**Universidad de la Ciencias Informáticas**

**Facultad 10**


# “Repositorio de Contenidos para el Diccionario Enciclopédico Libertad”.

**Trabajo de Diploma para optar por el título de**

**Ingeniero en Ciencias Informáticas**

**Autor:** Leticia García Rodríguez

**Tutor:** Ing Reinier Elejalde Chacón

**Co-tutor:** Ing Evelio Maykel Medina Manrique

**Ciudad de La Habana, 2010**

**“Año 51 de la Revolución”**

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas; para que hagan el uso que estimen pertinente con el mismo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Leticia García Rodríguez

_____

Firma del Autor

Reinier Elejalde Chacon

_____

Firma del Tutor

*“Para el logro del triunfo siempre ha sido indispensable pasar por la senda de los sacrificios.”*

*Simón Bolívar*

## *Agradecimientos*

*Hoy después de muchos años de estudio y esfuerzo se hace realidad mi sueño, el de mis padres y en general el de mi familia. Para hacer posible el mismo muchas personas me ayudaron, apoyaron y depositaron su confianza en mí, hoy tengo la oportunidad de hacerles saber cuan agradecida estoy de que hayan formado parte de este sueño.*

*En primer lugar quiero agradecer a la persona que hizo posible que yo estuviese aquí, por su infinito apoyo a lo largo de mis años de pre y sobre todos en los de universidad, a mi tío Serafín, porque más que tío es un padre de corazón, al cual nunca le podré estar lo suficientemente agradecida por todo lo que ha hecho por mí.*

*A mi Madre por su infinito espíritu de sacrificio y entrega para conmigo, desde el día que dijo que yo sería solo para ella, por tu apoyo todos estos años, por tu infinito amor, compresión y por ayudarme a que este momento llegara, por creer siempre en que yo podría y sobre todo por dejar que yo persiguiera mis sueños y seguirme en ellos, nunca sabrás cuanto te admiro y nunca podré encontrar los hechos que demuestren cuanto te amo.*

*También quiero agradecerles a mis tías, en especial a Nelda, gracias por escucharme, mimarme y malcriarme, y sobre todo por quererme como si fuera también tu hija y hacerme sentir como tal. A mis queridísimas tías Julia y Dignora no se pongan celosas, no saben cuánto les agradezco su apoyo estos cinco años, gracias por hacerme mucho mejor la vida todo este tiempo y por todos sus consejos, muchísimas gracias, por ser mi madres más cercanas este tiempo.*

*A mi papá, gracias por tu inmenso cariño, creer ciegamente en mi y ser lo mejor que has podido conmigo, como con nadie más.*

*A mis primas y primos, en especial mi hermana de corazón Liliana, a Maylin, Yaniel, Yaneicita y a los mellizos.*

*Quiero agradecer de manera especial a mi tutor, que a lo largo de mi carrera ha sido compañero, amigo, confidente y hoy mi guía en la realización de este trabajo, sin el cual estoy segura no hubiese podido realizar. Nunca podré pagarte todo lo que has hecho por mí en todo este tiempo en mi y en mi familia*

*tienes otra familia para ti. También agradecer a mi co-tutor Mikel Evelio que ha sido un gran amigo todos estos años.*

*NO quiero dejar de agradecerles a amigos como Denis y Alain por sus constantes consejos y regaños cuando hicieron falta.*

*Gracias a mis padrinos Juancito y Caridad, que me han hecho más fuerte y segura para acabar mis estudios con éxito y continuar de igual forma en mis próximas etapas de la vida.*

*Agradecer también a mis amigas que hoy ya no están aquí pero su apoyo ha sido fundamental para mí, a mi hermanita Aimé muchas gracias por tantos secretos compartidos y permitirme ser tu madrina, a Yairelis e Iliana y a los muchachos de mi antiguo grupo. Gracias también a mis compañeros de mi actual grupo, ahí también he encontrado personas muy especiales como Aylen, Liana que ya hoy tampoco está aquí, entre otros.*

*Por último pero no menos importante quiero agradecerle a mi novio, el cual ha fuerza de regaños y mucho amor me ayudo a salir adelante en los momentos en que lo veía todo oscuro. Gracias por ser tan exigente conmigo cuando yo no lo fui, por contar hasta diez cuando tuve un problema y la cogí contigo y aun cuando la tesis no me daba tiempo de tener detalles contigo no dudar que te quiero con el alma. Espero poder contar contigo siempre a mi lado.*

*Leticia*

## *Dedicatoria*

*Le dedico esta tesis en primer lugar a mis padres, por su apoyo incondicional e infinito amor.*

*A mi familia de manera general por ser tan especiales conmigo a mis tías y primos.*

*Y le dedico mus especialmente este trabajo a dos personas que son muy importante en mi vida, que aunque hoy no están junto conmigo físicamente siempre están mi corazón, porque hicieron de mi su hija más chiquita y con su amor llenaron el lugar que otros no ocuparon. A mi abuela María y mi abuelo Selestino donde quiera que se encuentren este trabajo es para ustedes.*

## Resumen

El presente trabajo propone la implementación de un repositorio de contenidos para almacenar y centralizar la información que posteriormente contendrá el Diccionario Enciclopédico Libertad. Se estudiaron para ello las tendencias actuales y las tecnologías que posibilitarían realizar una aplicación lo más amigable posible para todo el personal que interactúe con ella. Se modela el proceso que debe seguir el desarrollo del repositorio y se definen los actores, así como las principales funciones con las que se contará para gestionar los contenidos y los usuarios. Se explican las funcionalidades que se añaden a través de estándares como la JRS-170 y Jackrabbit. Esta propuesta es una contribución al propósito cubano de proteger los recursos digitales, tal y como se ha podido hacer con los materiales en soportes tradicionales. La preservación de los recursos digitales es un reto al que sin duda tendrá que enfrentarse toda institución que aspire a preservar para futuros usuarios lo que se genera hoy en formato digital.



Palabras claves: repositorio de contenido, JRS-170, Jackrabbit.

# Índice

# Introducción

En la actualidad las nuevas tecnologías de la Información y las Comunicaciones (TIC) están actuando vertiginosamente sobre la sociedad, motivando y acelerando los procesos de cambio, creando nuevos espacios en las estructuras educacionales y laborales. Estas tecnologías están cambiando radicalmente las formas de trabajo, los medios a través de los cuales las personas acceden al conocimiento, se comunican y aprenden, la educación en todos los niveles de edad y profesión.

Dichas tecnologías están produciendo importantes transformaciones en la sociedad cubana, no solo a escala nacional sino a escala mundial. Éstas marcan la característica fundamental que distingue el momento histórico actual, y el que viviremos con los años, de tal manera que hoy se habla de una "Sociedad de la Información y del Conocimiento". Esta nueva sociedad se caracteriza por un predominio de la gestión de la información y su digitalización, protagonizar un cambio en las relaciones laborales, económicas, culturales y la forma de pensar de los individuos.

El Estado Cubano, ha identificado la conveniencia y necesidad de dominar e introducir en la práctica social las Tecnologías de la Información y las Comunicaciones y lograr una cultura digital como una de las características imprescindibles del hombre nuevo, lo que facilitaría a la sociedad acercarse más hacia el objetivo de un desarrollo sostenible. Es por ello que, desde finales del pasado siglo se implementó en Cuba la "Política Nacional de Informatización de la Sociedad", la cual establece las acciones fundamentales para la construcción en Cuba de la sociedad de la información y el conocimiento.

La recopilación de la información en medios poco legibles, ya obsoletos, explican por sí solos la necesidad que tiene el país de esos conocimientos sociales y políticos debido a su cultura, que a través de los medios informáticos y con el auge de la sociedad del conocimiento posibilitan la recopilación de información según pasan los hechos; la enseñanza de la Historia desempeña un papel fundamental en la estructuración del sistema de conocimiento sobre la realidad más actual, al permitir el enfoque científico

de los problemas globales de la contemporaneidad. De ahí la necesidad de crear formas de conservar lo que pueda ser conservado.

Un diccionario enciclopédico es una obra de consulta que agrega a los diccionarios de la lengua, información sobre Historia, Política, Ciencias, Arte y Literatura, Geografía, Deporte entre otras áreas del conocimiento, y personajes destacados dentro de ellas. Generalmente las entradas se ordenan alfabéticamente y para su compilación y edición participa gran número de especialistas, encargados de recopilar toda la información especializada del momento y presentarla de manera que resulte comprensible para el lector profano y aceptable para el especialista. Normalmente se presenta en varios volúmenes impresos y/o digital (María, 2009).

En el año 2000 surge la idea de crear un diccionario enciclopédico cubano a propósito de la adquisición de libros para el Programa Editorial Libertad donde aparecieron en algunos diccionarios y enciclopedias extranjeras, conceptualizaciones tergiversadas de la Revolución Cubana, la figura de Ernesto Guevara y Fidel Castro desde la visión hegemónica del primer mundo. Esto convirtió en uno de los Programas de la Batalla de Ideas el desarrollo del Proyecto del Diccionario Enciclopédico con el nombre Libertad, cuyo objetivo era tener un lugar en el cual exponer conceptos y términos desde los fundamentos que sustentan la Revolución Cubana.

Para darle cumplimiento a esta tarea se dividió el proyecto en varias etapas. En una primera etapa se decidió recolectar la información que contendría el Diccionario Enciclopédico, para lo cual se desarrolló una aplicación que permitiría crear los formularios de recogida de información (plantillas digitales) de acuerdo a las categorías que fuesen necesarias crear. Más tarde se creó otra aplicación que permitía llenar las plantillas digitales que recogen en sí la información que contendría el Diccionario Enciclopédico, a lo cual llamaremos activos digitales.

Una vez concluida esta fase, es momento ideal para pensar en los datos recolectados, ya sean las plantillas o los activos digitales, dónde y cómo serán guardados, ya que es una vieja aspiración de toda institución que genera un alto volumen de contenidos tenerlos documentados y gestionarlos con visión de

futuro. Por tal motivo se hace necesario el uso de algún servicio de información para compartir, divulgar, acceder, conocer, representar la información del Diccionario Enciclopédico.

El principal objetivo que propone este trabajo es encontrar el establecimiento de un criterio de medida referido al desarrollo de algunos servicios de información para compartir documentos, información importante, imágenes y presentaciones, y eso se traduce en que se necesita una interfaz de entrada a un servicio por medio de la cual sus usuarios puedan suscribirse y luego añadir contenidos y que, por supuesto, por detrás de todo esto exista un sistema de almacenamiento y organización de esa información que garantice una eficiente recuperación de los contenidos guardados.

Para la organización de la información y para garantizar el contenido expuesto, existen los repositorios de contenidos los cuales brindan la posibilidad de tener expuesto razonablemente los datos recopilados con anterioridad facilitando así que los materiales estén centrados en una misma ubicación. Un repositorio de contenido es un servidor o un conjunto de servicios usados para almacenar, buscar, acceder y controlar el contenido. El mismo provee este servicio a las aplicaciones especializadas en el manejo del contenido, tales como gestores documentales, administradores de contenido Web, sistemas de almacenamiento y recuperación de imágenes u otras aplicaciones. Además provee servicios sobre el contenido como el almacenamiento, la clasificación, la seguridad, el control y las consultas a dichas aplicaciones. Lo que distingue la gestión de contenido de otras aplicaciones de bases de datos es el nivel de control ejercido sobre objetos o elementos de contenido individuales y la habilidad para buscar dichos contenidos. El acceso a estos servicios requiere encapsular las peticiones en paquetes de seguridad para prevenir el acceso no autorizado o los cambios sobre los contenidos.

