento, un recurso puede ser una imagen, un documento (por ejemplo un PDF), un video, un audio, etc. Las área de trabajo se podrán compartir con otros colaboradores, a los cuales se les podrá conceder el permiso de lectura (rol de LECTOR), de no visualización (rol NULO), pero sólo uno en un momento determinado será capaz de editar el área de trabajo¹⁵ (rol de ESCRITOR), para lograr tal fin se diseñó el siguiente diagrama de clases que permite modelar lo anteriormente dicho.

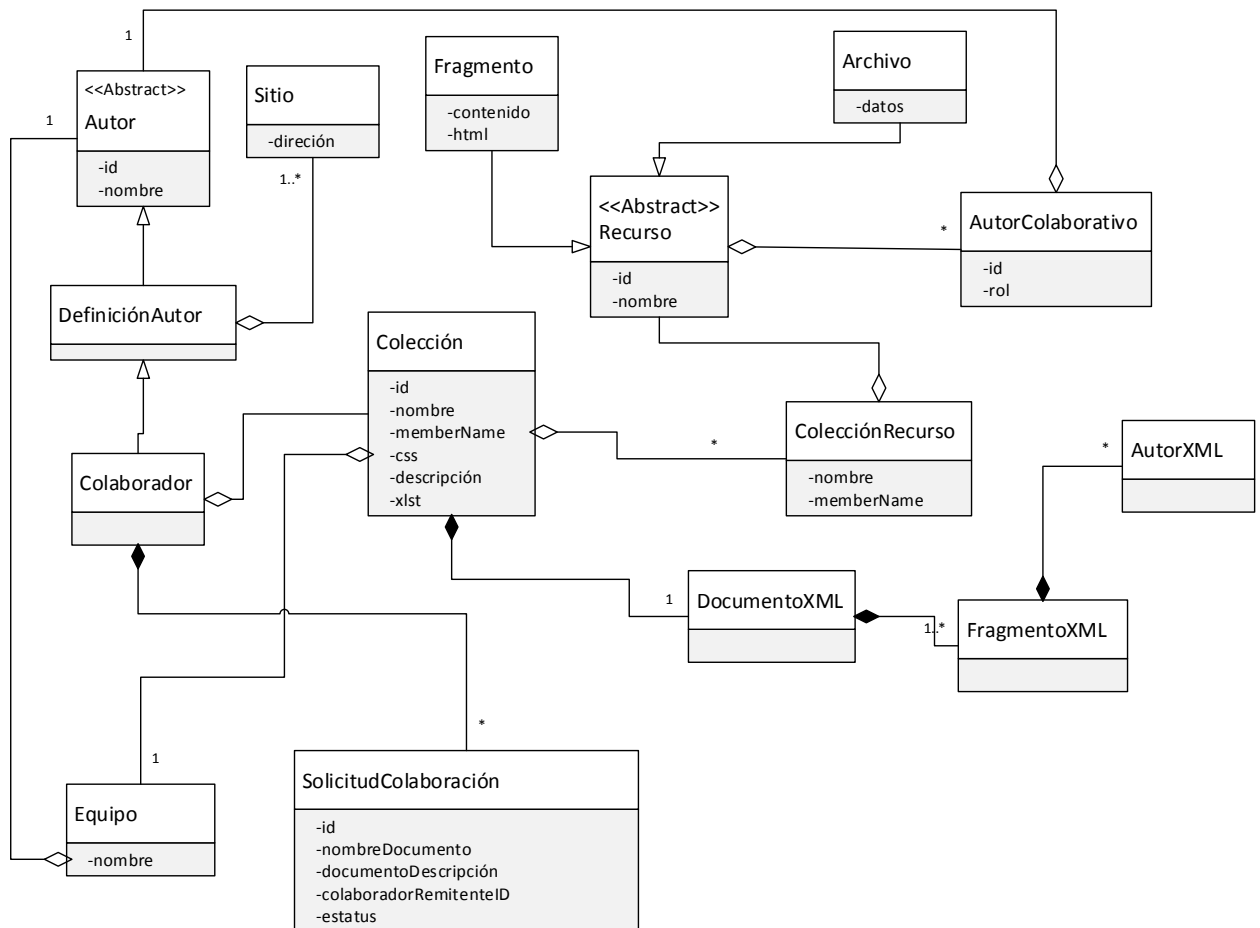


Figura 6.4: Diagrama de clases que representan al modelo de datos.

¹⁵Se debe de señalar que en la especificación de PIÑAS y Alliance se permite que múltiples colaboradores tomen el rol de ESCRITOR así como el de ADMINISTRADOR, sin embargo, en esta tesis no se logra implementar tal característica

En el diagrama de clases de la Figura 6.4 se modelan las entidades principales de la plataforma PIÑAS, entre las que se incluyen la entidad *Base* misma que se representa por medio de la clase `Colaborador`, también las entidades `Colección` y `Recurso` que son representadas con clases del mismo nombre. La clase `Recurso` se utiliza para representar a cualquier tipo de archivo, mientras que la clase `Fragmento` modela las áreas de trabajo las cuales tiene la ventaja de tener una representación en HTML.

6.7. Proceso de compartición y edición de documentos

El proceso para compartir documentos fue implementado usando el **patrón de diseño estado** (*design pattern state*) [Gamma et al., 1995]. El cual sirve para implementar proceso modelados con un diagrama de estados finitos o autómatas. El proceso cuenta con los siguientes estados:

- Nuevo documento compartido.
- Definición de equipo.
- Invitaciones.
- Espera de respuesta de solicitudes.
- Definición de áreas de trabajo.
- Distribución de documento.
- Edición de documento.
- Documento finalizado.

Y las siguientes transiciones:

- Crear documento.
- Crear equipo.
- Enviar solicitudes.
- Solicitudes recibidas.
- Agregar área de trabajo.
- Agregar autor y rol.
- Áreas listas (M áreas definidas).
- Enviar documento.

- Distribuir actualización.
- Edición de área finalizada.
- M áreas finalizadas.

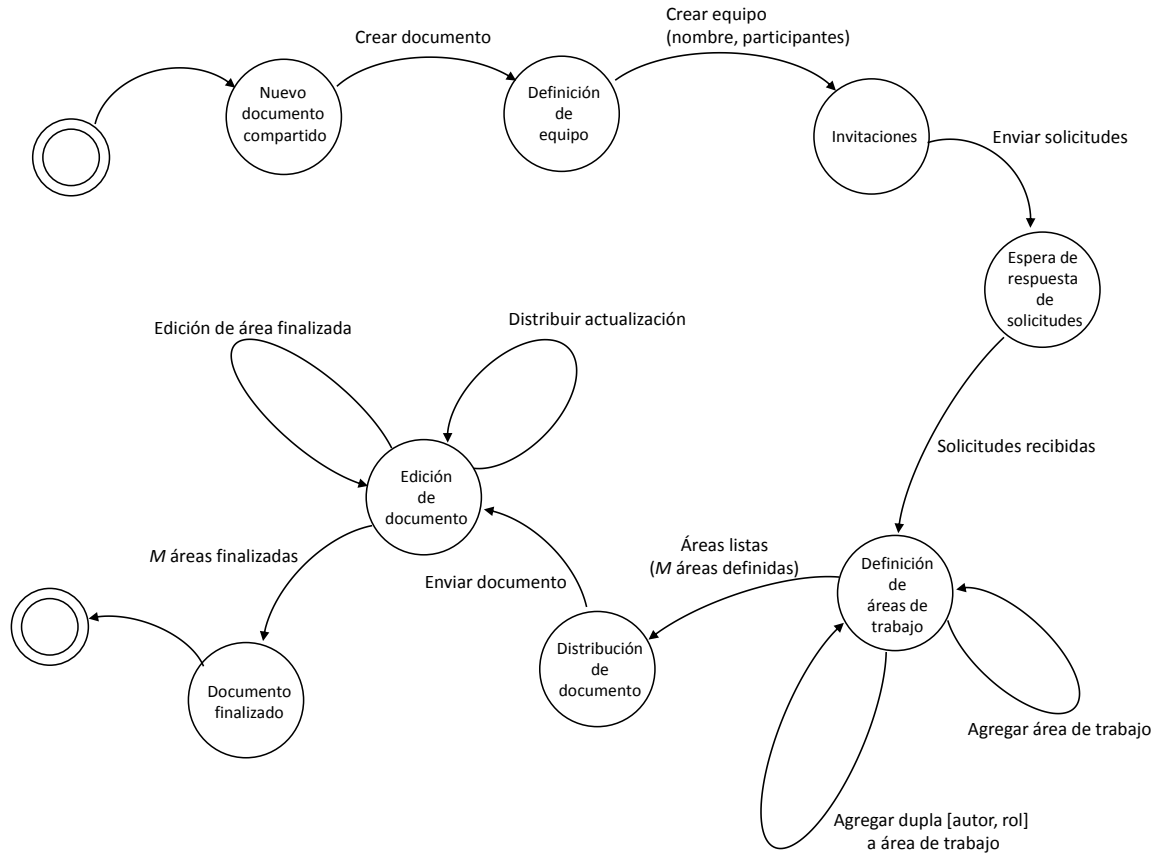


Figura 6.5: Esquema que muestra el proceso de compartición de recursos modelado con un autómata de estados finitos.

En el estado *Nuevo documento compartido*, el *framework* crea un objeto de tipo *Colección*, se instancia un nuevo objeto de tipo *DocumentoXML*, y se asigna nombre y una descripción a la colección. Todo esto por medio de la transición de *Crear documento*. Una vez que se pasa al estado *Definición de equipo* se crea un objeto de tipo *equipo* y se asocia con la colección (por medio de la transición *Crear equipo*). Una vez en el estado *Invitaciones*, por medio de la transición *Enviar solicitudes* se crean las invitaciones para cada uno de los participantes al instanciar objetos de tipo *SolicitudColaboración*. Estos objetos son enviados a los sitios de los participantes. Una vez que se verifica que las solicitudes fueron aceptadas, el colaborador puede continuar con el proceso por medio de la transición *Solicitudes recibidas*. La anterior transición permite que se mueva al estado *Definición de áreas de trabajo*. Cuando se está en este estado el *Framework* permite al colaborador definir áreas de trabajo al mismo tiempo

que permite asignar colaboradores junto con sus roles, esto por medio de las siguientes dos transiciones: *Agregar área de trabajo* la cual permite definir M áreas que conformaran a un documento completo y, *Agregar dupla [autor, rol]* la cual permite asignar colaboradores y roles a cada área de trabajo. Una vez que se ha terminado de definir las áreas de trabajo se puede pasar al estado *Distribución de documento* por medio de la transición *áreas listas*. En este estado el *framework* se encarga de reunir las definiciones de autor de todos los colaboradores que participan en el documento, para que estos sean enviados a los sitios de los colaboradores que participan en el documento. Después el *framework* envía el documento ya con las áreas definidas. Una vez que al *framework* se le notifica que la información fue recibida por los sito de los colaboradores, el proceso pasa al estado *Edición del documento* lo que significa que los colaboradores pueden iniciar el proceso de edición en este estado se puede hacer uso de las siguientes transiciones: *Distribuir actualizaciones*, esta transición se manda a llamar cada vez que un colaborador decide compartir una actualización de un fragmento (área de trabajo) y/o recurso del documento compartido, la otra transición es *Edición de área finalizada* la cual indicaría que el fragmento (área de trabajo) ya no sufrirá más modificaciones. En el momento en el que el *framework* detecte M áreas finalizadas, entonces el *framework* pasara al estado de *Documento finalizado*, momento en el que termina el proceso de compartición y edición del documento (ver Figura 6.5). Se debe hacer notar que cada una de las transiciones del autómata corresponde a métodos correspondientes al objeto de tipo `ProceosComparticionDocumento`.

6.8. Envío de información

Para enviar cualquier objeto (colecciones, colaboradores, recursos, etc) se usa el Algoritmo 1.

Algorithm 1 Algoritmo para enviar: colecciones, recursos, colaboradores, definiciones de autor entre otros.

Require: Objeto a ser enviado, conjunto de destinatarios.

Ensure: No regresa nada.

```

1: while conjunto de destinatarios no vacío do
2:   for cada destinatario d do
3:     if enviarObjeto(objeto,d) es true then
4:       Eliminar d del conjunto de destinatarios.
5:     end if
6:   end for
7:   if conjunto de destinatarios no vacío then
8:     Pausar el proceso y esperar un tiempo aleatorio.
9:   end if
10: end while

```

El Algoritmo 1 básicamente indica que se debe de especificar el objeto a ser enviado y

un conjunto de destinatarios en la forma de una lista de objetos de tipo *DefinicionesAutor*, los cuales a su vez contienen un conjunto de sitios de almacenamiento, a los que se les hará llegar el objeto a enviar. Una vez especificados los datos de entrada se procede a enviar la información. En principio se intenta enviar a cada uno de los destinatarios el objeto. Si el proceso de enviar el objeto tiene éxito entonces se elimina de la lista al destinatario que ya recibió el objeto, si esto falla el *Framework* espera un tiempo aleatorio para volver a intentarlo con los destinatarios a los que no se pudo hacer llegar la información y el proceso se repite hasta que quede la lista vacía.

En el Algoritmo 1 se usa un función llamada `enviarObjeto` la cual sabe cómo enviar objetos usando la tecnología de RESTful. La clase que implementa este algoritmo es abstracta, con el fin de que se implemente de manera separada e independiente el método para enviar objetos, esto con el fin de reducir el acoplamiento y la dependencia con la tecnología seleccionada. Las clases que heredan de la clase abstracta `EnviarObjeto` son las encargadas de implementar el método `enviarObjeto`, las cuales necesariamente deben contener la información necesaria para comunicarse mediante RESTful. Es importante señalar que para este caso se implementó el **patrón de diseño estrategia** [Gamma et al., 1995]. La clase denominada `Comunicación` es la encargada de implementar la comunicación mediante RESTful usando JAX-RS (ver Figura 6.6).