En la actualidad el Diccionario Enciclopédico Libertad no cuenta con algún servicio fiable a través del cual se garantice el almacenamiento y centralización de la información, es decir una vez creadas las plantillas y activos digitales las mismas se almacenan en algún lugar físico en el disco duro de la computadora, lo cual incurre en un bajo nivel de aseguramiento en la recuperación y acceso a dicha información. Esta es la razón por la cual se hace imprescindible la búsqueda de un mecanismo para implementar un repositorio

de contenidos que almacene centralizadamente la información concerniente al Diccionario Enciclopédico Libertad.

Por lo que se plantea como **problema científico**: ¿Cómo viabilizar el almacenamiento y centralización de la información concerniente al Diccionario Enciclopédico Libertad?

Como **objeto de estudio** se tiene: Los repositorios de contenidos, y como **campo de acción**: Los repositorios de contenidos bajo la Java Content Repository (JCR).

Para darle solución a la problemática planteada se traza como **objetivo general**: Desarrollar una herramienta que permita almacenar las plantillas digitales con la información del Diccionario Enciclopédico Libertad.

Para dar cumplimiento a dicho objetivo fue necesario identificar los siguientes **objetivos específicos**:

1. Realizar un estudio de las principales tecnologías y herramientas más comunes para crear o implementar Repositorios de Contenidos.
2. Dar una solución de diseño que permita que la herramienta sea factible para todo tipo de persona que trabaje con ella.
3. Crear una herramienta que permita instalar un Repositorio de Contenido y almacenar los contenidos en él.

Para darle cumplimiento a estos objetivos se realizarán las siguientes tareas por cada objetivo

**Tareas:**

**Objetivo1:**

- Comparar los principales estándares para almacenar contenidos y seleccionar uno.
- Comparar los principales lenguajes de programación y seleccionar uno para implementar el repositorio.
- Comparar las principales metodologías y herramientas de Ingeniería de Software Asistida por Computación o herramientas CASE y seleccionar una para modelar y diseñar la herramienta que se desea implementar.

**Objetivo2:**

- Definir y modelar la arquitectura del sistema.
- Generar los diagramas de clases del diseño y colaboración.

**Objetivos3:**

- Implementar en el lenguaje de programación seleccionado una herramienta que permita crear un repositorio de contenido.
- Crear un instalador de la aplicación que sea multiplataforma.

**La idea a defender** es que con la implementación de un repositorio de contenidos se podría mantener de forma centralizada la información que más tarde contendría el Diccionario Enciclopédico Libertad, posibilitándose además el almacenamiento, la clasificación, la seguridad, el control y las consultas a dichas aplicaciones. Con esta aplicación también se lograría ocultar las complejidades de las tecnologías de la información a los usuarios finales y se minimizaría el costo de la labor humana por procedimientos automáticos.

Atendiendo a las especificaciones planteadas durante el desarrollo de este trabajo se utilizan varios métodos. Dentro de los teóricos se utilizara la Modelación para representar prototipos del sistema que constituyen posibles soluciones a la automatización del proceso de creación del Repositorio. El método Análisis-Sintético para identificar todos los conceptos así como las definiciones más importantes relacionadas con el tema que permita generar una propuesta adecuada a la situación planteada. El análisis Histórico-Lógico para determinar la evolución y desarrollo hasta la actualidad de los repositorios, así como los diferentes repositorios que existen y sus distintos usos.

Entre los métodos empíricos se utilizará la Entrevista con el cliente para poder definir las funcionalidades o requisitos que desee que cumpla el sistema. Otro método que se utilizará es el de Observación mediante el cual podremos valorar los avances realizados en el estudio de las diferentes herramientas y tecnologías a usar.

Para un mejor entendimiento tanto del problema como de la solución que se propone, el siguiente trabajo se ha estructurado de la siguiente manera: Introducción, cuatro capítulos que serán descritos a continuación, las recomendaciones, bibliografía, referencias bibliográficas, y los anexos.

**Capitulo 1**: Fundamentación Teórica, donde se expondrán los principales conceptos relacionados con los repositorios de contenidos, el estándar o especificación JCR, así como un estudio de las herramientas seleccionadas para modelar, diseñar e implementar el sistema.

**Capitulo 2**: Procesos de Negocio, donde se describe el flujo de los procesos involucrados en la solución a modo de comprenderlos totalmente. Se plantea la elaboración del modelo de dominio, los requisitos funcionales y no funcionales del sistema así como la solución propuesta para el sistema que se desea diseñar.

**Capitulo 3**: Diseño del sistema, donde se exponen a través de un conjunto de artefactos la solución que se le dará al problema en cuestión, dentro de los cuales son fundamentales el diagrama de interacción (secuencia y/o colaboración) de los casos de uso más significativos del sistema.

**Capitulo 4**: Implementación del sistema, se define la implementación del sistema, así como las pruebas que se le aplicaran al mismo. La estructura en clases y componentes que garanticen la capacidad operacional del mismo y los defectos y pruebas realizadas al sistema a lo largo del ciclo de vida del producto.

# 1. Fundamentación Teórica

## Introducción

Sin apartarse de las funciones tradicionales de conservadores de la memoria histórica y cultural, los archivos desarrollan un papel fundamental en todo lo relacionado con la gestión integral de cualquier tipo de información durante todo su ciclo de vida, desde su nacimiento hasta su expurgo o conservación permanente, convirtiéndose en piezas claves para las instituciones para las que trabaja. Esta condición aumenta en la actualidad a medida que las instituciones difundan las ingentes cantidades de información contenidas en los documentos primarios que atesoran.

Esta difusión sin embargo, planteará como ya lo ha hecho en otros entornos como el bibliotecario o el académico, problemas para su localización, búsqueda y recuperación. Dichos retos han de solventarse con la creación de un nuevo tipo de estructuras de información y aplicaciones informáticas que no sólo sean capaces de solucionar los problemas archivísticos tradicionales, sino de añadir nuevas funcionalidades basadas fundamentalmente en el acceso y la difusión inteligente de la información. De esta forma será posible crear/producir, y no solo conservar/consumir nuevos recursos electrónicos que estén al servicio de todos los usuarios.

## La conservación de los recursos electrónicos.

Es evidente que las bibliotecas, los archivos y los museos han demostrado su capacidad a lo largo de los siglos de preservar los materiales del pasado. Las profesiones a que dieron vida estas instituciones han establecido normas, criterios, pautas, etc., para guiar las políticas y las acciones en cuanto a la preservación de la memoria intelectual de nuestra civilización universal. Ahora se plantea el gran reto de cómo proteger los recursos digitales, tal y como se ha podido hacer con los materiales en soportes tradicionales.

Las administraciones de diversos países, empresas e instituciones han entendido la necesidad de pasar a la acción, y de establecer políticas y emprender acciones de preservación para asegurar la supervivencia de la producción digital, como ya se había hecho históricamente con los documentos impresos y en soportes tradicionales, mediante las leyes nacionales del depósito legal.

Como es de imaginar las dificultades son notables, para empezar los métodos tradicionales de preservación de la producción bibliográfica son de difícil aplicación en el entorno digital porque, los recursos digitales pueden instalarse en servidores de cualquier parte del mundo. En segundo lugar la producción digital tiene un crecimiento exponencial, siendo aún más variable la durabilidad de los materiales y en consecuencia, limitada la posibilidad de acceso permanente al patrimonio. Finalmente, es preciso señalar la cuestión de la propiedad intelectual del producto digital, sin un derecho basado en el principio de copia para la preservación, que asegure la conservación y perdurabilidad del patrimonio digital, con las limitaciones comerciales que sean necesarias.

Las principales estrategias aplicadas actualmente para frenar la pérdida de información digital son, de forma resumida:

- Preservación de la tecnología
- Migración de los datos
- Emulación de las aplicaciones informáticas originales

Sin embargo, estas estrategias se consideran soluciones a corto plazo a un problema a largo plazo. Es decir, el conocimiento actual más punteros no garantiza la capacidad para preservar lo que está creado para un futuro medio y lejano.

En el escenario que en la actualidad se desarrollan las instituciones u organizaciones y con el desarrollo de las tecnologías de la información y el conocimiento se hace necesario la aplicación de determinados sistemas, métodos, procedimientos y otros instrumentos que respondan a las expectativas en el área de la gestión de la información, la documentación y lo más importante el almacenamiento de toda información generada en medios eficientes como los repositorios de contenidos.

La preservación de los recursos digitales es un reto al que tarde o temprano tendrá que enfrentarse toda institución que pretenda preservar para futuros usuarios lo que se genera hoy en formato digital. Sin lugar a dudas hay un largo camino por delante.

**Los repositorios de contenido.**

Los repositorios de contenido consisten en grandes volúmenes de datos que contienen documentos de textos o imágenes u otros materiales en soporte digital los cuales pueden ser vistos como base de datos no estructuradas. El usuario localiza documentos y objetos haciendo una descripción de ellos. El resultado es que el mismo "**navega**" en el repositorio localizado realizando consultas y refinando resultados hasta que queda satisfecho.
Los mismos se crean por la necesidad de lograr un establecimiento permanente para la inspección pública de objetos y su conservación, fundamentado en una gestión sólida y en firmes garantías financieras (Casals, 2008).

Estos tipos de repositorios combinan algunas ventajas de los sistemas de archivos y de las bases de datos. De los sistemas de archivos, el repositorio adopta el almacenamiento jerárquico de archivos sin estructura y los permisos para el control de acceso. En lo que concierne a las bases de datos, el repositorio también soporta almacenamiento de datos estructurados, consultas, transacciones y comprobaciones de integridad. Los repositorios de contenidos soportan además funcionalidades tales como el control de revisiones o el historial de cambios.

A diferencia de los ordenadores personales o de las PC de escritorio, los repositorios suelen contar con sistemas de Backup y mantenimiento preventivo y correctivo, lo que hace que la información se pueda recuperar en el caso que la máquina o PC quede inutilizable (Casals, 2008). Los mismos suponen además un paso adelante en el trabajo con la capa de datos en los productos software. Frente a las tradicionales bases de datos, proporcionan un acceso mejor y más flexible que las tradicionales consultas SQL. Asimismo, permiten aislarse de la implementación subyacente, lo que implica una mejor portabilidad y escalabilidad.

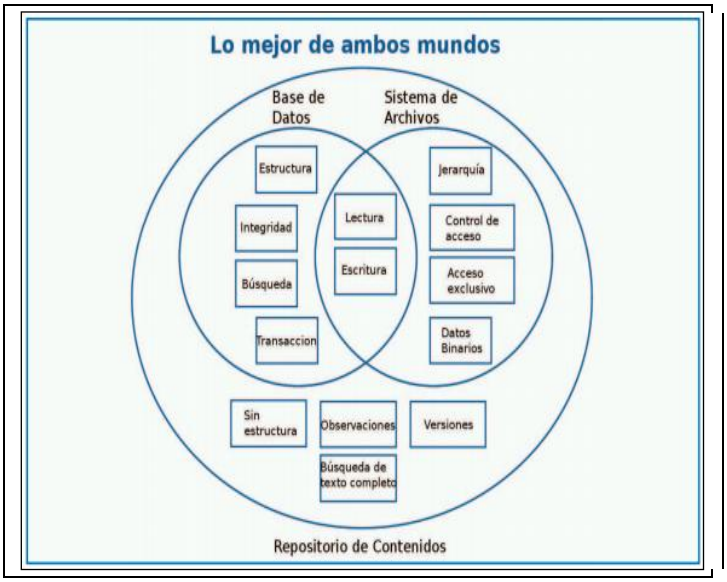



**Figura 1.1 Repositorio de Contenidos**

**Historia**

Los repositorios —**también denominados archivos digitales o depósitos**— constituyen generalmente archivos digitales de los productos generados diariamente por una entidad y que se encuentran accesibles a los usuarios con pocas o ninguna barreras y con la característica de ser interoperables.

El origen de la palabra española repositorio deriva del latín **repositorium**, que significaba armario, alacena. Este significado se generalizó (por metonimia) en español y es recogido en el Diccionario de la Real Academia Española (DRAE) como: **"Lugar donde se guarda algo",** y de ahí se aplicó al léxico específico de la informática para designar los repositorio de información digital.

El primer repositorio en crearse fue *ArXiv*, fundado por *Paul Ginsparg* en 1991 en Los Álamos, Estados Unidos, para la Física de Altas Energías, las Matemáticas y las Ciencias de la Computación. Actualmente, contiene alrededor de 300 000 trabajos y se utiliza ampliamente por investigadores de todos los

continentes. En estos momentos, se administra desde la Universidad de Cornell. Su éxito lo ubica como el modelo de difusión científica más efectivo en el ***Movimiento de acceso abierto*** (Flores, 2007).

En 1996, se creó REPEN, *Research Papers for Economics*, una iniciativa para crear una base de datos de acceso público en economía y disciplinas relacionadas, y en 1997, *CogPrints*, desarrollado por *Steven Harnad* en la Universidad de Southampton, Reino Unido, en el área de psicología, neurociencias y lingüística. En el área de las ciencias biomédicas, se encuentra *PubMed Central*, creado en el 2000 a raíz de la iniciativa de *Harold Varmus*. Hasta mayo de 2007, el *OpenDOAR* recogía 881 repositorios en todo el mundo; el 80 % de ellos se clasifican como institucionales (Flores, 2007).

Paulatinamente han ido surgiendo nuevas ideas y nuevas implementaciones de repositorios de contenidos para ir dándole solución a los distintos problemas que se presentan en determinadas empresas y/o instituciones.

**Tipos de repositorios de contenidos**

De los tantos tipos de repositorios con los que podemos encontrarnos en la actualidad, se han difundido más los repositorios temáticos y los institucionales.

*Repositorios temáticos:*

Fueron los primeros repositorios en aparecer, los mismos recogen documentos científicos y/o académicos de una o varias disciplinas científicas específicas y son los investigadores de diversas instituciones quienes contribuyen autoarchivando sus trabajos. Como ejemplos se pueden mencionar ArXiv (Física, Matemática, Computación y ciencias afines), CogPrints (Psicología), REPEC (Economía), E-Lis (Bibliotecología y Ciencias de la Información) entre otros.

*Repositorios institucionales:*

Recogen la producción de una institución y es la forma más extendida; actualmente, se centran en una organización (universidad, departamento, instituto, sociedades científicas). Es posible definir políticas para

que los miembros añadan contenidos. En esta clasificación, también se incluyen los repositorios de tesis doctorales.

**Ventajas de los repositorios de contenido**

- Conceden a cada institución o participante individual la autonomía necesaria para manipular la información preservada.
- Aseguran que la copia de cada documento preservado sobreviva el tiempo que pueda ser de interés para cualquier interesado potencial.
- Aseguran que cualquier consumidor autorizado pueda encontrar y usar cualquier contenido preservado independientemente de cualquier error cometido por algún tercero, asegurando de esta manera la disponibilidad de los contenidos.
- Aseguran que cualquier consumidor tiene evidencia accesible para determinar cuándo la información recibida es suficientemente confiable para la aplicación, velando además por la integridad de dicha información.
- Ocultan las complejidades de las tecnologías de la información para los usuarios finales.
- Minimizan el costo de la labor humana por procedimientos automáticos siempre y cuando sea factible.
- Permite manipular contenidos que no pueden ser soportados en papel (Flores, 2007).

**Sistemas existentes en el mundo. Políticas**

La mayor cantidad de repositorios se centra en Estados Unidos y Europa. En esta última región, lideran los países Alemania y Reino Unido. El 80 % de estos repositorios corresponde a los institucionales en los cuales predominan las tesis y disertaciones, los informes no publicados, los artículos científicos (preprint y postprint) y las presentaciones en eventos.

- Las políticas de preservación digital y las referidas a los procedimientos para el envío de documentos al repositorio no están definidas en un número considerable de estos, lo que constituye una debilidad para la sostenibilidad exitosa de los repositorios pues de estas políticas depende el crecimiento y conservación del patrimonio científico de dichas instituciones.

  Las políticas de preservación digital se encuentran indefinidas principalmente en Sudáfrica y Nueva Zelanda y en la mayoría de los repositorios de la India, Japón, Australia, Chile y los países seleccionados en la región europea.

- Las políticas de uso, calidad y normalización de metadatos no están definidas en un elevado porcentaje de repositorios. En aquellos que la definen, se establecen permisos para usos comerciales y no comerciales. El establecimiento de políticas que favorezcan el rehúso de los metadatos con fines no lucrativos es una estrategia consecuente con todo el movimiento para ampliar el acceso y el intercambio de información.

  Las políticas de uso, calidad y normalización de metadatos no se definen completamente en la India, Japón, Australia, Nueva Zelanda, Chile, Brasil, Alemania, Italia, Holanda, Canadá y Estados Unidos. Los procedimientos para el envío de documentos no se definen en un número considerable de los repositorios implementados en la India, Japón e Italia.

- Las políticas de propiedad intelectual en muchas regiones se encuentran indefinidas. En sentido general, estimulan que los autores mantengan el derecho de autor sobre sus trabajos y cumplan con las legislaciones establecidas en cuanto a períodos de embargo y licencias de los editores. En algunos casos, se controla o se restringe la accesibilidad a los textos completos de acuerdo con los propósitos de los usuarios.

Aunque no abunda la información relativa a los costos de implementación de repositorios institucionales, se pudo percibir la creencia de que su costo es muy bajo, tal vez por la existencia de una gran cantidad de plataformas de software libre que permiten la gestión de documentos digitales. Sin embargo, en este aspecto, es necesaria una inversión importante en recursos humanos, sobre todo en la etapa inicial de implementación, además de un servidor potente de acceso rápido. Los costos de desarrollo, pero sobre todo de mantenimiento y conservación, aún no se conocen del todo.

Es importante estimular la participación de los usuarios, sea mediante la promoción de las potencialidades de los repositorios para lograr visibilidad e impacto en la comunicación científica, como por medio de políticas nacionales o institucionales que establezcan la obligatoriedad del depósito. Como reflejan investigaciones recientes, sólo con este enfoque integrado podrán lograrse altos porcentajes de auto-archivo.

Tras un estudio de los principales conceptos relacionados con los repositorios de contenidos y la necesidad del cliente de crear uno para la gestión de la información producida y preservada se hace un estudio de las principales tecnologías y herramientas que existen hoy para darle solución a la problemática.

**Lenguajes de Programación**

Los lenguajes de programación son un conjunto de símbolos, caracteres y reglas (programas) que les permiten a las personas comunicarse con la computadora. Además son la notación para la descripción precisa de algoritmos o programas informáticos. Son el conjunto de instrucciones que permiten al programador pensar claramente sobre la complejidad del problema a resolver, de manera que pueda ordenarlas convenientemente para la creación de un programa ejecutable por la computadora. Los mismos tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, cálculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación.

Teniendo en consideración que se realizará una aplicación de escritorio se hace a continuación una relación de los principales lenguajes de programación que se emplean mundialmente para el desarrollo de dichas aplicaciones a fin de encontrar mediante un análisis de las características individuales de cada uno, el que se adecue más a las necesidades de este trabajo.

**CSharp(C#)**

C Sharp (C#) es un lenguaje de programación orientado a objetos que utiliza el modelo de objetos de la plataforma .NET. Diseñado para crear una amplia gama de aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Es un lenguaje similar a Java aunque incluye mejoras derivadas de otros lenguajes.

Fue diseñado para combinar el control del lenguaje de bajo nivel como C y la velocidad de programación de lenguajes de alto nivel como Visual Basic.

Posee ventajas frente a otros lenguajes debido a que es un lenguaje simple, moderno, de propósito general orientado a objetos. Es usado para desarrollar componentes de software que se pueden usar en ambientes distribuidos. Las características de la recolección de elementos no utilizados y la compatibilidad con las clases de .NET Compact Framework hacen que sea un idioma ideal a la hora de desarrollar aplicaciones móviles confiables y seguras.

Es un lenguaje simple, eficaz y con seguridad de tipos. Con sus diversas innovaciones, C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C. Posee un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz, además de otras herramientas.

El proceso de generación de C# es simple en comparación con el de C y C++, y es más flexible que en Java. No hay archivos de encabezado independientes, ni se requiere que los métodos y los tipos se declaren en un orden determinado. Un archivo de código fuente de C# puede definir cualquier número de clases, estructuras, interfaces y eventos (Basic.net, 2008).

**Visual Basic**

Visual Basic es una herramienta. Simplifica en gran medida la tarea de trasladar una aplicación de escritorio a un dispositivo móvil o de crear rápidamente una aplicación cliente enriquecida. Al igual que ocurre con C#, Visual Basic utiliza .NET Compact Framework. Los desarrolladores, ya familiarizados con Visual Basic, podrán trasladar las aplicaciones existentes o crear otras nuevas de forma muy rápida.

Está diseñado para generar de manera productiva aplicaciones con seguridad de tipos y orientadas a objetos. Permite a los desarrolladores centrar el diseño en Windows, la Web y dispositivos móviles. Como con todos los lenguajes que tienen por objetivo Microsoft .NET Framework, los programas escritos en este lenguaje se benefician de la seguridad y la interoperabilidad de lenguajes. Esta generación de Visual

Basic continúa la tradición de ofrecer una manera rápida y fácil de crear aplicaciones basadas en .NET Framework.

Incluye nuevas características para el desarrollo rápido de aplicaciones. Una de estas características proporciona acceso rápido a las tareas frecuentes de .NET Framework, así como información e instancias de objeto predeterminadas que estén relacionadas con la aplicación y su entorno en tiempo de ejecución. Las nuevas características de idioma incluyen la continuación de ciclos, la eliminación garantizada de recursos, la sobrecarga de operadores, los tipos genéricos y los eventos personalizados. Proporcionan interoperabilidad de lenguajes, recolección de elementos no utilizados, seguridad mejorada y control de versiones.

**C++**

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, como extensión del lenguaje de programación C. Es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (Taringa.net, 2007). Las principales características de C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica (templates). Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel como son la posibilidad de redefinir los operadores (sobrecarga de operadores) e identificación de tipos en tiempo de ejecución. C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel.

Tanto C como C++ son lenguajes de programación de propósito general. Todo puede programarse con ellos, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos y procesadores de texto, pasando por juegos, aplicaciones a medida, entre otros tipos. Es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original.

C++ es el lenguaje de desarrollo que se prefiere cuando el rendimiento es fundamental o a la hora de desarrollar aplicaciones de nivel de sistema, controladores de dispositivos o complementos de pantalla. C++ no admite .NET Compact Framework, pero en su lugar proporciona un subconjunto del conjunto de (API, por sus siglas en inglés) Win32. Esto es posible para aplicaciones escritas en código de C# administrado o de Visual Basic para tener acceso a código de C++ contenido en archivos de biblioteca de enlaces dinámicos (DLL, por sus siglas en inglés) mediante interoperabilidad.

**Java**

Java es un lenguaje de programación orientado a objetos que toma muchas de sus sintaxis de C y C++, pero tiene un modelo de objetos más simples y elimina herramientas de bajo nivel como punteros (HELLFREDMANSON, 2009). Se creó con cinco objetivos principales: usar la metodología de la programación orientada a objetos, permitir la ejecución de un mismo programa en múltiples sistemas operativos, incluir por defecto soporte para trabajo en red, ejecutar código en sistemas remotos de forma segura, ser fácil de usar.