6.9. Interfaz

Los desarrolladores pueden interactuar con el framework mediante la clase `Controlador` que implementa el **patrón fachada** [Gamma et al., 1995]. Es por medio de esta clase que se instancian todos los objetos del *framework*. La creación de los objetos por medio del operador `new` es bloqueado para que la única manera de crear objeto sea a través de la fachada, se debe mencionar que la creación de objetos es por medio del patrón **Simple Factory** [Gamma et al., 1995]. Algunos métodos que podemos encontrar son los siguientes (En la Figura 6.7 podemos encontrar un fragmento de los métodos disponibles):

- Crear colaboradores (definiciones de autor).
- Crear colecciones.
- Definir áreas de trabajo (fragmentos).
- Asignar autores y roles a áreas de trabajo.
- Crear equipos.
- Asociar un equipo con colección.
- Distribuir colecciones entre los miembros de un equipo.
- Distribución de recursos entre los miembros de un equipo.

- Eliminar un recurso de una colección.
- Persistencia de los objetos del modelo.

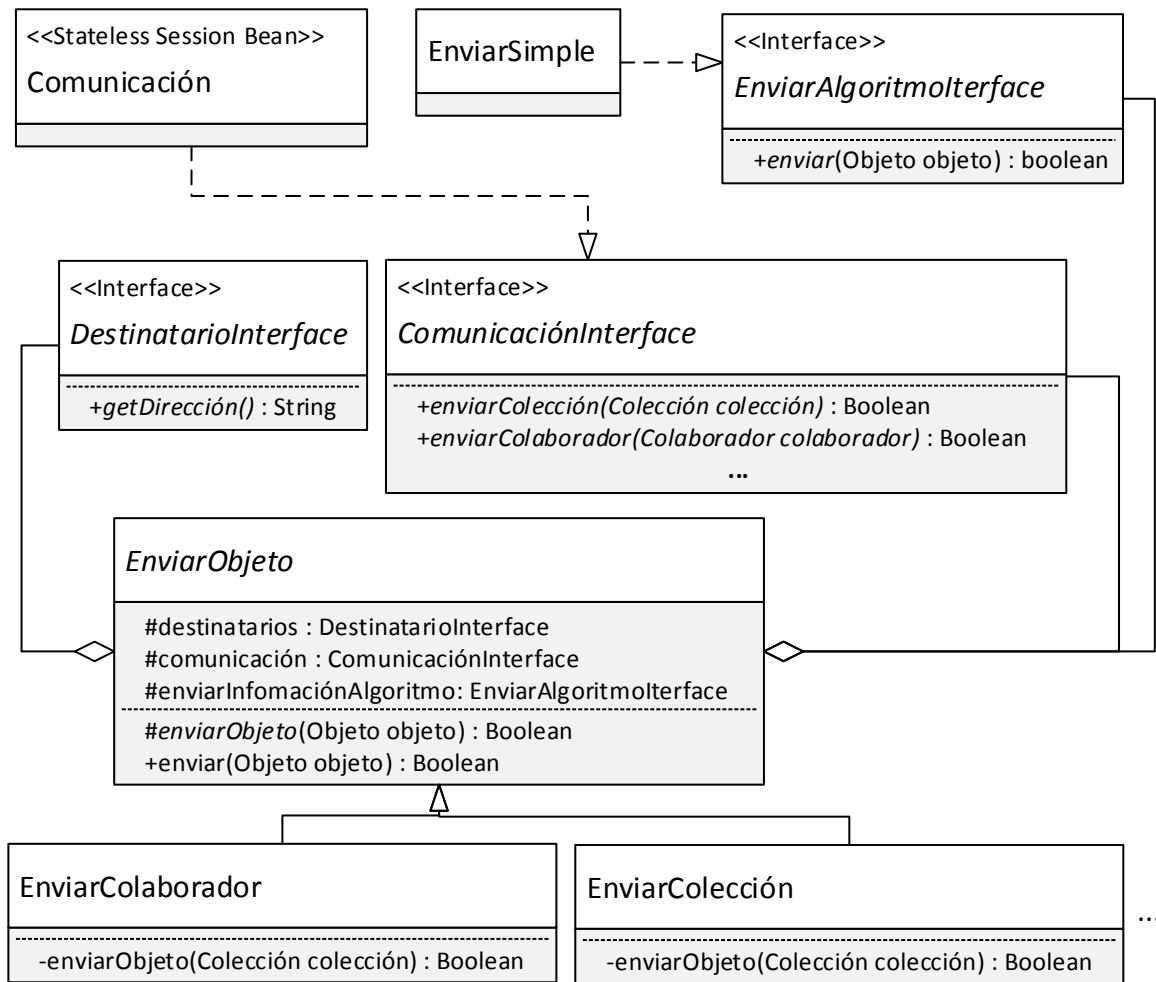


Figura 6.6: Clases involucradas en el envío de información.

Para utilizar el *framework* se necesita una instancia del objeto **Controlador**, esta instancia es la que proporciona acceso a todas las utilidades del *framework*. El *framework* fue creado en base a EJBs lo que permite solicitar al contenedor de EJBs que nos proporcione una instancia, lo que se hace con la anotación `@EJB`.

```

@EJB
Controlador control;
  
```

Para crear un objeto del modelo del *framework* se invoca a alguno de los métodos de la instancia Controlador por ejemplo para registrar un colaborador nuevo se hace lo siguiente:

```
Colaborador colaborador = control.registrarColaborador(nombreColaborador);
```

Para el caso de obtener un colaborador previamente registrado se usa el siguiente método:

```
Colaborador colaborador = control.buscarColaborador(nombreColaborador);
```

Para iniciar un documento en el que podrán colaborar varios usuarios se manda a llamar al siguiente método con el parámetro nombre del colaborador:

```
ProcesoCompartirDocumento pcd;  
pcd = control.crearProcesoCompartirDocumento(nombreColaborador);
```

```
38 public interface ControladorBean {  
39  
40     public void aceptarSolicitudColaboracion(Colaborador colaborador, String solicitudId);  
41  
42     public void aceptarSolicitudColaboracion(Colaborador colaborador, SolicitudColaboracion sc);  
43  
44     public Colaborador actualizar(Colaborador obj);  
45  
46     public Archivo agregarArchivo(String nombre, byte[] datos);  
47  
48     public void agregarAutorRol(Coleccion coleccion, Fragmento fragmento, Colaborador autor, Rol rol);  
49  
50     public Colaborador agregarColaborador(String nombreColaborador);  
51  
52     public Equipo agregarColaborador(Equipo equipo, Colaborador colaborador);  
53  
54     public Coleccion agregarColeccion(Colaborador colaborador, String nombreColeccion, String descripcion);  
55  
56     public Equipo agregarEquipo(Coleccion coleccion, String equipoNombre);  
57  
58     public String agregarFragmento(Colaborador colaborador, Coleccion coleccion);  
59  
60     public Coleccion agregarRecurso(Coleccion coleccion, Recurso recurso, String nombreRecurso);  
61  
62     public Colaborador agregarSitio(Colaborador colaborador, String sitio);  
63  
64     public String almacenar(Colaborador colaborador);  
65  
66     public String almacenar(DefinicionAutor definicionAutor);
```

Figura 6.7: Parte del conjunto de métodos que ofrece el *Framework*.

Capítulo 7

Pruebas

En este capítulo se presentan las pruebas realizadas al *framework* desarrollado en esta tesis. Primero se presenta las características y objetivo de las pruebas (ver Sección 7.1). Para probar el *framework* se decidió usarlo en el desarrollo de una aplicación web, la que se presenta su arquitectura, su flujo de pantallas y el proceso que se debe de seguir para utilizar el *framework* (ver Sección 7.2). Una vez que se presentan las características de la aplicación web se procede a probar el mecanismo de distribución de recursos (ver Sección 7.3). Y para finalizar se presenta las pruebas realizadas a las URLs con el fin de mostrar que el *framework* puede manejar múltiples URLs para referenciar a un mismo recurso (ver secciones 7.4 y 7.5).

7.1. Características y objetivo de las pruebas

El principal objetivo de las pruebas consiste en analizar el comportamiento del *framework* con el fin de apreciar la implementación de las especificaciones de PIÑAS/Alliance. En particular se desea observar el uso de los *identificadores globales* para los colaboradores, colecciones y recurso, también se desea observar que los datos viajan empaquetados en XML y viajan por HTTP, se quiere apreciar el uso de las URLs como medio para referenciar objetos.

Los usuarios del *framework*, interactuarán con los objetos del modelo de datos (ver Sección 6.6), donde principalmente harán referencia a los colaboradores, colecciones y recursos por medio de los identificadores globales.

El sistema desarrollado en esta tesis es un *framework* que viene empaquetado en un archivo `.jar` el cual puede ser importado a cualquier aplicación web o bien puede instalarse previamente en un servidor de aplicaciones.

Para probar el *framework* se creó una aplicación web que hiciera uso del *framework* con el fin de apreciar las facilidades que ofrece a los desarrolladores. Las funciones del *framework* que se probaron son las siguientes:

- Registro de colaboradores con el registro de sitios de almacenamiento.

- Creación de documento compartido.
- Recepción de solicitudes.
- Uso del protocolo HTTP en conjunción con XML para el envío y recepción de información.
- Prueba de la independencia de las URLs con respecto al cambio de nombre y/o eliminación de un recurso.
- Prueba sobre el uso de múltiples URLs para apuntar a un mismo recurso.

Para el desarrollo de aplicaciones web, Java utiliza el patrón de diseño MVC2¹, en las que el modelo se implementa mediante *Entity Beans*, la parte del controlador se hace usando los llamados *Managed beans* y para la parte de la vista se usa la tecnología *Java Server Faces* (JSF).

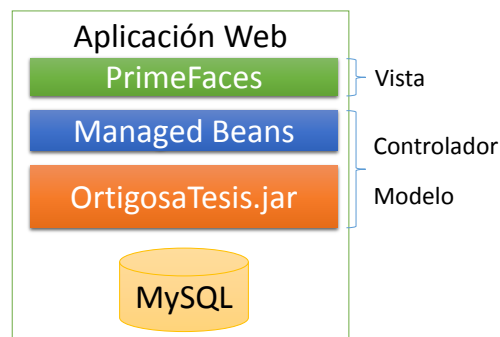


Figura 7.1: Arquitectura de la aplicación web de prueba.

En el caso de la aplicación desarrollada para probar el uso del *framework*, se opta por usar PrimeFaces² 5.0 [PrimeTek, 2017] en lugar de *Java Server Faces*.

Para la parte del controlador se usará los *Managed Beans* junto con el *framework*, para el almacenamiento de los datos se usa como gestor de bases de datos a MySQL (ver Figura 7.1). El servidor de aplicaciones que se utilizara para realizar las pruebas es Payara Server 173.

7.2. Desarrollo de la aplicación de prueba

Esta aplicación de prueba constara con un flujo determinado de páginas web el cual estará controlado con el objeto principal del *framework*, el objeto es de tipo `ProcesoCompartirDocumento`

¹Model View Controller

²PrimeFaces es un *framework* basado en *Java Server Faces*, con componentes web adicionales y facilidades para el uso de AJAX.

(En la Figura 7.2 se muestra el conjunto de páginas con las que contara la aplicación de prueba). Las páginas web corresponden a la capa de vista (del modelo MVC2), las cuales fueron desarrolladas con PrimeFaces.

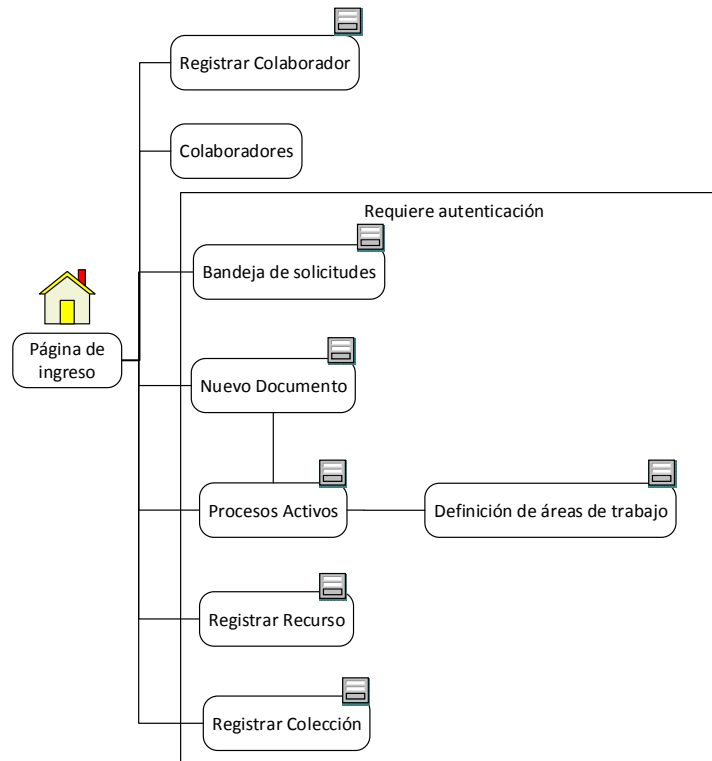


Figura 7.2: Flujo de pantallas de la aplicación web de prueba.

Las páginas que no necesitan que un usuario este registrado son las siguientes:

- Registro de colaboradores: página que permite el registro de colaboradores en el sistema.
- Colaboradores: página que permite explorar el contenido de la base de datos.

Aun cuando la página *Registro de colaboradores* debe ser accesible por un usuario con los privilegios necesarios para tal fin, en esta aplicación de prueba no se consideran permisos de acceso a partes de la aplicación web.