Una de sus características es la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware y/o software. Es un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras, que se utiliza para realizar aplicaciones en Internet.

Este lenguaje es independiente de la plataforma y posee un entorno de ejecución ligero y gratuito. Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java se ha convertido en un componente habitual en los PCs de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se pueda ejecutar en cualquier PC.

En las primeras versiones existían importantes limitaciones en las APIs de desarrollo gráfico, cosa que en la actualidad ya no ocurre. Actualmente el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, entre otras es relativamente sencillo.

No existe un paradigma de diseño ni un lenguaje de programación que se ajuste a todas las necesidades, por lo cual debe escogerse en cada caso la tecnología que mejor satisfaga los requerimientos. Es por ello que para llevar a cabo el desarrollo de la herramienta que es objeto de esta investigación se decidió el uso del lenguaje de programación **Java** por las ventajas y potencialidades que tiene frente a otros lenguajes, al poder implementarse con independencia de la plataforma y como software libre, permitiendo obtener productos de excelente calidad, en menor tiempo y, por consiguiente, con menores costos (Ciberaula, 2007).

**Plataforma de desarrollo.**

En informática, una plataforma de desarrollo se le denomina al entorno de software común en el cual se desarrolla la programación de un grupo definido de aplicaciones. Generalmente se encuentra vinculada directamente a un sistema operativo; sin embargo, también se puede encontrar ligada a una familia de lenguajes de programación o a una interfaz de programación de aplicaciones.

**Microsof.NET**

La Infraestructura .NET describe todas las tecnologías que conforman el nuevo ambiente para concebir y ejecutar aplicaciones robustas, escalables y distribuidas. Microsof.NET propone una serie de características como son: una interfaz de usuario, datos de acceso, la conectividad de bases de datos, criptografía, desarrollo de aplicaciones web, algoritmos numéricos y de comunicaciones de red. (Canal_Visual_Basic.net. 2008)

La Plataforma .NET está conformada por un Lenguaje Común en Tiempo de Ejecución (CLR, por sus siglas en inglés) y la Biblioteca de Clases de la Plataforma .NET algunas veces llamada la Biblioteca de Clases Base (CBL, por sus siglas en inglés). Los programas desarrollados para .NET Framework se realizan en un entorno de software que resuelve las exigencias de tiempo de ejecución del programa. El CLR suministra la apariencia de una aplicación de la máquina virtual para que los desarrolladores no tengan en cuenta las capacidades específicas de la CPU que ejecutará el programa. El CLR además proporciona otros servicios significativos como la seguridad, la gestión de memoria, y manejo de excepciones. La biblioteca de clases es utilizada por los programadores, que unen con su propio código

para producir aplicaciones y también ofrece soporte a los archivos de entrada y salida como para las bases de datos. Sus principales características son:

- **Multilenguaje**: Todos los lenguaje de programación pueden adaptarse a la plataforma .NET y ejecutarse en ella.
- **Interoperabilidad**: La interoperabilidad entre los distintos trozos de códigos escritos es total.
- **Portabilidad**: Debido a la abstracción del programador respecto a los sistemas operativos una aplicación .NET puede ser ejecutada en cualquiera de ellos siempre y cuando disponga de una versión de la plataforma.

La Plataforma .NET consiste de dos componentes principales:

- **El Lenguaje Común en Tiempo de Ejecución** el cual es el fundamento de la Plataforma .NET
- **La Biblioteca de Clases de la Plataforma .NET o Framework Classes**, es una colección orientada a objetos de tipos reusables que pueden utilizarse para desarrollar aplicaciones en el rango de aplicaciones tradicionales desde la línea de comandos o interfaces de usuario gráficas (GUI, por sus siglas en inglés) hasta aplicaciones basadas en las últimas innovaciones que provee ASP.NET tales como Web Forms y servicios web XML.

**J2SE**

Java Platform, Standard Edition o Java SE (conocido como Plataforma Java 2, Standard Edition o J2SE) nos proporciona un entorno de escritorio Core Java y desarrollo de aplicaciones Java, y es la base de Java 2 Platform, Enterprise Edition (J2EE) y tecnologías Java Web Services. Tiene el compilador, herramientas, módulos de ejecución, y la API de Java que le permiten escribir, probar, implementar y ejecutar applets y aplicaciones.

J2SE incluye nuevas actualizaciones del lenguaje de programación Java centradas en la facilidad de despliegue, nuevas funcionalidades de gestión y monitorización de aplicaciones, mejora del soporte al

cliente en equipos de sobremesa y un mayor rendimiento. J2SE es la plataforma software de despliegue y desarrollo utilizado por muchos fabricantes y desarrolladores para suministrar una amplia gama de servicios y contenidos seguros para redes. J2SE impulsa Sun Java Enterprise System, las herramientas Sun Java Studio Enterprise y Sun Java Desktop System.

"*La tecnología Java ofrece a los desarrolladores la mejor plataforma para innovar y desarrollar fácilmente",* afirma José Manuel Estrada, arquitecto Java en Sun Microsystems Ibérica. "*J2SE supone un importante logro para la comunidad de desarrolladores Java y es un hito muy importante para dicha tecnología. Estas mejoras en el lenguaje de programación benefician a una amplia variedad de desarrolladores de este lenguaje, ya que permiten un mayor aprovechamiento de las posibilidades que ofrece la plataforma Java". (*Noticiasdot.com. 2004*)*

J2SE es la primera plataforma para la tecnología Java y está experimentando una gran demanda en el entorno de los sistemas de sobremesa. Con más de 110 millones de descargas del Software Developer Kit (SDK) de J2SE y Java Runtime Environment (JRE) desde su disponibilidad en diciembre de 1998, su popularidad e impacto en la Informática en Red es indiscutible. La tecnología Java es una clara alternativa para los desarrolladores que emplean J2SE para desplegar aplicaciones para sistemas de sobremesa como la gestión financiera y visualización de datos. Las aplicaciones de comunicación como el chat a través de la web y las de mensajería instantánea están basadas en J2SE, así como una gran variedad de juegos multijugador, muchos de los cuales se pueden descargar en http://java.com/es/.*(*Noticiasdot.com. 2004*)*

Esta nueva plataforma demuestra el compromiso de Sun con la constante innovación sobre la plataforma Java. Las actualizaciones del lenguaje permiten que los desarrolladores sean más eficaces y productivos al soportar una codificación más rápida y segura. Las nuevas funcionalidades incluyen los tipos genéricos, los tipos enumerados, metadatos y la conversión de tipos primitivos. Las mejoras en el rendimiento de la nueva versión incluyen una reducción en el tiempo de inicio, así como un menor espacio de la memoria de impresión para conseguir un mayor rendimiento del desarrollo de aplicaciones en la nueva plataforma. *(*Noticiasdot.com. 2004*)*

En esta plataforma son claves la monitorización y facilidad de manejo, la cual incorpora algunos cambios mejorados que posibilitan que las aplicaciones basadas en tecnología Java desarrolladas sobre dicha plataforma sean desplegadas en los sistemas existentes de gestión empresarial basados en Simple Network Management Protocol (SNMP). Esto permitirá que el software JVM sea monitorizado y gestionado para lograr niveles de fiabilidad, disponibilidad y servicio más elevados.

**J2EE**

Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una especificación. Similar a otras especificaciones del Java Community Process (JCP), Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes a Java EE; estandarizado por la JCP.

Java EE incluye varias especificaciones de API, tales como conectividad de la base de datos de Java (JDBC, por sus siglas en inglés), Remote Method Invocation (RMI), e-mail, servicio de mensajes Java (JMS), Servicios Web, XML, entre otros y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, integrable a la vez con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de ponerse en función de las tareas de mantenimiento de bajo nivel.( Creación_de_ Software,2009)

Para el desarrollo de la aplicación que se propone en conjunto con el lenguaje de programación, se seleccionó la plataforma J2SE debido a la robustez de la misma, asegurando así que la aplicación resultante sea lo más ligera en términos de consumo de recursos de la PC del cliente.

**JSR-170**

**El Proceso de la Comunidad Java**, o Java Community Process, establecido en 1998, es un proceso formalizado el cual permite a las partes interesadas a involucrarse en la definición de futuras versiones y características de la plataforma Java.
El proceso JCP conlleva el uso de Java Specification Request (JSR), las cuales son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java. Las revisiones públicas formales de JSRs son controladas antes de que los JSR se conviertan en final y sean votados por el Comité Ejecutivo JCP. Un JSR final suministra una implementación de referencia la cual da una implementación libre de la tecnología en código fuente y un Kit de Compatibilidad de Tecnología para verificar la especificación de la API.( Torres,C.2009). En el año 2002 con el auge del los sistemas de administración del contenido (CMS), ya sean web o imágenes o cualquier otro tipo, los vendedores de dichos sistemas se habían dado la tarea de crear implementaciones específicas o propias de su repositorio de contenido, los cuales cambian incluso, en determinados casos, en distintas versiones del mismo producto. Esto era sin duda alguna un problema para los desarrolladores de aplicaciones sobre dichos sistemas, los cuales tenían que aprender cómo trabajar con la API que les proveía el vendedor del sistema, lo cual traía además que su implementación se viera atada a ese tipo de sistema específicamente siendo así costoso para planes futuros de cambiar a otro tipo de versión del producto o de sistema. Por lo que la Comunidad presentó el proyecto de una nueva especificación.

La JSR – 170 se define a sí misma como “un estándar o norma, con una implementación independiente para tener acceso bidireccional en un nivel determinado dentro de un repositorio de contenido” y va más allá definiendo un repositorio de contenidos como “un sistema de alto nivel de administración de la información que supone un nivel superior a los tradicionales repositorios de datos” el cual implementa servicios sobre los contenidos tales como: versionado, búsqueda, control de acceso, categorización del

contenido, monitorización de los eventos sobre un contenido o auditoría del mismo, entre otros tantos.( Sunil, P. 2006)

La API JCR definida por la norma o estándar JSR-170, hace un intento por estandarizar una API que pueda ser usada para acceder a un repositorio de contenidos de la manera más trasparente posible. Su lanzamiento final fue en el año 2003. ( Java_Community_Process. 2006.)

**Necesidad de la API de Java para repositorios de contenidos (JSR – 170).**

Con el incremento del número de vendedores propietarios de repositorios de contenidos, se hace necesaria una interfaz de programación común para acceder y consumir los servicios ofertados por estos repositorios de contenidos y es aquí donde comienza a jugar su papel la JSR – 170 la cual provee una interfaz que puede ser usada para conectarse a cualquier tipo de repositorio. Se puede ver la JSR- 170 como la API JDBC para repositorios de contenidos, permitiéndole al programador desarrollar su programa independientemente de cualquier implementación de repositorio de contenidos. En tiempo de ejecución se puede configurar el programa para trabajar con cualquier implementación nativa de la JSR – 170 como JackRabbit o Communique. En caso que el repositorio no sea nativo de la JSR – 170 como Documentum o Vignette, entonces se debiera usar algún tipo específico de puente o adaptador que se encargue de convertir los métodos de la JSR – 170 en métodos de la API usada.

En la figura siguiente se describe la estructura de una aplicación que usa la API propuesta por la JSR – 170. En tiempo de ejecución, esta aplicación puede trabajar con cualquiera de los repositorios 1, 2, 3 de los cuales sólo el repositorio 2 es nativamente JSR – 170, los otros dos necesitan un adaptador [driver] para interactuar con la JSR – 170. Otra cosa importante es que la aplicación no tiene que preocuparse por dónde o cómo se almacena el contenido; mientras el repositorio1 usa un sistema de bases de datos relacionales, el repositorio 2 usa un sistema de ficheros, el repositorio 3 usa XML; de la misma manera que pudiera existir otro que usara para el almacenamiento una combinación de cualquiera de estos.

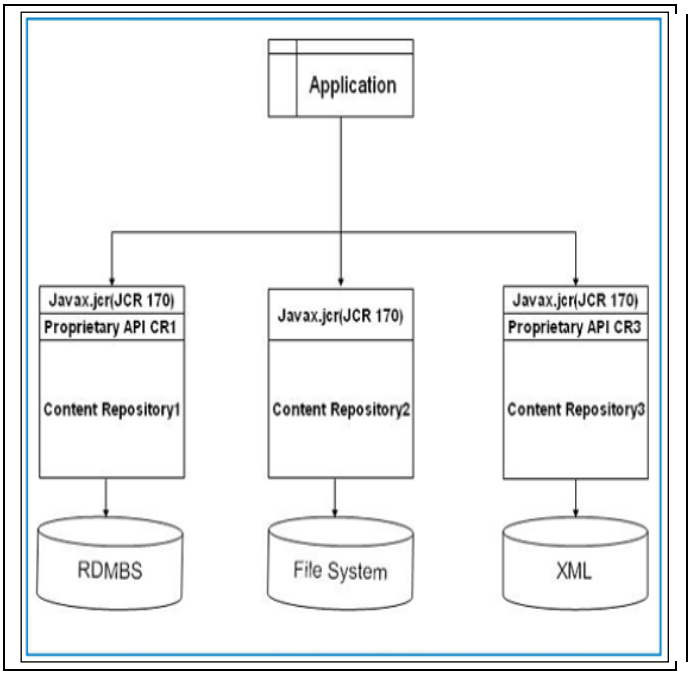



**Figura 1.2 Propuesta de la API por la JSR-170**

**Objetivos de la JSR – 170:**

No debe estar atada a ninguna arquitectura, implementación o protocolo: El API es, claro está, esencialmente un conjunto de interfaces Java, las cuales pueden ser implementadas en variedad de formas. El principal reto es permitir suficiente flexibilidad en la API de modo que pueda ser usada tanto en modelos de repositorios jerárquicos como no jerárquicos.

Debe ser fácil de usar, desde el punto de vista de los programadores: De cierta manera la API está diseñada para ser lo más simple y ligera posible. Particularmente, la API tiene un simple modelo de objetos y más bien se concentra en la representación de las funcionalidades del núcleo del repositorio.

Debiera permitir una implementación relativamente fácil: Se ha hecho un gran esfuerzo por asegurar que la API será relativamente fácil de implementar en base a los mejores vendedores de repositorios de contenido.

Debiera estandarizar además algunas funcionalidades más complejas que pudieran ser una necesidad para aplicaciones más avanzadas relacionadas con el contenido: Reconociendo que existe una gran tensión entre este objetivo y el anterior la especificación se divide en diferentes niveles de conformidad, para así permitir a los diferentes fabricantes de repositorios de contenido adaptarse a ella como deseen y evitar que un número innecesariamente alto de estos no pueda adaptarse a la interfaz porque sólo quieran implementar una parte de la funcionalidad. JSR-170 especifica un Nivel1, un Nivel2 y un conjunto de características avanzadas opcionales para los repositorios.

- **Nivel 1, Sólo lectura**: proporciona una API de sólo lectura y de fácil implementación para cubrir una serie de aplicaciones que tan solo necesitan buscar y obtener información de los repositorios de contenido. Este nivel proporciona métodos de introspección en los nodos y propiedades y ofrece un acceso de lectura jerárquico al contenido almacenado en el repositorio.

Este nivel está pensado para aplicaciones como Porltes[1], Plantillas de Sistema de gestión de contenidos (CMS, por sus siglas en inglés), Informes, Exportaciones, u otras aplicaciones que se dediquen a recolectar, buscar, presentar o mostrar información de uno o más repositorios.

- **Nivel 2, Repositorios modificables**: especifica todas las herramientas de modificación necesarias para interaccionar bidireccionalmente con un repositorio de contenido de modo elegante y granularizado.

  Entre las aplicaciones escritas usando la API de Nivel2 se destacan las aplicaciones de gestión o en general cualquier aplicación que genere información, datos o contenidos, tanto en forma estructurada como desestructurada.

1

Los Porlets son otras de las especificaciones Java más usadas en la actualidad. Básicamente son pequeñas aplicaciones capaces de ser integradas en portales que, a modo de ventanas, generan tan sólo su propio contenido. El portal es capaz de manejar una unión de diferentes porlets para generar las diferentes vistas del mismo. Para ver más puede visitar la web de la especificación porlet, JSR-168: http://jcp.org/en/jsr/detail?id=168

- **Características avanzadas**: Sobre los niveles 1 o 2 pueden existir un número de elementos que especifican funcionalidades avanzadas para los repositorios. Entre estos se incluyen herramientas como:
  - Versionado (*versioning*)
  - Transacciones (JTA)
  - Consultas usando SQL
  - Observación y bloqueo explícito de contenidos.

**El modelo del repositorio*:***

Un repositorio de contenidos según la JSR-170 consiste en un conjunto de espacios de trabajos (workspaces), aunque por lo general se usa uno solo, el cual contiene un árbol de elementos (items). Un elemento pudiera ser un nodo (node) o una propiedad (property). Cada nodo puede tener cero o más nodos hijos y cero o más propiedades hijas. Cada espacio de trabajo contiene uno y sólo un nodo raíz, el cual no tiene padre, o sea, que no es hijo de ningún otro nodo. Todos los otros nodos tienen un padre. Las propiedades por su parte siempre tienen un padre, que siempre será un nodo y no tendrán hijos, son consideradas las hojas del árbol. Todo el contenido en un repositorio es almacenado dentro del valor de una propiedad.

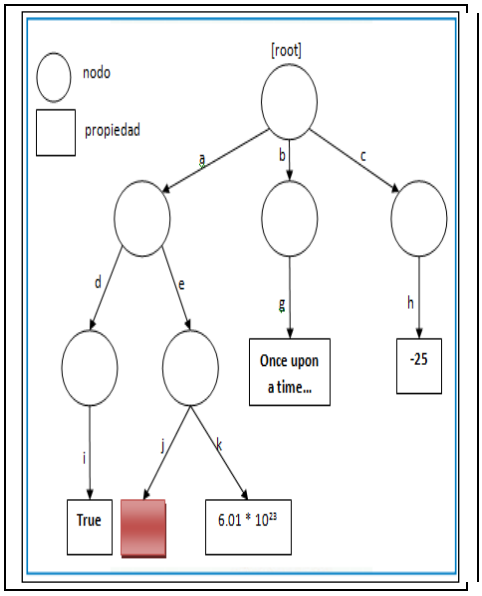



**Figura 1.3 Modelo de Repositorio**

**En el diagrama anterior se observa el único nodo raíz d un espacio de trabajo determinado el cual tiene los nodos hijos a, b** y **c** los cuales tiene cada uno sus propios hijos y propiedades. Por ejemplo, el nodo **a** tiene dos nodos hijos, que son **d** y **e**. El nodo **e** tiene las propiedades **j** y **k**. La propiedad **j** contiene una imagen mientras que la **k** contiene un valor en coma flotante ($6.01 * 10^{23}$). De la misma manera la propiedad i contiene el valor booleano True.

Cada elemento en la jerarquía puede ser identificado por un camino (path) absoluto. Por ejemplo, el path absoluto / se refiere al nodo raíz, mientras que el path /a/d/i se refiere a la propiedad con valor *True*. Los caminos absolutos siempre comienzan con el carácter /.

Un camino relativo especifica un nodo o una propiedad relativa a otra localización dentro de la jerarquía. Por ejemplo, relativo al nodo */a* en el diagrama anterior, el camino a la propiedad con valor *True* es */d/i.*

**Ventajas de la JSR – 170**. ( Sunil, P. 2006)

La JSR – 170 en conjunto con la API JCR tiene diferentes ventajas sobre los usuarios de un repositorio de contenido.

- **Desarrolladores**: No tienen que perder tiempo aprendiendo a usar la API específica del vendedor o proveedor del sistema de gestión de contenidos. Con la JSR el desarrollador debe ser capaz de trabajar con cualquier compilación de la JCR – 170. Antiguamente tenían que elegir entre un buen sistema con grandes características y pobres herramientas de desarrollo o un sistema con buenas herramientas de desarrollo pero pobres características. Ahora que la interfaz entre el repositorio de contenidos y el sistema de administración del contenido está estandarizada, el desarrollador puede seleccionar las mejores opciones de las dos partes.

- **Corporaciones**: No tendrán que enfrentarse a problemas de dependencias de un proveedor específico. Comúnmente las corporaciones tienen más de un sistema de gestión de la información, ya sea porque los departamentos usan diferentes sistemas o porque la corporación adquiera nuevas empresas que usan sistemas diferentes. Anteriormente, en el pasado, las corporaciones gastaban cantidades considerables de dinero intentando que estos diferentes sistemas interactúen entre ellos. Ahora con la JSR – 170 las corporaciones pueden asegurar que la misma aplicación puede trabajar con todos los sistemas.

- **Vendedores de sistemas**: Se veían forzados a desarrollar y mantener sus propias implementaciones de repositorios de contenidos, los cuales significan grandes cantidades de infraestructuras de código. Ahora ellos pueden dejarle la implementación de repositorios de contenidos a otros grupos de desarrollos y concentrar sus esfuerzos en otras competencias de su verdadero producto.

**JackRabbit**

JackRabbit es una aplicación 100% compatible con la JSR – 170, el cual tiene implementadas todas las funcionalidades especificadas por la JSR – 170 hasta el nivel 2 e incluye la mayoría las características opcionales señaladas por esta especificación. Además de eso, más allá de la API JCR, JackRabbit tiene numerosas extensiones y características administrativas que son necesarias para trabajar con el repositorio y que no son especificadas por la JSR – 170.

**Arquitectura de JackRabbit**

En la siguiente figura se muestra la arquitectura de JackRabbit, la cual puede ser descrita en tres capas: Capa de Aplicaciones de Contenido, API para el repositorio de contenidos y la de Implementación del repositorio de contenidos.