Las páginas que requieren autenticación no es por motivos de seguridad sino porque se requiere saber que usuario esta interactuando con la aplicación.

Las páginas que necesitan que un colaborador esté identificado son las siguientes:

- Bandeja de solicitudes: esta página se encarga de mostrar una bandeja con todas las solicitudes de colaboración recibidas.

- Nuevo documento: esta página le permite a un colaborador definir un documento que será compartido entre varios colaboradores.
- Procesos activos: esta página muestra el registro de todos los procesos de compartición de documento iniciados y que están en proceso.
- Definición de áreas de trabajo: esta página permite crear áreas de trabajo (fragmentos) así como asociarles colaboradores y roles a cada una de ellas.
- Registrar recurso: esta página permite registrar un recurso en una colección³, recurso que será distribuido por el sistema a los sitios de almacenamiento correspondientes.
- Registrar colección: página que permite registrar colecciones.

El desarrollo de aplicaciones web con la tecnología java ofrece dos tecnologías una es por medio de los llamados JSP⁴ que son archivos con etiquetas HTML a las que se les pueden agregar incrustaciones de código Java por medio de *scriptlets* y de manera análoga con *Servlets* los cuales son lo contrario a los JSPs en este caso se trata de aplicaciones que se ejecutan en un servidor y donde su salida estándar es texto con etiquetas HTML.

La otra tecnología que ofrece para crear aplicaciones web Java es JSF (*Java Server Pages*) el cual separa la parte referente a la vista de la parte controladora y del modelo. La vista se construye con etiquetas muy similares a HTML las cuales tiene la particularidad de poder acceder directamente a los *datos miembro* de un tipo de objeto especial conocido como *Managed Bean*, objeto que contiene la lógica del negocio.

Algo que se debe mencionar es que todas las vistas de esta aplicación de prueba fueron creadas con el *framework PrimeFaces*. En esta aplicación la única función de las vistas será la de presentar y recolectar datos, toda la lógica de la aplicación estará concentrada en los *Managed Beans*.

PrimeFaces ofrece ayuda con respecto a mostrar y capturar datos. Los datos que ingresan los usuarios son almacenados en los *datos miembro* de los *Managed Bean*. Otro aspecto importante que se usa de *PrimeFaces* es que las acciones de los usuarios pueden asociadas a métodos pertenecientes a los *Managed Bean*. Por ejemplo si un usuario hace *click* en un botón de la vista este dispara la ejecución de un determinado método de un determinado *Managed Bean*.

Para hacer uso del *framework* un desarrollador sólo necesita colocar, como dato miembro de un *managed bean*, una referencia tipo *Controlador* con su respectiva anotación *@EJB* para que el contenedor de EJBs se encargue de instanciar el objeto. Esto es todo lo que el desarrollador debe de hacer para llamar al *framework*.

³Se debe de recordar que una colección es un contenedor que agrupa los recursos y fragmentos asociados a un documento compartido.

⁴*Java Server Pages*

7.2.1. Registrar colaborador

Para registrar un colaborador solo se necesita de un **nombre** el cual será almacenado en el dato miembro `colaboradorNombre`. Una vez que el usuario ingresa un nombre este puede mandar a registrar a tal usuario dando *click* en el botón *agregar* (ver Figura 7.3), la acción que se ejecutara cuando el usuario presione el botón será el código del método `agregarColaborador()` (ver Figura 7.4).

Tal como se observa en la Figura 7.4 es en el método `agregarColaborador()` donde se manda a llamar a una funcionalidad del *framework*. En este caso se manda a llamar al método del framework encargado de registrar colaboradores, método que sólo necesita como parámetro el nombre del colaborador.

Esta vista también permite asociar sitios a los colaboradores, para tal fin se manda a ejecutar el método `agregarSitio()` (ver Figura 7.4) el cual está asociado a otro botón dentro de la vista. Es justo este método el que invoca al método `agregarSitio(Colaborador colaborador, String sitio)` del objeto Controlador.

Gracias al *framework* el desarrollador no tiene que preocuparse por cuestiones relacionadas al almacenamiento tales como crear tabla o, crear consultas. Lo único que debe de hacer el desarrollador es usar los objetos que proporciona el *framework* e invocar sus métodos.

ID	Nombre
192.168.100.103_0	ortigosa
192.168.100.103_1	jose
192.168.100.103_2	luis

Figura 7.3: Página de registro de colaborador.

```
@Named
@ViewScoped
public class RegistrarColaborador implements Serializable {

    @EJB
    private Controlador controlador;
    private String colaboradorNombre;
    private String sitio;
    private Colaborador colaborador;

    public void agregarColaborador() {
        colaborador = controlador.agregarColaborador(colaboradorNombre);
    }

    public void agregarSitio() {
        colaborador = controlador.agregarSitio(colaborador, sitio);
    }
}
```

Figura 7.4: Fragmento de código perteneciente al *Managed Bean* encargado de registrar colaboradores.

7.2.2. Ingreso a la aplicación de prueba

Lo normal en las aplicaciones web es tener objetos dedicados a la validación de los datos ingresados así como objetos dedicados a la verificación de las credenciales de los usuarios. Sin embargo este no es el objetivo de esta página de ingreso, el único fin de crear esta página es crear una sesión para mantener los datos correspondientes al usuario que está usando la aplicación en determinado momento y poder así determinar qué datos mostrarle en pantalla.

Una vez que los datos son capturados por un usuario con ayuda de la página mostrada en la Figura 7.5, estos datos son enviados a los datos miembros del Managed Bean que se muestra en la figura 7.6.

El *dato miembro* `usuarioNombre` se encarga de almacenar el nombre de colaborador ingresado por el usuario, este dato es pasado al método del objeto principal del *framework* `buscarBaseColaborador(String colaboradorNombre)` este se encarga de regresar un objeto tipo `Colaborador` sólo si se encuentra, en caso de no encontrar al colaborador regresa `null` (ver Figura 7.6). El colaborador encontrado es almacenado en una sesión, para que pueda ser usado en todo momento hasta que el usuario cierre la sesión.

Del lado del desarrollador la única acción que hizo fue manda a buscar al colaborador, el desarrollador no se tuvo que preocupar por crear una consulta a base de datos.



Figura 7.5: Página de ingreso al sistema.

```

@Named
@SessionScoped
public class UserManagerBean implements Serializable {

    @EJB
    private Controlador controlador;
    private Colaborador user;
    private String usuarioNombre;

    public String login() {
        if(usuarioNombre == null || usuarioNombre.isEmpty()) {
            addMsg("Error", "ingresa datos validos");
            return "login";
        }
        user = controlador.buscarBaseColaborador(usuarioNombre);
        if (user != null) {
            FacesContext.getCurrentInstance().getExternalContext().getSessionMap()
            return "solicitudesColaboracion";
        } else {
            addMsg("Error", "datos incorrectos");
            return "login";
        }
    }
}

```

Figura 7.6: Fragmento de código para el *Managed Bean* asociado a la pantalla de ingreso.

7.2.3. Crear un documento compartido

La creación de un documento compartido se implementó en varias páginas web, la primera se encarga de crear el grupo de usuarios que colaboraran en el documento (ver Figura 7.7), para tal fin un desarrollador sólo necesita mandar a llamar al método `crearProcesoCompartirDocumento()` este metodo se encargara de regresar un objeto tipo `ProcesoCompartirDocumento` (ver Figura 7.8), lo siguiente es invocar al método `crearDocumento(String nombre, String descripción)` del objeto `ProcesoCompartirDocumento`, después se manda a ejecutar el método `crearEquipo()` el cual requiere como parámetros un nombre de equipo y también se necesita una lista de objetos tipo `Participantes`, esta objeto es `Participante` sólo encapsula datos referentes a una dirección de sitio y a un nombre de colaborador (estos datos son llenados con ayuda de *PrimeFaces*), lo siguiente que se debe de hacer es invocar al método `enviarSolicitudes()`, este método se encarga de hacer llegar las invitaciones a cada uno de los colaboradores.

Desde el punto de vista del desarrollador, este no tuvo que hacer otra cosa más que crear al objeto encargado de todo el proceso de compartición del documento, luego debe de invocar los métodos necesarios para darle nombre y descripción al documento, para posteriormente crear un grupo especificando los datos de los invitados y mandar a enviar las solicitudes de colaboración.



The screenshot shows a web browser window with the URL `http://localhost:8080/TesisOrtigosaApp/faces/login.xhtml`. The page title is "Tesis Ortigosa" and the logo is "Cinvestav". The user is logged in as "luis" with IP "192.168.100.105_2" and a "Salir" button. The main content area is titled "Documento Compartido" and contains three sections: "Definir documento" with fields for "Nombre documento:" and "Descripción documento:"; "Definir equipo" with a field for "Nombre equipo:"; and "Enviar invitaciones" with fields for "Id colaborador:", "Nombre colaborador:", and "Dirección colaborador:". A footer at the bottom reads "Ortigosa Flores José Luis - ortigosa08@gmail.com".

Figura 7.7: Página para crear un nuevo documento compartido.

```

@Named
@ViewScoped
public class ProcesoNuevoBean implements Serializable {

    @EJB
    private Controlador controlador;
    private Colaborador user;
    private ProcesoCompartirDocumento pcd;

    private String documentoNombre;
    private String documentoDescripcion;
    private String equipoNombre;

    private List<Participante> invitaciones;

    public String crearEquipo() {
        pcd = controlador.crearProcesoCompartirDocumento(user);
        pcd.crearDocumento(documentoNombre, documentoDescripcion);
        pcd.crearEquipo(equipoNombre, invitaciones);
        pcd.enviarSolicitudes();
        return "procesosActivos";
    }
}

```

Figura 7.8: Fragmento de código para el *Managed Bean* encargado de formar crear un equipo y de enviar invitaciones.

The screenshot shows the 'Tesis Ortigosa' web application. At the top right, there is a user profile section with the text 'Usuario' and '192.168.100.105_1 jose' and a 'Salir' button. The main content area is titled 'Procesos Activos' and contains a table with the following data:

Procesos activos					
Proceso ID	Documento	Documento descripción	Equipo	Estado actual	Estatus
192.168.100.105_3	documento	nuevo documento	equipo 0	3	EN_PROCESO

Below the table, there is a 'Continuar proceso' button. Underneath, there is a section titled 'Estatus solicitudes' with a table that currently shows 'No records found.' The footer of the page reads 'Ortigosa Flores José Luis - ortigosa08@gmail.com'.

Figura 7.9: Página que mostrar a los procesos activos de un colaborador. Cada proceso representa a un documento en algún estado del proceso de compartición y edición de documentos.

Página de procesos activos

Cuando las solicitudes han sido enviadas a los colaboradores el sistema almacena el estado del proceso en un objeto tipo `EstadoProceso` cada colaborador tiene asociada una lista con objetos de este tipo, para obtener la lista de los procesos activos de un colaborador se usa el método `getProcesos()`, luego con ayuda de *PrimeFaces* se muestra la lista al usuario (ver Figura 7.9).

Para continuar con un proceso simplemente se manda a invocar a al método `crearProcesoCompartirDocumento()` de la clase `Controlador` pero esta vez se le pasa el objeto `ProcesoEstado` correspondiente al proceso que se desea continuar, el método regresa un objeto de tipo `ProcesoCompartirDocumento`, el cual carga todos los datos necesarios para seguir el proceso donde sea que se haya quedado.

The screenshot shows a web browser window with the URL `http://localhost:8080/TesisOrtigosaApp/faces/login.xhtml`. The page title is "Tesis Ortigosa" and the user is logged in as "Luis". The main content area is titled "Solicitudes de Colaboración Recibidas" and contains a table with the following data:

Solicitudes recibidas					
Id solicitud	Colaborador solicitante id	Colaborador solicitante	Nombre del documento	Descripción documento	Fecha creación
192.168.100.105_6	192.168.100.105_1	jose@192.168.100.10	documento	nuevo documento	Mon Sep 25 05:18:57 CDT 2017

Below the table, there is a button labeled "Aceptar solicitud seleccionada". The footer of the page reads "Ortigosa Flores José Luis - ortigosa08@gmail.com".

Figura 7.10: Página bandeja de solicitudes de colaboración.

7.2.4. Recepción de solicitudes de colaboración

En la Subsección 7.2.3 se terminó diciendo que las solicitudes fueron enviadas a los colaboradores. Ahora aquí veremos qué pasa cuando llegan las solicitudes. Internamente las solicitudes son enviadas a un servicio web llamado `ws/registrar/solicitud-colaboracion`, este servicio web es el encargado de checar quien es el destinatario y de asociarle una nueva solicitud. Lo importante es que el desarrollador no tiene que preocuparse por ninguno de estos detalles, simplemente el desarrollador invoca el método `getBandejaSolicitudes()` de una instancia de `Colaborador`, esto le regresara al desarrollador obtener una lista con objetos de tipo `SolicitudesColaboración`, esta lista contiene las solicitudes que han sido recibidas hasta el momento en el que se invocó el método. La lista le es útil al desarrollador para crear una

pantalla con una lista de solicitudes de colaboración tal como se muestra en la Figura 7.10.

Para que una solicitud sea aceptada el desarrollador sólo necesita invocar al método `aceptarSolicitudColaboración(SolicitudColaboración sc)` de una instancia de la clase `Controlador`, lo que significa que el desarrollador no tendría mayor complicación que esa ya que el método se encargara de todo lo necesario para notificarle al remitente de la solicitud que si participara. Uno de las cosas que se realiza al invocar al método es enviar el *identificador global* de la solicitud más un booleano indicado la respuesta del destinatario, estos datos son enviados a un servicio web llamado `ws/solicitud-colaboración/respuesta`.