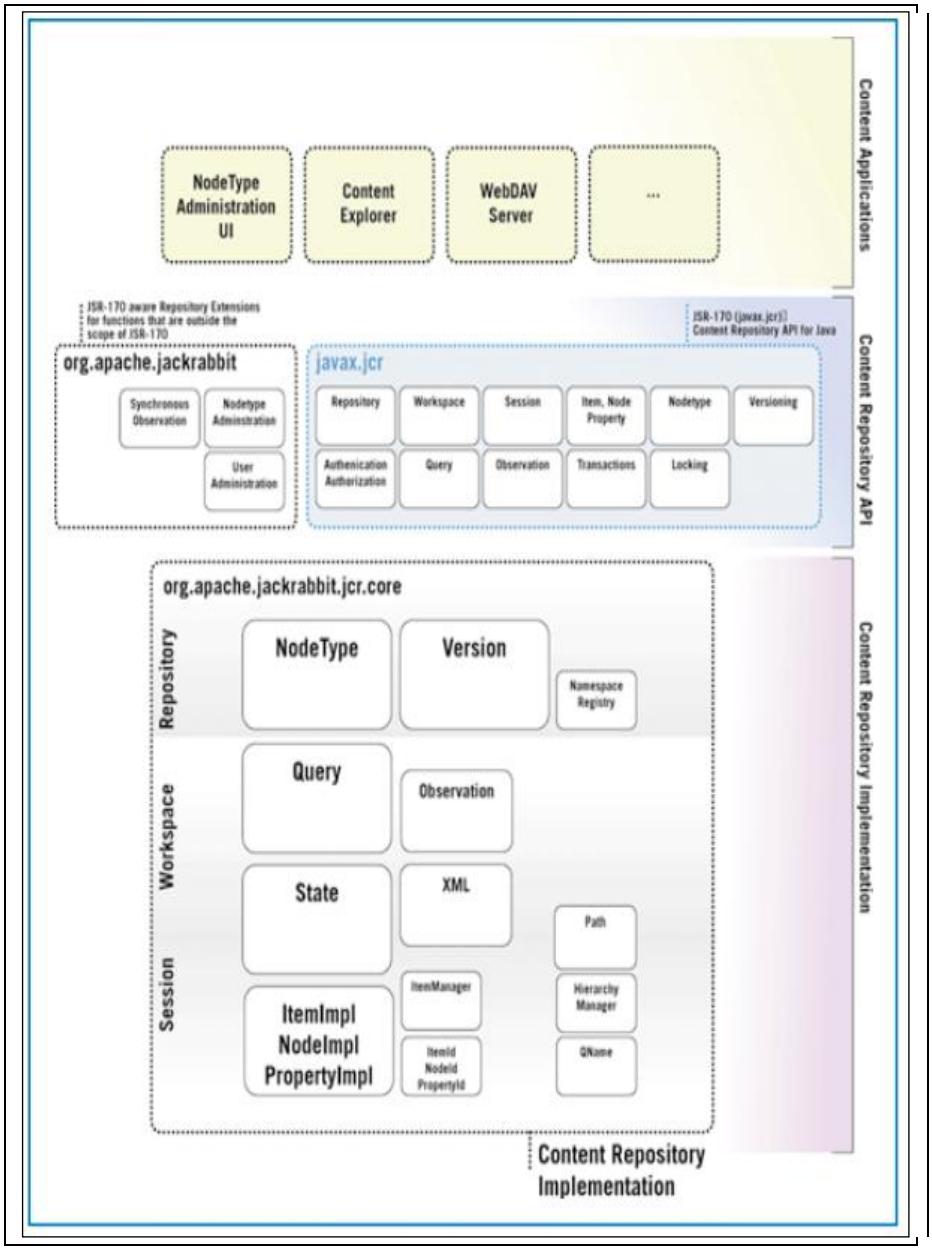



**Figura 1.4 Vista arquitectónica de Jackrabbit**

Las **aplicaciones de contenido** interactúan con el repositorio a través de la API JSR – 170. Existen numerosas aplicaciones que están disponibles para interactuar con los repositorios de contenido de la

JSR – 170, algunas de ellas genéricas como WebDAV server y otras pueden ser muy específicas y solo usan el repositorio de contenido como un almacén para guardar la información que es usada y generada por las distintas aplicaciones.

La **API para el repositorio de contenidos** está dividida en dos secciones

4. La API de repositorios de contenidos definida por la JSR-170.
5. Conjunto de características de los repositorios de contenidos, que no son contempladas en la JSR – 170, como son la sincronización, la administración de los tipos de datos y de los usuarios, entre otras.

La porción de la **Implementación del Repositorio de Contenidos** refleja los bloques de construcción más importantes de los repositorios de contenidos. El tamaño de los bloques simboliza aproximadamente la cantidad de código y por tanto la complejidad de los bloques funcionales individuales. Existen tres ámbitos en el repositorio de contenidos: (repositorio, espacio de trabajo y sesión) a los cuales atribuyen indistintamente todas las operaciones que se ejecutan sobre el repositorio.

**Estado del Arte**

**Hippo CMS**

Hippo CMS es un sistema centrado en el manejo del contenido, posee una interfaz gráfica de usuario basada en web que proporciona acceso al contenido en el repositorio. Proporciona una forma fácil de crear y administrar contenido, es decir, para escribir o subir, compartir, imprimir, buscar, consultar, editar y estructurar contenidos, ceder los derechos para hacerlo, tienen un contenido validado y las reglas de su negocio de restauración de versiones anteriores del contenido. Hippo CMS posee la interfaz estándar JCR mediante la cual tiene acceso directo al contenido dentro del repositorio, y todas las funcionalidades de gestión de contenidos del mismo. Hippo CMS se ejecuta como un servlet de Java y requiere un contenedor de servlets Java, como Tomcat o un servidor de aplicaciones Java Glassfish como para ejecutar la aplicación, además debe contar con un sistema gestor de base de datos - recomendablemente MySQL – encargado de almacenar los documentos binarios como son los ofimáticos,

las imágenes entre otros, los cuales se guardan como un solo paquete. Está bajo la licencia de código abierto lo que hace posible que esté disponible para cualquier usuario. (Hippo_Community_Site. 1999)

**Nuxeo**

Nuxeo es un software libre que ha sido diseñado para ser robusto altamente escalable y extensible, mediante el uso de código abierto. Usa tecnologías tales como el JCR , JSF , EJB3 , JBoss Seam , OSGi , y un enfoque orientado a Servicio . Puede ser utilizado para desarrollar aplicaciones basadas en servidor, tanto-web como aplicaciones de escritorio. En la actualidad brinda todo una gama de servicios como son la administración de contenidos, trabajo colaborativo, gestión de activos digitales, entre muchos otros. Para obtener resultados óptimos en el almacenamiento de documentos se recomienda el uso del sistema gestor de base de datos PostgreSQL. En términos legales se distribuye bajo la licencia LGPL.( Nuxeo_The_Community. 2006)

**OpenKM**

OpenKM es un sistema de gestión documental que permite, según sus desarrolladores, una gestión del conocimiento más sencilla. La misma está basada en tecnología J2EE y JBoss, puede ser instalada y ejecutada en varias plataformas. Otras tecnologías utilizadas son Jackrabbit y GWT (Google Web Toolkit – AJAX). Entre sus funcionalidades se encuentran: funciones de gestión del conocimiento y gestión documental, entre otras. Los requisitos para una pequeña instalación con menos de 25 usuarios y un tamaño de repositorio comprendido es entre 10-60GB, necesita de 1GB de memoria RAM, para su procesamiento 1.86 GHz y 150-250 GB de disco duro y para un mejor rendimiento esta necesita de 2GB a 4GB de memoria RAM.(OpemKM_Knowledge_Management. 2006). OpenKM está bajo la licencia GPL.

**Magnolia CMS**

Magnolia CMS es una de las opciones de gestión de contenido de Código Abierto, que permite la gestión de elementos digitales como ficheros de imágenes, audio y vídeo que pueden ser incluidos en cualquier página creada con Magnolia directamente desde el entorno. Entre sus características, incluye el cambio automático del tamaño y recorte de las imágenes, gestión de metadatos, galería de imágenes, entre otras funcionalidades y está basado en JSR-170. Magnolia incorpora varias herramientas de colaboración social

como foros, comentarios y registro de usuarios. Éste tipo de contenido generado por el usuario se puede agrupar en varios servidores para una máxima escalabilidad. La misma presenta una versión empresarial la cual es sumamente costosa y una comunitaria la cual esta accesible para todos. Magnolia CMS almacena todo el contenido (páginas web, imágenes, documentos, configuración, datos) en un repositorio de contenido. La implementación elegida para el repositorio es Apache Jackrabbit. (Magnolia, 2003)

**Consideraciones sobre los Gestores de Contenidos**

A pesar de la facilidad para la creación de los contenidos en Hippo CMS, su funcionamiento depende de la instalación de otros servicios como MySQL y contenedores de servlets Java. Esos servicios consumen gran cantidad de recursos (dígase CPU y memoria física), recursos de los que no disponen los clientes del proyecto: Diccionario Enciclopédico. Igualmente Nuxeo ofrece una gran gama de servicios, donde se incluye el trabajo colaborativo, lo cual va más allá de las necesidades del cliente. Necesita además del funcionamiento de un servicio similar a MySQL, esta vez PostgreSQL. El mismo tiene la característica de ser muy eficiente ante una alta demanda de procesamiento de información, lo cual no sería óptimo a aplicar, puesto que el tráfico de información sería bajo. OpenKM por su parte es uno de los sistemas de gestión documental más usados a nivel mundial por su sencilla gestión del conocimiento, parte de su tecnología está basada en JBoss, este es un servidor de aplicaciones usado en el mundo empresarial y para su rendimiento debe contar como mínimo con una memoria RAM de 2 GB, lo que tampoco cumple con los recursos disponibles. Magnolia CMS es la que más se adecua a las necesidades existentes, pero la misma es más un gestor de contenidos web y ofrece más servicios de los necesitados para el almacenamiento de la información para el Diccionario Enciclopédico. Por tal motivo surge la necesidad de implementar una herramienta que de forma rápida y sencilla sea capaz de gestionar los contenidos y a los usuarios que tendrán acceso al repositorio, adaptándose a los requisitos y requerimientos establecidos previamente por el cliente para el almacenamiento y centralización de la información. Para darle solución a esta problemática Jackrabbit propone tres modelos de despliegue, uno de ellos propone que el repositorio vaya embebido en la aplicación:

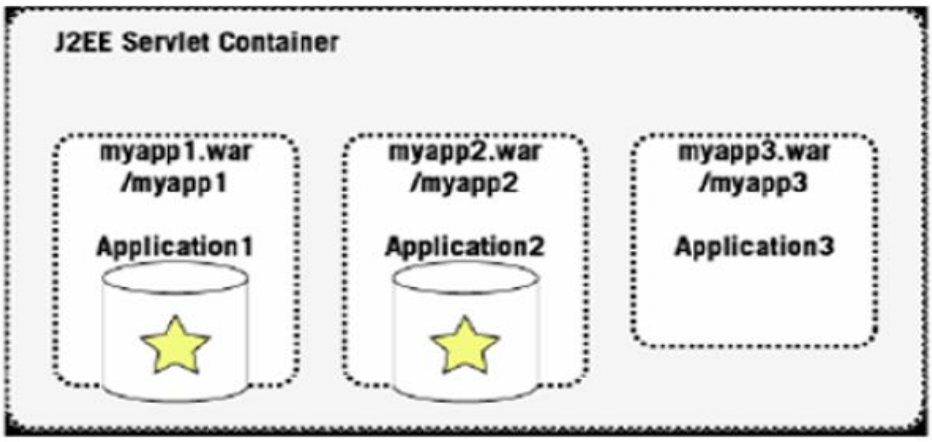



**Figura 1.5 Modelo de despliegue de Jackrabbit 1: Dentro de la aplicación**

En este modelo, el repositorio y la aplicación corren en la misma JVM. Sólo la aplicación tiene acceso a dicho repositorio. Esto es lo ideal para pequeñas aplicaciones porque no requieren de dependencias externas para ejecutarse.

**Metodología de desarrollo de Software.**

Las metodologías para el desarrollo de software son consideradas el "conjunto de actividades necesarias para transformar los requisitos de los usuarios en un sistema software"(José Ignacio Peláez Sánchez). Una metodología puede seguir uno o varios modelos de ciclo de vida. El ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo. La metodología indica cómo hay que obtener los distintos productos parciales y finales.

Existen varias generaciones de metodologías:

- Desarrollo Convencional (Sin Metodología).
- Desarrollo Estructurado.

- Desarrollo Orientado a Objetos.

Sin embargo para el desarrollo del sistema que será propuesto solamente se hará un estudio de las metodologías orientadas a objeto para seleccionar la que guiará el proceso de desarrollo del sistema a implementar.

La esencia del desarrollo orientado a objetos es la identificación y organización de conceptos del dominio de la aplicación y no tanto de su representación final en un lenguaje de programación. En esta metodología se eliminan fronteras entre fases debido a la naturaleza iterativa del desarrollo orientado al objeto. Aparece una nueva forma de concebir los lenguajes de programación y su uso al incorporarse bibliotecas de clases y otros componentes reutilizables. Hay un alto grado de iteración y solapamiento, lo que lleva a una forma de trabajo muy dinámica.

Entre los aspectos positivos de la metodología orientada a objetos podemos encontrar que son iterativas e incrementales, fácil de dividir el sistema en varios subsistemas independientes y se fomenta la reutilización. Algunas de estas metodologías se explican a continuación.

**Extreme Programing (XP)**

XP es una metodología de desarrollo ágil de SW, que puede ser usado por equipos pequeños y medianos. Para la construcción de aplicaciones con elevada calidad, con un mínimo gasto fijo, dentro de un presupuesto y calendario previsible. Con XP, son priorizados los resultados de trabajo inmediatos, haciendo el proceso de desarrollo más sencillo y aplicando el sentido común. También aumenta la productividad en el desarrollo, potenciando al máximo el trabajo en equipo.

Principios y prácticas de XP *(*Daniel Gomez y Elisabet Aranda*)*

- Retroalimentación a escala fina: Desarrollo guiado por pruebas. Juego de planificación. Cliente presente. Programación en pares.
- Proceso continuo en lugar de por lotes: Integración continua. Liberación pequeña.

- Entendimiento compartido: Diseño simple. Metáfora del sistema. Propiedad colectiva del código. Convenciones del código.
- Bienestar del programador: Paso sostenible.

Ciclo de vida de XP

El ciclo de vida de XP se enfoca en el carácter iterativo e incremental del desarrollo. Una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas; y en el caso de XP, corresponden a un conjunto de historias de usuarios. Las iteraciones son relativamente cortas, pues XP plantea que entre más rápido se le entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo.

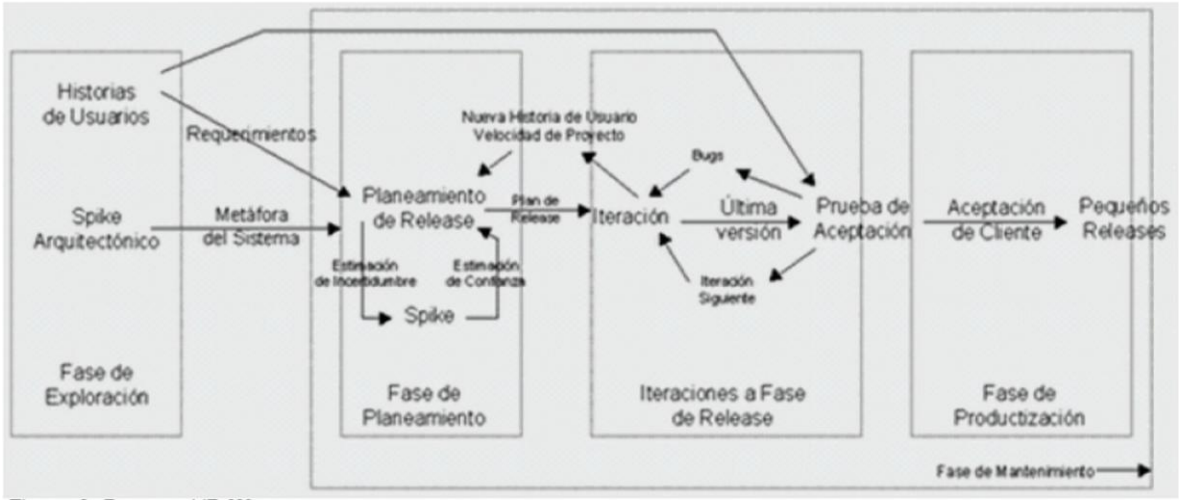



**Figura 1.6 Ciclo de la Metodología Extreme Programing**

Fases de XP (Patricio Letelier y Mª Carmen Penadés)

- **Exploración:** Los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. El equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

- **Planeación:** Se fija una fecha en la que estará terminado el grupo de historias más pequeño y más importante.
- **Iteraciones:** Esta fase incluye varias iteraciones del sistema antes de la primera entrega. El cliente decide que historias van a ser implementadas para cada iteración.
- **Producción:** En esta fase aparecen nuevos cambios y se tiene que decidir si serán incorporados o no en dicha entrega.
- **Mantenimiento:** En esta fase por lo general se necesita un esfuerzo extra de los programadores para satisfacer los requerimientos del cliente.
- **Muerte:** Se llega a esta fase cuando el cliente está satisfecho con el sistema, o cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

**Microsoft Solution Framework (MSF)**

MSF ofrece una guía sobre cómo organizar personas y proyectos para planificar, desarrollar y desplegar soluciones tecnológicas de software de una forma orientada al cambio. Constituye un disciplinado proceso ágil de desarrollo de SW, y un marco de trabajo extensible y adaptable.

Principios de MSF

- Asociación con los clientes.
- Trabajo hacia una visión compartida.
- Entrega de resultados incrementales.
- Inversión en la calidad.
- Fortalecimiento de los miembros del equipo.
- Establecimiento de responsabilidades claras.
- Aprendizaje de todas las experiencias.
- Promoción de la comunicación abierta.

- Mantenimiento ágil, adaptación al cambio.

Ciclo de vida de MSF

El proceso de MSF combina las mejores prácticas del desarrollo Cascada y Espiral, con puntos de fijación para cada iteración.

La integración sin problemas de MSF en Visual Studio Team System soporta el desarrollo iterativo con el aprendizaje continuo y el refinamiento. La definición del producto, el desarrollo, y las pruebas se producen en la superposición de iteraciones incrementales resultantes de la finalización del proyecto. Las iteraciones pequeñas permiten reducir el margen de error en sus estimaciones y proporcionar información rápida acerca de la exactitud de los planes de su proyecto. Cada iteración debería traducirse en una parte estable de todo el sistema.


**Figura 1.7 Ciclo de la Metodología MSF**

Fases de MSF (Gattaca.SA.)

- **Visión**: Obtener una visión del proyecto compartida, comunicada, entendida y alineada con los objetivos del negocio. Además, identificar los beneficios, requerimientos funcionales, sus alcances, restricciones; y los riesgos inherentes al proceso.
- **Planeación**: Obtener un cronograma de trabajo que cumpla con lo especificado en la fase de Visión dentro del presupuesto, tiempo y recursos acordados. Este cronograma debe identificar puntos de control específicos que permitan generar entregas funcionales y cortas en el tiempo.
- **Desarrollo**: Obtener iterativamente de la mano de la fase anterior y la siguiente, versiones del producto entregables y medibles que permitan de cara al cliente probar características nuevas sucesivamente. Esto incluye ajustes de cronograma necesarios.
- **Estabilización**: Obtener una versión final del producto probada, ajustada y aprobada en su totalidad.
- **Instalación**: Entregar (instalar) al cliente el producto finalizado en su totalidad. Como garantía se han superado con éxito las etapas anteriores.

**Rational Unified Process (RUP)**

RUP es un proceso de desarrollo de software y a la vez, es un conjunto de actividades dirigidas a transformar los requerimientos del cliente en un sistema de software. Aumenta la productividad del equipo de desarrollo, permitiendo a cada miembro compartir un lenguaje común como por ejemplo Unified Modeling Lenguaje (UML), un proceso y una vista de cómo revelar el software. RUP es un proceso configurable y es soportado por herramientas que automatizan partes grandes del proceso. Son usadas para crear y mantener los diversos artefactos de la ingeniería de software.

Principios fundamentales de RUP

- Adaptar el proceso.
- Equilibrar las prioridades de los interesados que están enfrentadas.
- Colaborar con los otros equipos.

- Demostrar el valor de forma iterativa.
- Elevar el nivel de abstracción.
- Centrarse continuamente en la calidad.

El ciclo de vida de RUP

La vida de un sistema transcurre a través de ciclos de desarrollo, desde que comienza hasta que termina, en cada ciclo se repite el proceso unificado de desarrollo. Cada uno consta de cuatro fases (Inicio, elaboración, construcción, transición) y concluye con una versión del producto. Cada fase se subdivide en iteraciones. Una iteración es una secuencia de actividades con un plan establecido y criterios de evaluación, cuyo resultado es una versión del software.

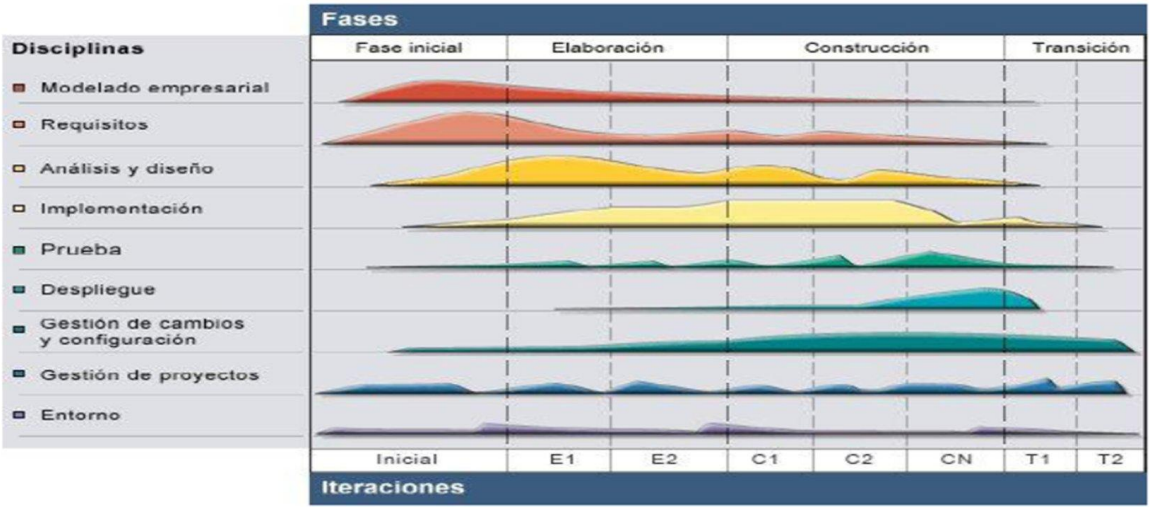



**Figura 1.8 Fases e interacciones de la Metodología RUP**

El ciclo de vida de RUP se caracteriza por ser:

- **Dirigido por casos de uso:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura.
- **Iterativo e Incremental:** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada mini proyecto es el resultado de una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto.

Fases de RUP

Una fase es un período de tiempo entre dos objetivos importantes. Al final de cada fase se lleva a cabo una valoración para determinar si los objetivos de la fase se han alcanzado. Una valoración satisfactoria permite que el proyecto continúe a la fase siguiente.

- **Inicio**: Establece un acuerdo entre todos los interesados respecto a los objetivos del ciclo de vida para el proyecto. La fase inicio es muy significativa fundamentalmente en los esfuerzos de desarrollo nuevos, pues son más arriesgados para los requisitos y para la actividad comercial. Deben abordarse antes de que el proyecto pueda continuar.
- **Elaboración**: Establece una línea base para la arquitectura del sistema para proporcionar una base estable para el grueso del diseño y del esfuerzo de implementación en la fase de construcción. La arquitectura evoluciona a partir de una consideración sobre los requisitos más significativos (los que tienen un gran impacto en la arquitectura del sistema) y una valoración de los riesgos. La estabilidad de la arquitectura se evalúa mediante uno o más prototipos arquitectónicos.
- **Construcción**: Se clarifican los requisitos restantes y se completa el desarrollo del sistema basándose en la arquitectura de línea base. La fase de construcción es, de alguna manera, un

proceso de fabricación, en el que se pone el énfasis en la gestión de los recursos y el control de las operaciones para optimizar los costes, la planificación y la calidad.