Figura 7.11: Página para definir las áreas de trabajo así como la definición de roles para los colaboradores.

7.2.5. Definición de áreas de trabajo

Otra de las páginas creadas para el proceso de crear un documento compartido es la que se usa para definir áreas de trabajo la cual se muestra en la Figura 7.11, para llegar a esta pantalla se tuvo que haber pasado por el proceso que se describe a continuación.

Una vez que las solicitudes fueron aceptadas por colaboradores esto se le notificara al colaborador que inicio el proceso de creación del documento compartido. Algunas solicitudes podrían haber sido aceptadas y algunas otras no, entonces será tarea del que inicio el proceso decidir si hay suficientes colaboradores para continuar la edición del documento, si este no es el caso entonces puede continuar esperando a que hayan suficientes solicitudes aceptadas, en el caso de que si haya suficientes solicitudes aceptadas puede seguir con el proceso y proceder a definir las áreas de trabajo.

Del lado del desarrollador para indicar que son suficientes solicitudes como para continuar primero debe instanciar el objeto de la clase `ProcesoCompartirDocumento` pasándole un objeto tipo `EstadoProceso`, una vez instanciado el objeto `ProcesoCompartirDocumento` se puede invocar a su método `solicitudesRecibidas()` lo que automáticamente hace que se mueva al estado *Definir áreas de trabajo*.

```

@Named
@ViewScoped
public class DefinirAreasBean implements Serializable {

    @EJB
    private Controlador controlador;
    private ProcesoCompartirDocumento pcd;
    private String fragmentoSelected;
    private String colaboradorSelected;
    private String rolSelected;
    private Fragmento fragmento;

    public void agregarFragmento() {
        pcd.agregarAreaTrabajo();
    }

    public void agregarCoautor() {
        pcd.agregarAutorRol(fragmentoSelected, colaboradorSelected, rolSelected)
    }

    public String enviarDocumentos() {
        pcd.areasListas();
        pcd.enviarDocumento();
        return "procesosActivos";
    }
}

```

Figura 7.12: Fragmento de código para el *Managed Bean* encargado de definir áreas de trabajo (fragmentos).

Es en este estado que el desarrollador tiene que invocar a los métodos `agregarAreaTrabajo()` y `agregarAutorRol()` para definir fragmentos y agregarles colaboradores y roles a cada fragmento. En el *Managed Bean* de la Figura 7.12 se muestra lo que el desarrollador tiene que escribir de código. Se debe mencionar que estos métodos se pueden llamar todas las veces que sean necesarias.

Una vez que se ha terminado el proceso de definición de áreas de trabajo el usuario debe indicar que ya ha terminado y es justo en este momento que se debe proceder a enviar el documento a todos los colaboradores. Por lo tanto el desarrollador debe hacer que se invoque el método `areasListas()` para posteriormente mandar a llamar el método `enviarDocumento()` ambos del objeto `ProcesoCompartirDocumento`. El último método será el encargado de en-

viar tanto las definiciones de autor de cada uno de los participantes en la edición como de la colección con los fragmentos.

```

public void agregarRecurso() {
    coleccion = controlador.agregarRecurso(coleccion, recurso, recursoNombre);
}

public void subirArchivo() {
    byte[] buff = new byte[2048];
    try {
        InputStream in = file.getInputStream();
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        while (in.read(buff) != -1) {
            out.write(buff);
        }
        recurso = controlador.agregarArchivo(file.getFileName(), out.toByteArray());
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
}

```

Figura 7.13: Fragmento de código para agregar un recurso a una colección dada.

The screenshot shows a web browser window with the URL `http://localhost:8080/TesisOrtigosaApp/faces/login.xhtml`. The page title is "Tesis Ortigosa" and the user is logged in as "jose" with IP "192.168.100.105_1". The main content area is titled "Registrar Recurso en Colección". It contains a table for "Colecciones" with the following data:

Id	Nombre	Equipo
192.168.100.105_4	documento	equipo 0

Below the table is a section for "Recurso de la colección" with a "No records found." message. There is a form with the following fields and buttons:

- Archivo:
- Nombre archivo:
- Id archivo:
- Nombre recurso:
-

The footer of the page reads: "Ortigosa Flores José Luis - ortigosa08@gmail.com".

Figura 7.14: Página para registrar recursos en colecciones.

7.2.6. Registrar recursos

El registro de recursos se hace por medio del método `agregarRecurso(Colección colección, Recurso recurso, String recursoNombre)` el cual se invoca a través una instancia de la clase `Controlador`, esto quiere decir que para agregar recursos un desarrollador sólo debe de invocar a dicho método. El *framework* ofrece una clase llamada `Archivo` para manejar cualquier tipo de archivo, esta clase hereda de `Recurso` por lo que es posible usarlo con el método `agregarRecurso()` (ver Figura 7.13).

En el caso de la aplicación de prueba se usó este método y clase para subir archivos, con el fin de apreciar el funcionamiento del *framework* al distribuirlo de manera automática a todos los miembros que pertenecen al equipo que colabora en el documento compartido. El método fue usado en conjunto con la página web que se muestra en la Figura 7.14, cuya función es la de seleccionar una colección y agregarle un archivo.

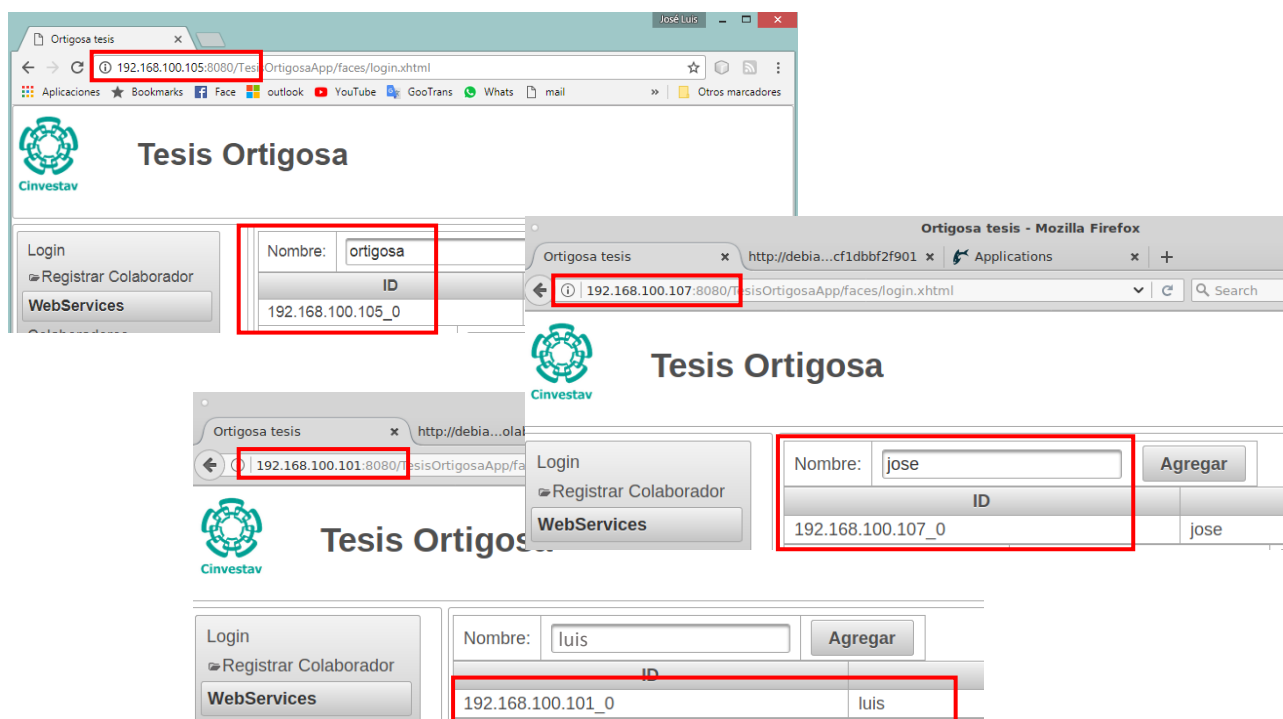


Figura 7.15: Registro de colaboradores en tres sitios de almacenamiento diferentes.

7.2.7. Procesos de creación de documento compartido en la aplicación de prueba

Cada una de las páginas mostradas en la sección anterior usa el *framework* desarrollado en esta tesis. Ahora, con el fin de probar su funcionamiento se procederá a registrar a tres colaboradores en tres sitios de almacenamiento (computadoras) diferentes.

En el sitio de almacenamiento uno (con IP: 192.168.100.105) se registrara al colaborador llamado *ortigosa*, en el sitio de almacenamiento dos (con IP: 192.168.100.107) se registrara al colaborador *jose* y en el sitio de almacenamiento tres (con IP: 192.168.100.101) se registrara al colaborador *Luis* (ver Figura 7.15).

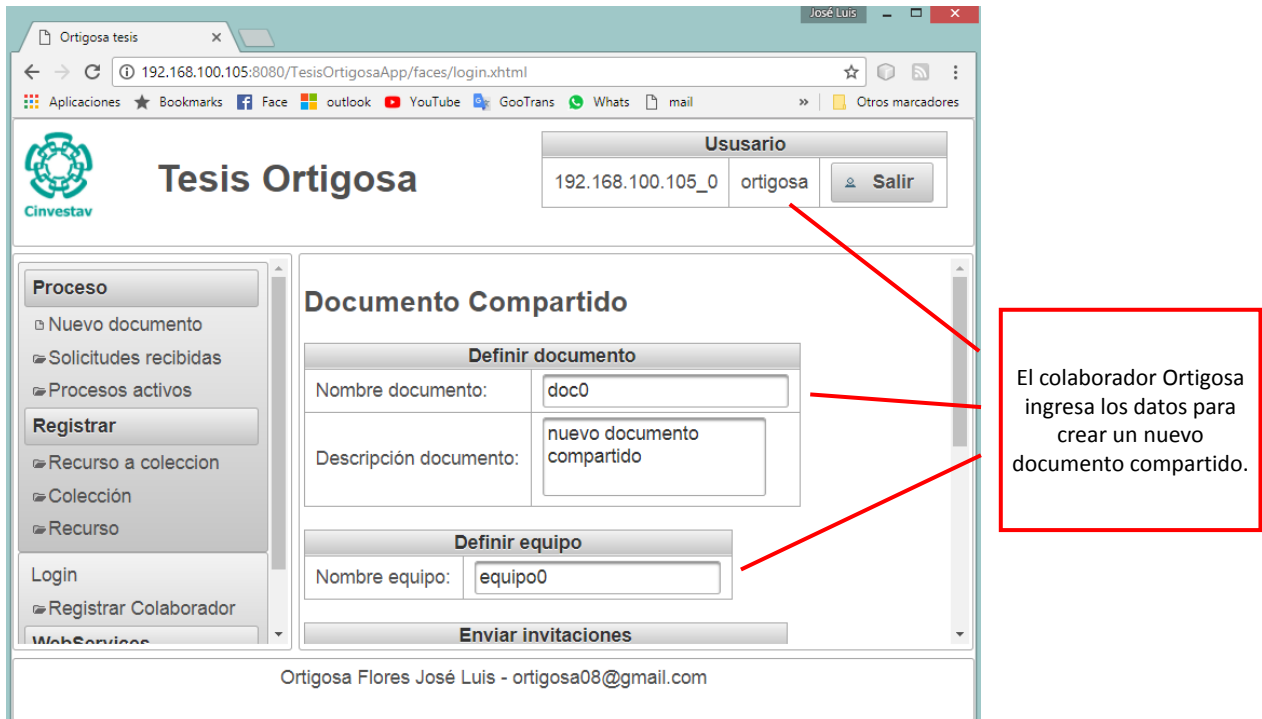


Figura 7.16: Colaborador creando un documento compartido.

Una vez que los colaboradores están registrados en sus respectivos sitios de almacenamiento, el colaborador llamado *ortigosa* decide crear un documento en el que quiere que colaboren los usuarios *jose* y *Luis*. Par lo que ingresa en el sistema y se dirige a crear un nuevo documento. El colaborador *ortigosa* crea un documento con el nombre *doc0* y le asocia un equipo llamado *equipo0* (ver Figura 7.16).

Después de que el colaborador dio nombre al documento, agregó una descripción y dio nombre al equipo de trabajo, entonces este procede a ingresar los datos de los colaboradores que participarán en el documento compartido, los datos requeridos son: el identificador del colaborador, su nombre y una dirección de sitios de almacenamiento (ver Figura 7.17).

Una vez que el colaborador *ortigosa* ha enviado las solicitudes de colaboración, a éste se le muestra que el proceso está activo en espera de que las solicitudes sean aceptadas, para lo que el sistema le muestra la página de procesos activos.

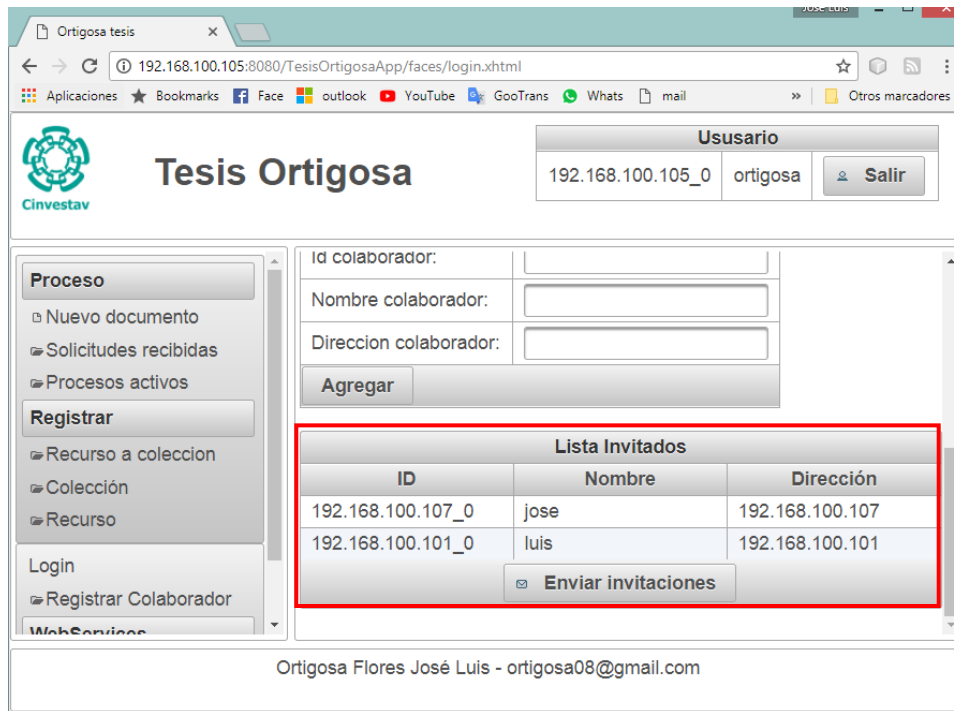


Figura 7.17: Registro de los participantes por parte del colaborador ortigosa.

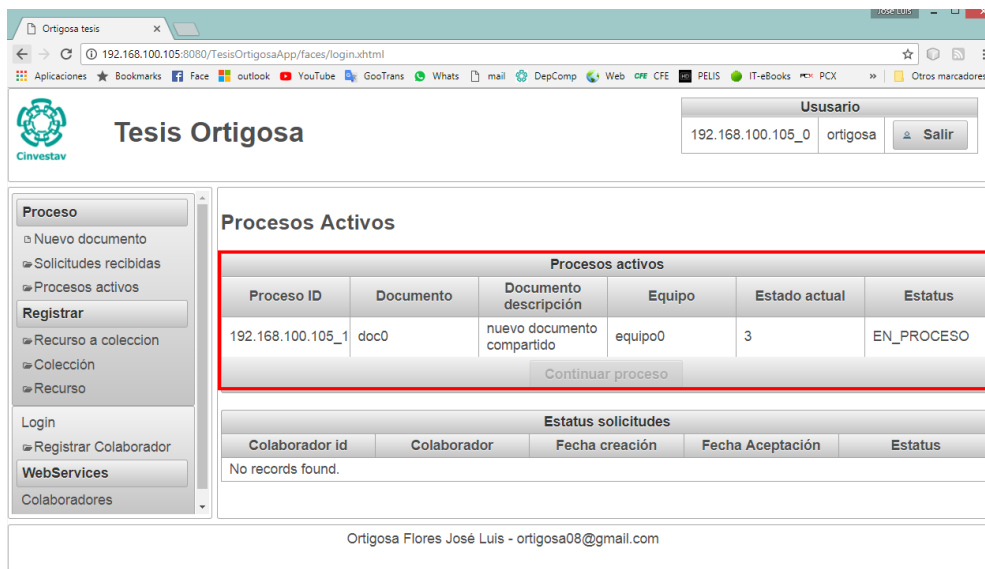
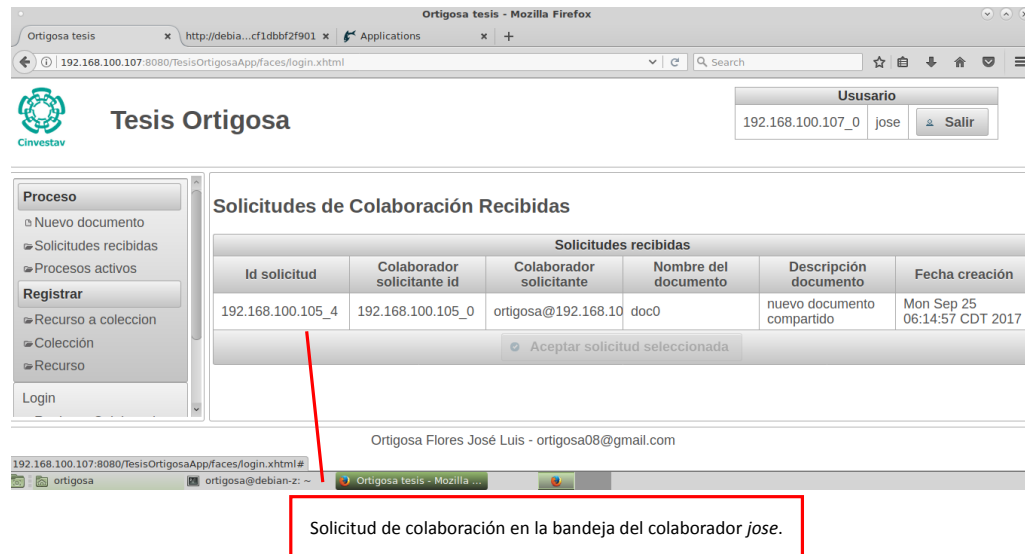


Figura 7.18: Figura que muestra a la página de procesos activos, en señal de que se está a la espera de que los participantes o invitados acepten las solicitudes enviadas.

Mientras tanto el colaborador *jose* ingresa al sistema en su computadora y se percata de que en su bandeja de solicitudes tiene un nuevo mensaje de parte de *ortigosa*. Esta solicitud de

colaboración la acepta y en ese momento el sistema le notifica a *ortigosa* que la solicitud fue aceptada (ver Figura 7.19). Esto mismo sucede con el colaborador *luis* (ver Figura 7.20).

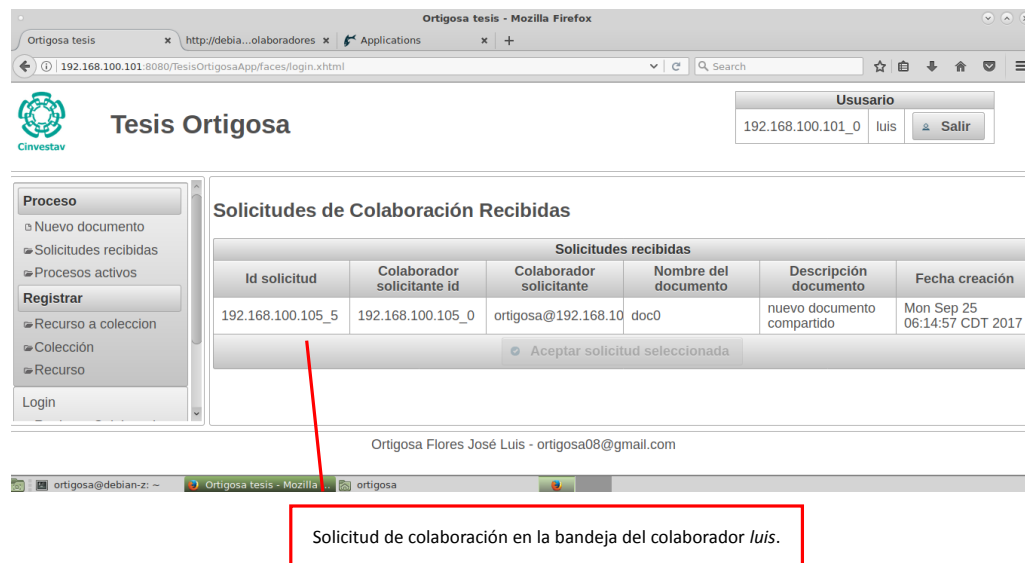


The screenshot shows the 'Tesis Ortigosa' interface. The user is logged in as 'jose' (ID: 192.168.100.107_0). The main content area is titled 'Solicitudes de Colaboración Recibidas' and contains a table with the following data:

Id solicitud	Colaborador solicitante id	Colaborador solicitante	Nombre del documento	Descripción documento	Fecha creación
192.168.100.105_4	192.168.100.105_0	ortigosa@192.168.10	doc0	nuevo documento compartido	Mon Sep 25 06:14:57 CDT 2017

Below the table, there is a button labeled 'Aceptar solicitud seleccionada'. A red line points from the 'Id solicitud' cell to a red-bordered text box at the bottom of the screenshot that reads: 'Solicitud de colaboración en la bandeja del colaborador jose.'

Figura 7.19: Página de bandeja de solicitudes de colaboración del colaborador *jose*, en la que aparece una solicitud de parte del colaborador *ortigosa*.



The screenshot shows the 'Tesis Ortigosa' interface. The user is logged in as 'luis' (ID: 192.168.100.101_0). The main content area is titled 'Solicitudes de Colaboración Recibidas' and contains a table with the following data:

Id solicitud	Colaborador solicitante id	Colaborador solicitante	Nombre del documento	Descripción documento	Fecha creación
192.168.100.105_5	192.168.100.105_0	ortigosa@192.168.10	doc0	nuevo documento compartido	Mon Sep 25 06:14:57 CDT 2017

Below the table, there is a button labeled 'Aceptar solicitud seleccionada'. A red line points from the 'Id solicitud' cell to a red-bordered text box at the bottom of the screenshot that reads: 'Solicitud de colaboración en la bandeja del colaborador luis.'

Figura 7.20: Pantalla de bandeja de solicitudes de colaboración del colaborador *luis*, en la que aparece una solicitud de parte del colaborador *ortigosa*.

Por otra parte, el colaborador *ortigosa* ingresa al sistema y revisa el estado de las solicitudes

enviadas a los colaboradores del documento *doc0*. Revisa que las solicitudes ya estén aceptadas (ve Figura 7.21).

The screenshot shows the 'Tesis Ortigosa' web application interface. The top navigation bar includes the Cinvestav logo and the title 'Tesis Ortigosa'. A user profile section shows the user 'ortigosa' with IP '192.168.100.105_0' and a 'Salir' button. The main content area is divided into two sections:

- Procesos activos:** A table with columns: Proceso ID, Documento, Documento descripción, Equipo, Estado actual, and Estatus. It shows one active process with ID '192.168.100.105_1', document 'doc0', description 'nuevo documento compartido', equipment 'equipo0', state '3', and status 'EN_PROCESO'. A 'Continuar proceso' button is located below the table.
- Estatus solicitudes:** A table with columns: Colaborador id, Colaborador, Fecha creación, Fecha Aceptación, and Estatus. It shows two accepted requests, both from 'Mon Sep 25 06:14:57 CDT 2017' and accepted on 'Mon Sep 25 06:18:41 CDT 2017' and 'Mon Sep 25 06:19:24 CDT 2017' respectively, with status 'true'. This table is highlighted with a red border in the original image.

The footer of the page displays the user's name 'Ortigosa Flores José Luis' and email 'ortigosa08@gmail.com'.

Figura 7.21: Pantalla que muestra el estado del proceso de compartición del documento *doc0*.

Una vez que *ortigosa* revisa que las solicitudes fueron aceptadas éste procede a definir fragmentos (áreas de trabajo) y a asociar colaboradores con roles a cada fragmento (área de trabajo) (ver Figura 7.22).

The screenshot shows the 'Definición de áreas de trabajo' section of the 'Tesis Ortigosa' web application. The top navigation bar and user profile are identical to Figure 7.21. The main content area is titled 'Definición de áreas de trabajo' and contains:

- Solicitudes:** A table with columns: id, Nombre, and solicitud aceptada. It shows two accepted requests with IDs '192.168.100.107_0' (Nombre: jose) and '192.168.100.101_0' (Nombre: luis), both with 'solicitud aceptada' set to 'true'.
- fragmentos:** A list of fragment IDs: '192.168.100.105_6' and '192.168.100.105_7'. An 'Agregar fragmento' button is located above this list.
- Colaboradores:** An 'Agregar colaborador' button is located at the bottom of the main content area.

The footer of the page displays the user's name 'Ortigosa Flores José Luis' and email 'ortigosa08@gmail.com'.

Figura 7.22: Pantalla para definir fragmentos (áreas de trabajo).

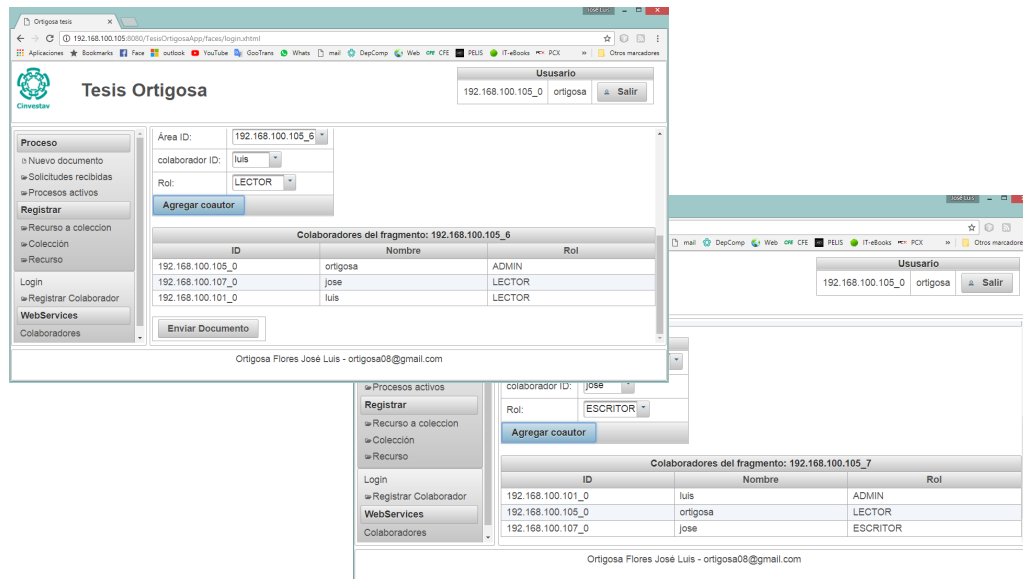


Figura 7.23: Colaborador *ortigosa* definiendo dos fragmentos (áreas de trabajo), a los cuales se les asignan colaboradores y roles.

Una vez que se definen los fragmentos (áreas de trabajo) el colaborador *ortigosa* procede a enviar los documentos a su equipo de trabajo (ver Figura 7.24).

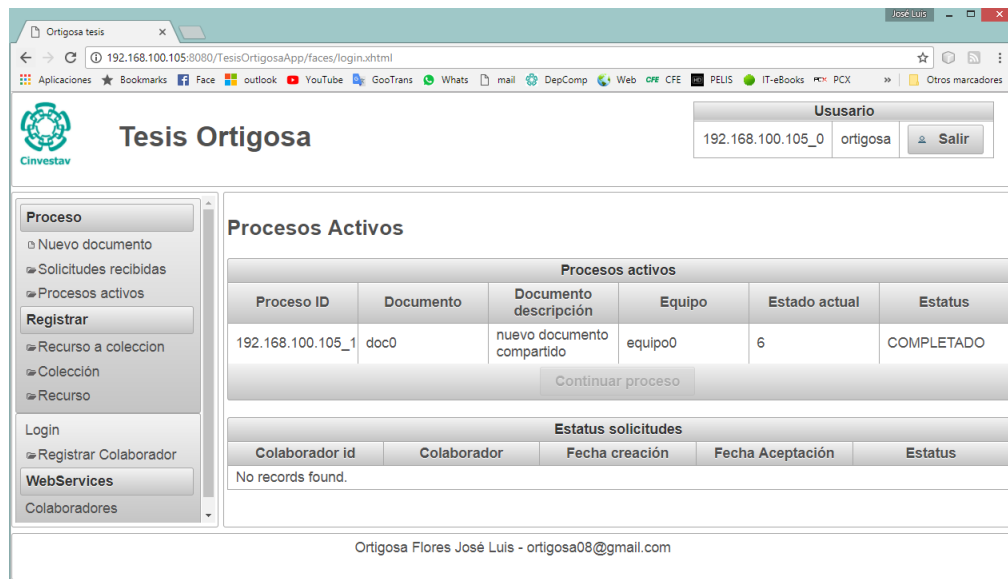


Figura 7.24: Proceso de compartición de documento ha terminado.

En este momento en todos los sitios de almacenamiento se encuentra la misma información, tanto el documento compartido como las definiciones de autor (ver Figura 7.25).

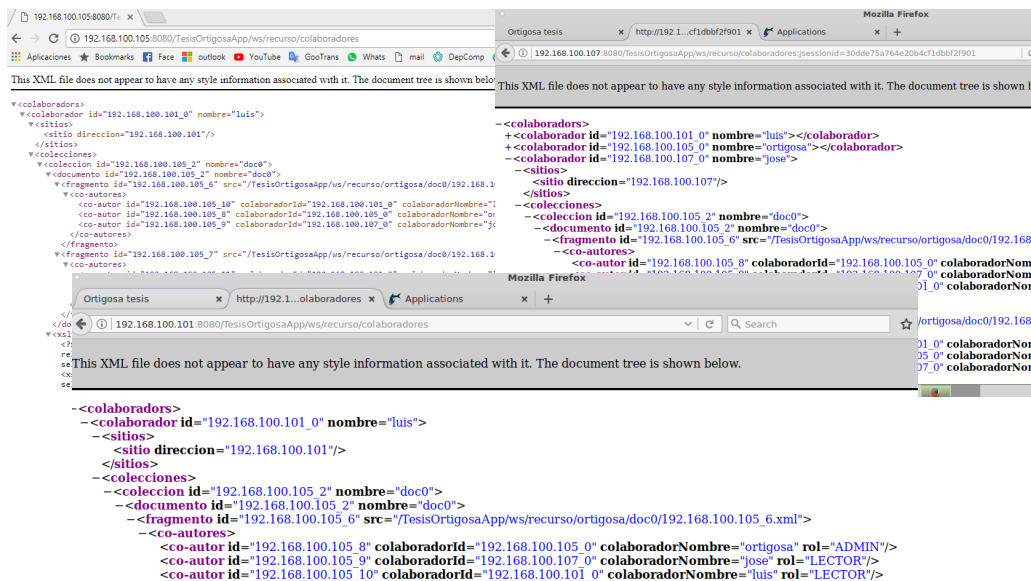


Figura 7.25: Sitios de almacenamiento cuentan con la misma información. Para lograrlo se hace uso del *framework* para solicitar todos los registros almacenados en cada sitio.

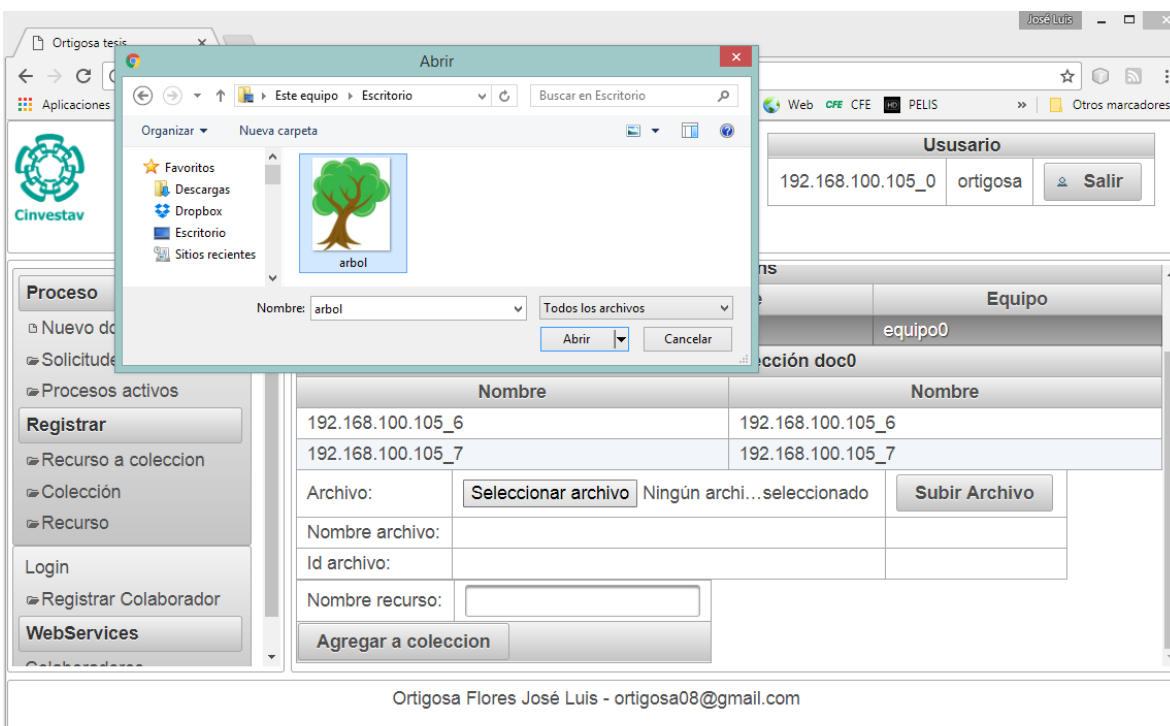


Figura 7.26: Pantalla para agregar un recurso a una colección, donde el colaborador *ortigosa* agrega una imagen a la colección de nombre *doc0*.



Figura 7.27: Pantalla que muestra al colaborador *ortigosa* registrando a la imagen *arbol.jpeg*.

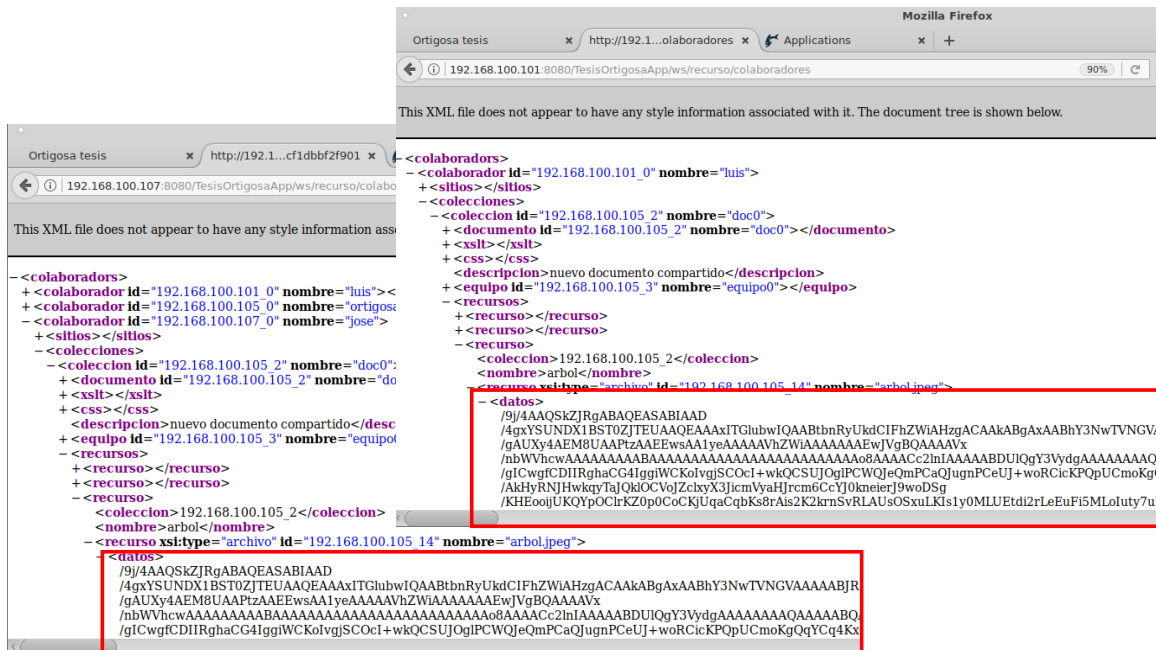


Figura 7.28: El recurso *arbol.jpeg* se encuentra en los sitios de los colaboradores.

7.3. Distribución de recursos

La aplicación web también usa la función encargada de distribuir un recurso que es agregado una colección por parte de uno de los integrantes que colaboran en dicha colección. En este ejemplo se propone agregar una imagen al documento creado en la sección anterior que lleva como nombre *doc0*. Para este ejemplo el colaborador *ortigosa* va a subir a la colección una imagen donde su binario será empaquetado en un objeto XML para ser enviado por HTTP (ver Figura 7.26).

En la Figura 7.27 se muestra que el recurso *árbol.jpeg* ha sido almacenado en la colección para verificar que este recurso ha sido enviado a los sitios de los colaboradores de la colección se checa el contenido que persiste en cada computadora (ver Figura 7.28). Aún cuando aparentemente este recurso parece estar varias veces, esto no es así. El recurso llamado *arbol.jpeg* sólo existe una vez (ver Figura 7.29).

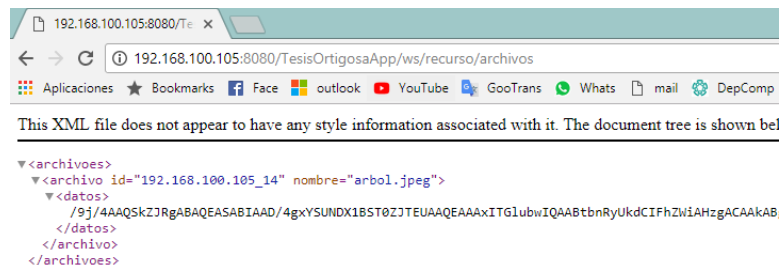


Figura 7.29: Servicio web que regresa a todo el conjunto de archivos almacenados en un sitio de almacenamiento.

The screenshot shows the 'Tesis Ortigosa' web application. The user is logged in as 'luis' (ID: 192.168.100.101_0). The interface includes a sidebar with navigation options like 'Nuevo documento', 'Registrar', and 'WebServices'. The main content area displays a table of collections:

ID	Nombre
192.168.100.101_0	luis
192.168.100.105_0	ortigosa
192.168.100.107_0	jose

Below the table, there is a search form with 'Id colaborador' set to '192.168.100.101_0' and 'luis', and a 'Buscar' button. A 'Agregar colección' button is visible. A second table shows collections for the user 'luis':

ID	Nombre
192.168.100.105_2	doc0
192.168.100.105_15	doc

A red box highlights the 'Agregar colección' button and the 'doc0' collection in the second table. A red arrow points from the text box to the 'Agregar colección' button.

El colaborador Luis tiene dos colecciones una llamada "doc0", la cual fue creada en el ejemplo anterior y "doc" en la que se agregar la imagen árbol.jpeg

Figura 7.30: Colecciones del colaborador *luis*.

7.4. Múltiples URLs apuntando a un recurso

Para este ejemplo el colaborador *luis* creará una colección (documento compartido) llamada *doc* (ver Figura 7.30) a la cual agregará el recurso *arbol.jpeg* pero él le pondrá un nombre distinto el cual será simplemente *tree* (ver Figura 7.31).

Ahora en este momento hay dos colecciones *doc0* y *doc* que contiene un mismo recurso el cual es *arbol.jpeg*, sin embargo, esas colecciones sólo almacenan una referencia a tal recurso, por un lado el colaborador *ortigosa* accede a tal recurso por medio de la URL:

`ortigosa/doc0/arbol`

y por el otro lado el colaborador *luis* accede al mismo recurso por medio de la URL siguiente:

`ortigosa/doc/tree`

Lo antes dicho se muestra en la Figura 7.32.

Si el colaborador *luis* decide eliminar el recurso *tree* (que apunta a *arbol.jpeg*) que está en la colección *doc*, tal operación no alteraría a la colección *doc0*, ya que lo único que se estaría eliminando es la referencia que se almacena en la colección *doc*.

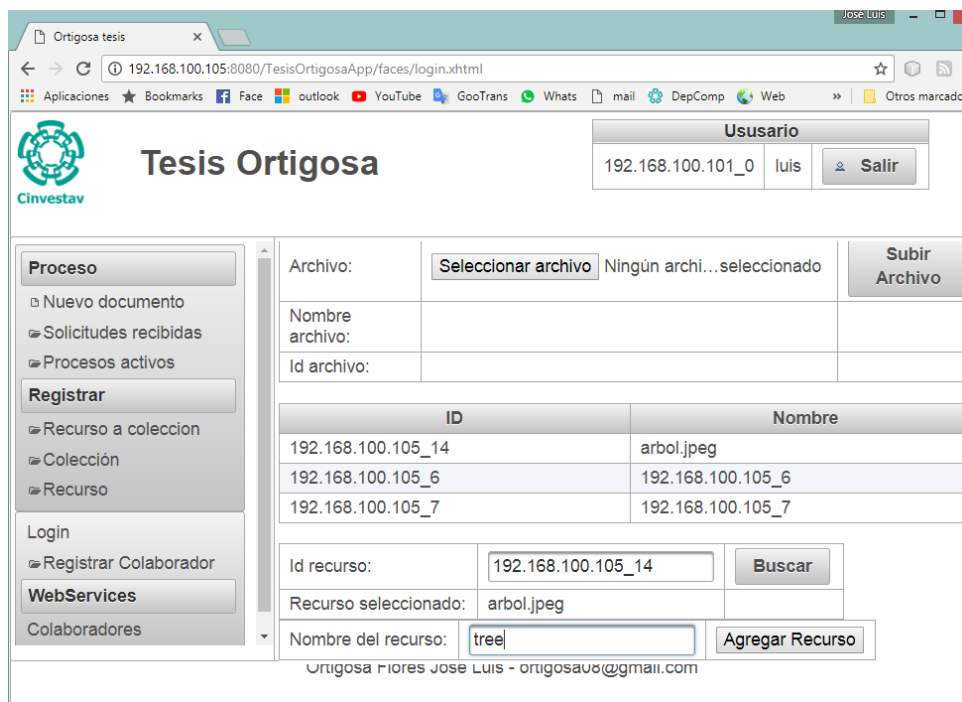


Figura 7.31: Pantalla que muestra a *luis* agregando el recurso llamado *arbol.jpeg* a otra colección *doc*.

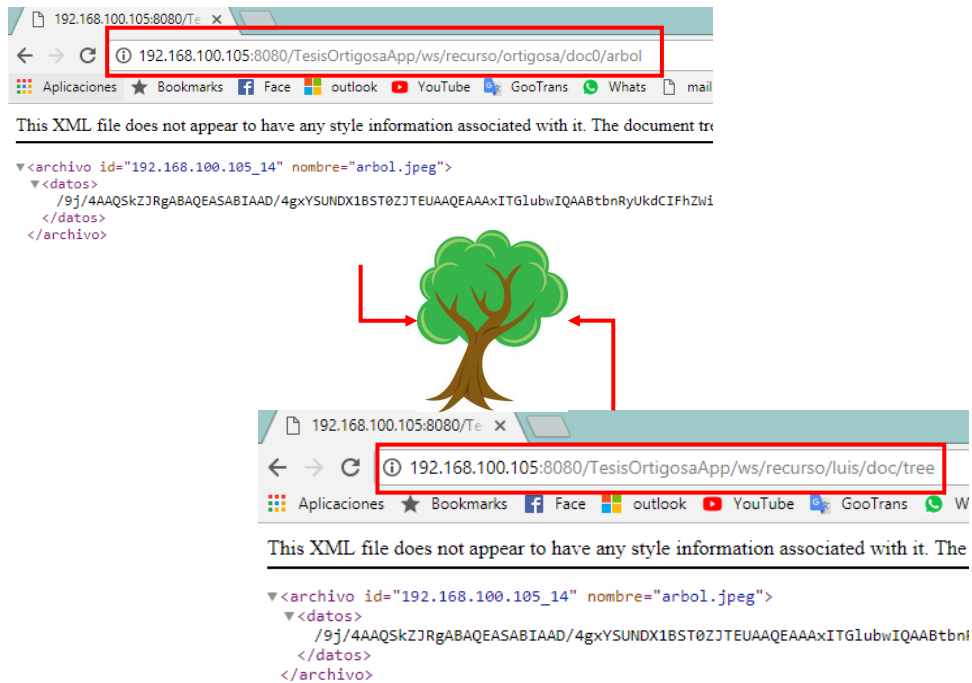


Figura 7.32: URLs apuntando al mismo contenido.

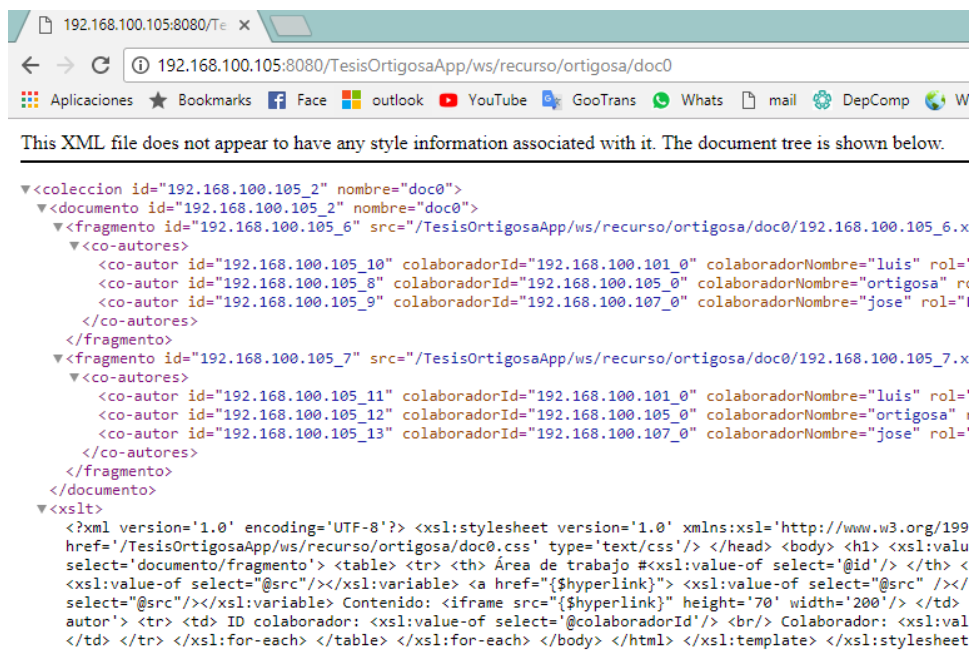


Figura 7.33: Contenido enviado por el sistema al ingresar a la URL ortigosa/doc0.

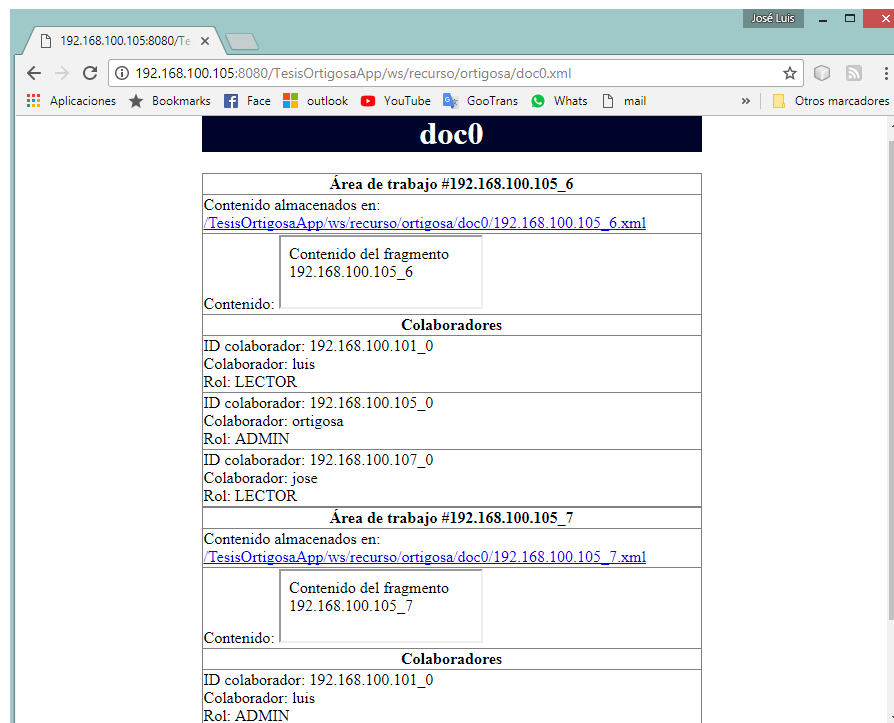


Figura 7.34: Visualización del archivo *doc0.xml* en el navegador.

7.5. URLs para acceder a los recursos

Cada una de las partes que conforman la base de un colaborador están asociadas a un URL respectivamente, lo que incluye a las colecciones y a sus recursos. Por ejemplo, en el caso de la colección *doc0* del colaborador *ortigosa* tiene asociada la siguiente URL:

`ortigosa/doc0`

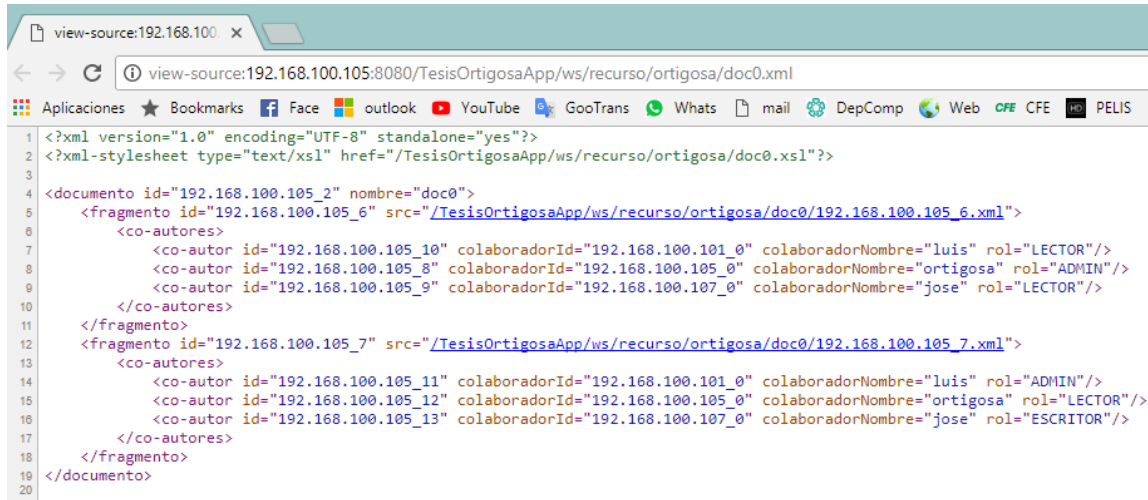
Esta URL es usada para enviar información, i.e., que si un sitio de almacenamiento tuviera la necesidad de obtener los datos de la colección *doc0*, entonces usaría esta URL para obtener la información (ver Figura 7.33). Si se quisiera acceder a la versión visible de la colección simplemente se accede a la siguiente dirección:

`ortigosa/doc0.xml`

Si se accede a tal URL desde un navegador, lo que se visualizaría es lo que se muestra en la Figura 7.32.

Aún cuando se podría pensar que el documento *ortigosa/doc0.xml* contiene toda la información necesaria para visualizar lo que se muestra en la Figura 7.34 esto no es así, la realidad

es que el archivo *ortigosa/doc0.xml* sólo contiene lo que se muestra en la Figura 7.35.



```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <?xml-stylesheet type="text/xsl" href="/TesisOrtigosaApp/ws/recurso/ortigosa/doc0.xsl"?>
3
4 <documento id="192.168.100.105_2" nombre="doc0">
5   <fragmento id="192.168.100.105_6" src="/TesisOrtigosaApp/ws/recurso/ortigosa/doc0/192.168.100.105_6.xml">
6     <co-autores>
7       <co-autor id="192.168.100.105_10" colaboradorId="192.168.100.101_0" colaboradorNombre="luis" rol="LECTOR"/>
8       <co-autor id="192.168.100.105_8" colaboradorId="192.168.100.105_0" colaboradorNombre="ortigosa" rol="ADMIN"/>
9       <co-autor id="192.168.100.105_9" colaboradorId="192.168.100.107_0" colaboradorNombre="jose" rol="LECTOR"/>
10    </co-autores>
11  </fragmento>
12  <fragmento id="192.168.100.105_7" src="/TesisOrtigosaApp/ws/recurso/ortigosa/doc0/192.168.100.105_7.xml">
13    <co-autores>
14      <co-autor id="192.168.100.105_11" colaboradorId="192.168.100.101_0" colaboradorNombre="luis" rol="ADMIN"/>
15      <co-autor id="192.168.100.105_12" colaboradorId="192.168.100.105_0" colaboradorNombre="ortigosa" rol="LECTOR"/>
16      <co-autor id="192.168.100.105_13" colaboradorId="192.168.100.107_0" colaboradorNombre="jose" rol="ESCRITOR"/>
17    </co-autores>
18  </fragmento>
19 </documento>
20

```

Figura 7.35: Contenido del documento *doc0.xml*.



```

<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <fragmento nombre="192.168.100.105_6" id="192.168.100.105_6">
  <xml><html> <body> Contenido del fragmento 192.168.100.105_6 </body> </html> </xml>
  <src>/TesisOrtigosaApp/ws/recurso/ortigosa/doc0/192.168.100.105_6.xml</src>
  - <colaboradores>
    - <autor-colaborativo id="192.168.100.105_10">
      <autor>192.168.100.101_0</autor>
      <rol>LECTOR</rol>
    </autor-colaborativo>
    - <autor-colaborativo id="192.168.100.105_8">
      <autor>192.168.100.105_0</autor>
      <rol>ADMIN</rol>
    </autor-colaborativo>
    - <autor-colaborativo id="192.168.100.105_9">
      <autor>192.168.100.107_0</autor>
      <rol>LECTOR</rol>
    </autor-colaborativo>
  </colaboradores>
</fragmento>

```

```

<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <fragmento nombre="192.168.100.105_7" id="192.168.100.105_7">
  <xml><html> <body> Contenido del fragmento 192.168.100.105_7 </body> </html> </xml>
  <src>/TesisOrtigosaApp/ws/recurso/ortigosa/doc0/192.168.100.105_7.xml</src>
  - <colaboradores>
    - <autor-colaborativo id="192.168.100.105_11">
      <autor>192.168.100.101_0</autor>
      <rol>ADMIN</rol>
    </autor-colaborativo>
    - <autor-colaborativo id="192.168.100.105_12">
      <autor>192.168.100.105_0</autor>
      <rol>LECTOR</rol>
    </autor-colaborativo>
    - <autor-colaborativo id="192.168.100.105_13">
      <autor>192.168.100.107_0</autor>
      <rol>ESCRITOR</rol>
    </autor-colaborativo>
  </colaboradores>
</fragmento>

```

Figura 7.36: Contenido del recurso *ortigosa/doc0/192.168.100.105_6* y *ortigosa/doc0/192.168.100.105_7*.

El que realmente le da formato al archivo *doc0.xml* es el archivo *doc0.xsl*, el cual a su vez está disponible en la dirección siguiente:

`ortigosa/doc0.xml`

De la Figura 7.35 se puede observar que lo único que se define en el *doc0.xml* es el nombre del documento, los fragmentos, además de los colaboradores y sus roles. El contenido de los fragmentos es almacenado en los archivos *192.168.100.105_6.xml* y *192.168.100.105_7.xml*, como se muestra en la Figura 7.34.

Aún cuando el formato se almacena en los archivos *ortigosa/doc0.xml* y *ortigosa/doc0.css*, no es necesario que ésta sea la única manera de interpretar los datos. Estos datos pueden ser interpretados. Por ejemplo, una aplicación JavaScript podría leer este archivo *ortigosa/doc0.xml* de alguna otra manera para mostrar una vista más dinámica e interactiva.

Si se accede a la dirección *ortigosa/doc0/192.168.100.105_6* en este caso el sistema regresara lo que se muestra en la Figura 7.36, que es lo que enviaría un sitio de almacenamiento cuando se le solicite que envíe un fragmento.

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se presentan las últimas observaciones y conclusiones del trabajo realizado (ver Sección 8.1) incluyendo el trabajo futuro 8.2.

8.1. Conclusiones

Las aplicaciones colaborativas desarrolladas para la Web implícitamente eliminan la heterogeneidad que existen a causa de los múltiples sistemas operativos y arquitecturas, gracias a que la Web actúa como un *middleware*¹. Al desarrollar aplicaciones colaborativas basadas en estándares Web se reduce la curva de aprendizaje y se facilita su mantenimiento.

La tecnología asociada a la Web no han sido desarrollados con el objetivo de satisfacer las necesidades de los sistemas colaborativos, sin embargo la generalidad de los estándares los dota de flexibilidad para adaptarlos a las necesidades de los sistemas colaborativos.

El problema de los hipervínculos rotos continua abierto a pesar de múltiples esfuerzos (*broken links*) entre los que se incluye: el agregar metadatos a la información (RDF [W3C et al., 2014]), sistemas de actualización basados en suscripciones como lo es RSS [Software, 2002], hasta soluciones más complejas relacionadas con el uso de inteligencia artificial [Martinez-Romo and Araujo, 2012], [Morishima et al., 2008], [Popitsch and Haslhofer, 2010] los cuales se enfocan en algunas causa del problema o en búsquedas no exhaustivas. Se busca una solución para convertir a la Web en una red de información o de conocimiento accesible para humanos y maquinas.

En esta tesis se enfrenta al problema de los hipervínculos rotos como obstáculo para aprovechar a las URLs para referenciar recursos. Para hacer frente al problema se usa la integridad de referencias proporcionada por las bases de datos en conjunto con los niveles de nombre propuestos por la plataforma PIÑAS.

¹capa de software que proporciona una interfaz común para el desarrollo de aplicaciones y que elimina las complicaciones asociadas a la comunicación de diferentes plataformas o arquitecturas

La parte que complementa la solución a los hipervínculos rotos es el uso de la replicación de datos, tarea que es muy demandante de ancho de banda más al estar usando HTTP como medio para transferir archivos XML. Sin embargo una ventaja de HTTP es que se evitan configuraciones en *firewalls* y *proxies*. La ventaja de usar XML es su tecnología XSLT para transformar archivos XML a HTML de manera transparente.

Puede parecer que el sistema está atado a vivir en el entorno de JavaEE, sin embargo este puede ser fácilmente adaptado para cambiar la manera en la que se comunica, gracias al uso de patrones de diseño, lo que permite cambiar la comunicación de RESTful a sockets o a RMI, o la manera de empaquetar la información, ya que se puede optar por mandar objetos serializados o JSON. Estos cambios eliminarían la ventaja que ofrece la Web, pero permitirían la creación de aplicaciones que tengan otras necesidades de comunicación.

El *framework* generado puede ser usado para desarrollar aplicaciones colaborativas web distribuidas, siempre teniendo en cuenta que sólo proporciona soporte para una interacción asíncrona, que el trabajo colaborativo debe contar con tareas independientes, y que no más de un colaborador puede editar un fragmento al mismo tiempo.

El *framework* crea una red de sitios de almacenamiento, que se comunican por medio de HTTP/XML, los cuales pueden dar soporte a múltiples usuarios quienes eventualmente se convertirán en colaboradores de documentos.

Para finalizar, se debe hacer hincapié en que el sistema desarrollado no proporciona apoyo completo para que una aplicación pueda considerarse colaborativa (*groupware*). La aplicación desarrollada sólo ofrece soporte y apoyo en los siguientes aspectos: proporciona un ambiente de trabajo distribuido, da soporte a la definición de fragmentos (áreas de trabajo) que son mutuamente excluyentes, incluye soporte para la distribución y acceso al contenido.

Según [Koch and Gross, 2006] para considerar una aplicación como colaborativa esta debe cumplir al menos los siguientes aspectos: Comunicación, Colaboración, Coordinación y Conciencia de colaboración. A este respecto el sistema desarrollado sólo da soporte parcial en el rubro de la colaboración y la coordinación, al brindar apoyo en la definición y gestión de áreas de trabajo, así como en apoyar la interacción asíncrona entre los colaboradores. El sistema creado no da soporte a la conciencia de grupo ni a la comunicación.

8.2. Trabajo futuro

El *framework* desarrollado cuenta con muchos puntos de oportunidad, entre los que se destacan los siguientes:

- La búsqueda de algoritmos eficientes dedicados a la replicación de recursos.

- Análisis de la API que ofrece el *framework* con el fin de reducir y optimizar.
- Análisis de la cantidad de datos enviados con respecto a los datos extra agregados por usar HTTP y XML.
- Análisis del consumo de ancho de banda.
- Probar con personas una aplicación colaborativa desarrollada con el *framework*.
- El sistema carece de un módulo encargado de la seguridad, una opción es integrar HTTPS en lugar de HTTP.

Bibliografía

- [Alkacon, 2016] Alkacon (2016). OpenCMS opencms documentation central. <http://documentation.opencms.org/central/>. [Online; accessed: 2017-05-04].
- [Buytaert, 2016] Buytaert, D. (2016). Drupal community documentation. <https://www.drupal.org/documentation>. [Online; accessed: 2017-05-05].
- [David and Borges, 2002] David, J. M. N. and Borges, M. R. (2002). Copse-web: An infrastructure for developing web-based groupware applications. In *Haake J.M., Pino J.A. (eds) Groupware: Design, Implementation, and Use. CRIWG 2002. Lecture Notes in Computer Science*.
- [Decouchant et al., 2008] Decouchant, D., Mendoza, S., and Rodríguez, J. (2008). A realistic and efficient distributed infrastructure for nomadic web cooperative work. In *2008 Mexican International Conference on Computer Science*, pages 151–162.
- [Gamma et al., 1995] Gamma, E., Vlissides, J., Johnson, R., and Helm, R. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley professional computing series, Canada.
- [Glass et al., 2001] Glass, B., Nielsen, J., and Fogg, B. (2001). Restoring broken links utilizing a spider process. patent number = US6253204 B1.
- [Group, 2017] Group, P. (2017). Php. <https://secure.php.net/>. [Online; accessed: 2017-09-10].
- [GroupKit, 2016] GroupKit (2016). Google. <http://www.groupkit.org/>. [Online; accessed: 2016-04-1].
- [Haslhofer and Popitsch, 2009] Haslhofer, B. and Popitsch, N. (2009). Dsnotify - detecting and fixing broken links in linked data sets. *2009 20th International Workshop on Database and Expert Systems Application, Database and Expert Systems Application, 2009. DEXA '09. 20th International Workshop on*, page 89.
- [INRIA and S.A., 1997] INRIA and S.A., G. (1997). Thot editor alpha 2.1e release. <http://src.gnu-darwin.org/ports/editors/thoteditor/work/Thot/doc/thoteditor/Compile.html>. [Online; accessed: 2017-05-10].

- [Koch and Gross, 2006] Koch, M. and Gross, T. (2006). Computer-supported cooperative work-concepts and trends. *Proc Conf of the Association Information And Management AIM*, pages 1–8.
- [Krakowiak, 2007] Krakowiak, S. (2007). Middleware architecture with patterns and frameworks.
- [Liliedahl, 2008] Liliedahl, D. (2008). *OpenCMS Development, extending and customizing OpenCMS through its Java API*. Packt Publishing, Birmingham.
- [Lopez and Skarmeta, 2003] Lopez, P. G. and Skarmeta, A. F. G. (2003). Ants framework for cooperative work environments. *Computer*, 36(3):56–62.
- [Marsic, 1999] Marsic, I. (1999). Disciple: A framework for multimodal collaboration in heterogeneous environments. *ACM Comput. Surv.*, 31(2es).
- [Martinez-Romo and Araujo, 2012] Martinez-Romo, J. and Araujo, L. (2012). Updating broken web links: An automatic recommendation system. *Information Processing and Management*, 48(2):183–203.
- [McKeever, 2003] McKeever, S. (2003). Understanding web content management systems: Evolution, lifecycle and market. *Industrial Management and Data Systems*, 103(9):686–692.
- [Microsoft, 2017] Microsoft (2017). Asp.net. <https://www.asp.net/>. [Online; accessed: 2017-09-10].
- [Morishima et al., 2008] Morishima, A., Nakamizo, A., Iida, T., Sugimoto, S., and Kitagawa, H. (2008). Pagechaser: A tool for the automatic correction of broken web links. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1486–1488.
- [Oracle, 2016a] Oracle (2016a). Mysql documentation. <https://dev.mysql.com/doc/>. [Online; accessed: 2017-05-01].
- [Oracle, 2016b] Oracle (2016b). Oracle help center. <https://docs.oracle.com/en/>. [Online; accessed: 2017-05-02].
- [Oracle, 2017] Oracle (2017). Java se at a glance. <http://www.oracle.com/technetwork/java/javase/overview/index.html>. [Online; accessed: 2017-09-10].
- [Popitsch and Haslhofer, 2011] Popitsch, N. and Haslhofer, B. (2011). Dsnotify – a solution for event detection and link maintenance in dynamic datasets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(Semantic Web Dynamics):266 – 283.
- [Popitsch and Haslhofer, 2010] Popitsch, N. P. and Haslhofer, B. (2010). Dsnotify: Handling broken links in the web of data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 761–770, New York, NY, USA. ACM.

- [Powel and Gill, 2003] Powel, W. and Gill, C. (2003). Web content management systems in higher education. *Educause Quarterly*, 26(2):43–50.
- [PrimeTek, 2017] PrimeTek (2017). Primefaces. <https://www.primefaces.org/>. [Online; accessed: 2017-09-16].
- [Rajabi et al., 2014] Rajabi, E., Sanchez-Alonso, S., and Sicilia, M.-A. (2014). Analyzing broken links on the web of data: An experiment with dbpedia. *Journal of the Association for Information Science and Technology*, 65(8):1721–1727.
- [Riehle, 2000] Riehle, D. (2000). *Framework Design: A Role Modeling Approach*. PhD thesis, Swiss Federal Institute Of Technology Zurich.
- [Salcedo and Decouchant, 1997] Salcedo, M. R. and Decouchant, D. (1997). Structured cooperative authoring for the world wide web. *Computer Supported Cooperative Work (CSCW)*, 6(2–3):157–274.
- [Sandoval, 2009] Sandoval, J. (2009). *RESTful Java Web Services*. Packt Publishing, Birmingham.
- [Software, 2002] Software, U. (2002). Feed validator, rss 2.0 specification. <https://validator.w3.org/feed/docs/rss2.html>. [Online; accessed: 2017-01-11].
- [Tomlinson, 2010] Tomlinson, T. (2010). *Beginning Drupal 7*. Apress, United States of America.
- [W3C et al., 2014] W3C, MIT, ERCIM, and Keio (2014). Resource description framework (rdf). <https://www.w3.org/2001/sw/wiki/RDF>. [Online; accessed: 2017-05-22].