- **Transición**: Se garantiza que el software esté disponible para los usuarios. La fase de transición puede acarrear varias iteraciones e incluye las pruebas del producto en preparación para la liberación, así como ajustes menores basados en la información de retorno de los usuarios. En este momento del ciclo vital, la información de retorno de los usuarios debe centrarse especialmente en el ajuste del producto, las cuestiones de configuración, instalación y utilización.

Para el desarrollo de la aplicación se utilizará la metodología RUP. Dicha metodología está estructurada en fases o etapas de desarrollo donde se obtendrán cada uno de los artefactos. Además proporciona una guía para las actividades de un equipo de desarrollo, dirige las tareas de cada desarrollador por separado y del equipo en conjunto, especifica los productos que deben desarrollarse y ofrece criterios para el control, medición de los productos y actividades del proyecto.

Esta metodología de desarrollo al estar basada en una fuerte interacción con el cliente y usuarios, permite obtener productos adecuados a las necesidades reales, ahorrando esfuerzos y aumentando la satisfacción del usuario final.

**Lenguaje de modelado**

**UML**

Para el desarrollo de la aplicación se utilizará al UML, como el lenguaje con que se modelarán los artefactos que se creen en el proceso de desarrollo del software. Se elije este lenguaje de modelado ya que el mismo es el que se emplea asociado a la metodología de desarrollo que se seleccionó (RUP).

UML es un lenguaje de construcción de modelos para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, con los que se construyen mayormente sistemas orientados a objetos.

Entrega una forma de modelar elementos conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reutilizables.

Es una especificación de notación orientada a objetos. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representan la arquitectura del proyecto.

También intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

UML es ahora un estándar, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otra rama.

**Herramienta CASE**

Las herramientas CASE son la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas para comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación. Estas representan una forma que permite Modelar los Procesos de Negocios de las empresas y desarrollar los Sistemas de Información Gerenciales. Para seleccionar la herramienta CASE que se empleará en el modelado de los artefactos se tendrá en cuenta aquellos programas que son más populares para el modelado en UML, que fue el lenguaje seleccionado, y además se eligieron aquellos programas que se encontraban bajo licencias libres, siendo posible su libre uso, estudio y modificación.

**ArgoUML**

"ArgoUML es una aplicación de diagramado de UML escrita en Java y publicada bajo la Licencia BSD Open Source. Dado que es una aplicación Java, está disponible en cualquier plataforma soportada por Java." Es la principal herramienta de fuente abierta para el modelado UML e incluye soporte para todos los estándares de diagramas UML. Funciona en cualquier plataforma Java y está disponible en diez idiomas. Fue instalado más de medio millón de veces en todo el mundo durante el 2005 y está en uso actualmente en muchos lugares. "Sin embargo, desde la versión 0.20, ArgoUML está incompleto. No es conforme completamente a los estándares UML y carece de soporte completo para algunos tipos de diagramas incluyendo los Diagramas de secuencia y los de colaboración." Construido en diseños críticos suministra una revisión no obstructiva del diseño y sugerencias para mejoras y su Interfaz de módulos es extensible. Además posee restricciones OCL (Object Constraint Language) para Clases y sirve para realizar una ingeniería inversa.

**BOUML**

BOUML es una herramienta libre para el modelado UML que permite especificar y generar código en C++, Java, Interface Definition Language (IDL), Hypertext Preprocessor (PHP) y Python. BoUmL es una herramienta de software libre. Pude ser redistribuida o modificada bajo los términos de Licencia Pública General (GNU). Se ejecuta bajo Unix/Linux/Solaris, MacsOS y Windows. BOUML es una herramienta rápida y no requiere mucha memoria para gestionar varios miles de clases. Es extensible y las herramientas externas llamadas plug-outs pueden ser escritas en C++ o Java.

BOUML se distribuye con la esperanza de que sea útil, pero sin garantía alguna; incluso sin la garantía implícita de comercialidad o aptitud para un propósito en particular.

Es la forma más fácil de desarrollar un proyecto que contenga un gran número de clases y que tengan la misma definición. Sólo BOUML y Enterpise Architect son capaces de invertir todas las fuentes de Java, el resto de las herramientas no disponen de suficiente memoria.

**Umbrello UML Modeller**

Umbrello UML Modeller es un Lenguaje Unificado de Modelado de diagramas de programas para KDE aunque funciona en otros entornos de escritorio. Herramienta libre que ayuda a crear y editar diagramas en el proceso de desarrollo de software. “Umbrello maneja gran parte de los diagramas estándar UML pudiendo crearlos, además de manualmente, importándolos a partir de código en C++, Java, Python, IDL, Pascal/Delphi, Ada, o también Perl (haciendo uso de una aplicación externa).

Así mismo, permite crear un diagrama y generar el código automáticamente en los lenguajes antes citados, entre otros. El formato de fichero que utiliza está basado en XML. También permite la distribución de los modelos exportándolos en los formatos DocBook y XHTML, lo que facilita a los proyectos colaborativos donde los desarrolladores no tienen acceso directo a Umbrello o donde los modelos van a ser publicados vía Web.”

**Visual Paradigm**

Visual Paradigm para UML es un galardonado producto que facilita a las organizaciones el diseño visual de los distintos diagramas. Esta herramienta de desarrollo de software ayuda a los equipos de desarrollo en la confección de los distintos modelos que van desde la construcción hasta el despliegue, aumentando al máximo la productividad.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML, además es Orientado a Objetos.

Visual Paradigm soporta un conjunto de lenguajes como son, tanto en la generación de código como en la ingeniería inversa. Puede generar código Java a partir de los modelos y viceversa. Cualquiera de los

cambios que se realicen en el código existente puede reflejarse en el modelo. Aunque es una herramienta gratuita, este programa se encuentra bajo licencias que no permiten el estudio y modificación de la misma.

Se empleará además para especificar y construir los diversos artefactos, la herramienta Visual Paradigm, ya que el producto terminado debe ser entregado al usuario con toda la documentación y el código fuente. De esta manera no se impone ninguna traba a la futura extensión del mismo, asegurando un trato justo, claro y transparente.

**Conclusiones del Capítulo.**

Para el desarrollo de este capítulo se llevó a cabo un profundo estudio sobre los elementos más importantes para la fundamentación teórica del tema, de esta manera se explicaron los términos básicos que hacen comprensible el tema, además de que se expusieron las normas, estándares , herramientas, técnicas y metodologías que permitirán el desarrollo de la aplicación. De esta manera se tomó la decisión de emplear:

- Como lenguaje de programación Java

- Como plataforma de desarrollo : J2EE
- Metodología de desarrollo: RUP
- Lenguaje de modelado: UML
- Herramienta CASE: Visual Paradigm

# 2. Procesos de Negocio

En el actual capítulo se describirán brevemente las funciones principales y el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis crítico de cómo se ejecutan actualmente dichos procesos. Se abordará también sobre las características que deberá tener el sistema para gestionar la creación de nuevos repositorios de contenidos para el Diccionario Enciclopédico “Libertad”.

De manera general se define la visión, los objetivos y el alcance del sistema, tanto funcional como técnico. Primeramente se brinda una vista general de los procesos que existen en el negocio, lo que permite comprender a qué se dedica el mismo, así como establecer una comunicación entre las partes involucradas en él; para ello en el contexto del capítulo se incluye un artefacto primordial de esta fase, el Modelo de Dominio.

Una vez llegado a ese punto se comienza a definir qué es lo que debe hacer el sistema, siendo así necesario enfocarse en la captura de requisitos. Esta es la razón por la cual se definen aquí los requisitos funcionales y no funcionales de la aplicación, los actores que interactúan con la misma y las relaciones que pudieran establecerse entre ellos, obteniendo de esta manera el diagrama de casos de uso del sistema, así como la descripción textual de cada uno de los requisitos identificados, los cuales son incluidos también en este documento.

**Objetivos estratégicos de la organización**

En la nueva era de la “Sociedad de la Información y del Conocimiento" que se vive hoy, uno de los problemas fundamentales a los que se enfrentan las diferentes instituciones y empresas es el de la conservación de la información generada. La recopilación de la información en medios poco legibles y ya obsoletos explica la necesidad de las empresas de almacenar de forma centralizada la información generada durante los procesos empresariales y han motivado a todos a la búsqueda e implementación de nuevas estrategias y mecanismos.

Para lograr una mayor organización de la información generada y para asegurar el contenido expuesto, existen los repositorios de contenidos los cuales brindan la posibilidad de tener expuesto razonablemente los datos recopilados y de manera centralizada.

En las empresas, los repositorios digitales abarcan desde los discos duros de almacenamiento de los documentos que se reciben o se generan, hasta las aplicaciones corporativas: ERPs (Planificación de los Recursos Empresariales), CRMs (Gestión de las Relaciones con los Clientes), gestores documentales, etc. Gestionar con eficiencia estos contenidos se convierte en un tema crucial para la propia supervivencia. Sus necesidades se centran en la localización rápida de los documentos que se necesitan para una auditoria o para la firma de un contrato, en el seguimiento comercial de las ofertas o en determinados resultados. Es necesario resaltar la importancia de la confidencialidad y, en consecuencia, la necesaria gestión de los accesos.

Así pues, los repositorios se diseñan como espacios de distribución y de intercambio de contenidos digitales reutilizables. De esta manera, los usuarios realizan aportaciones de interés al mismo tiempo que pueden buscar dentro del repositorio otros recursos de acuerdo a las condiciones previstas en cada caso.

Durante la concepción y conceptualización del proyecto Diccionario Enciclopédico “Libertad”, se determinó la necesidad de almacenar determinados recursos internos e inherentes a la actividad de manera centralizada, así como la de proveer servicios de información para compartir, divulgar, acceder, conocer y representar la información. Para darle solución a esto se propone como estrategia principal la implementación de una herramienta o sistema informático de la cual se obtendrían los siguientes beneficios:

1. Identificar claramente los perfiles de las personas que pueden acceder a determinados contenidos.
2. Simplificar los procesos, con la reducción de costes que conlleva

**Objeto de automatización**

Ante la petición de un cliente de crear un sistema para gestionar sus procesos empresariales siempre es necesario hacer un estudio de los mismos para determinar cuáles de estos, o qué porción de estos pueden ser automatizados. En este caso, para la modelación del sistema propuesto y de acuerdo al caso de estudio y los objetivos trazados así como las tareas se identificó la necesidad de automatizar:

**La creación del repositorio de contenidos**: lo cual dará la posibilidad de crear el repositorio de contenidos y más que ello configurar y mantener determinadas características del mismo, de manera que se encuentre accesible en todo momento para cualquier usuario que necesite acceder a la información contenida en el mismo, garantizando así la disponibilidad de la información.

**La gestión de los usuarios que tendrán acceso al repositorio**: lo cual permitirá a un administrador del sistema crear nuevos usuarios, así como asignarle a los mismos determinados permisos sobre los distintos recursos, lo cual asegure en gran medida la integridad de la información en el ciclo de uso de la misma dentro de la institución.

**La gestión de los contenidos pertenecientes al repositorio**: De esta manera se gestionará todo lo referente al contenido almacenado en el repositorio, desde la creación de los mismos hasta su eliminación o conservación permanente, incluyendo las búsquedas, consultas entre otras actividades.

De esta manera, una vez automatizados los puntos antes citados el cliente podrá sin dudas beneficiarse de todos las ventajas que tienen como subproducto los repositorios de contenidos.

**Información que se maneja.**

Un repositorio de contenidos en sí, almacena cualquier tipo de información digital, e incluso, pudieran hacerse referencias a información en soporte no digital. Específicamente el sistema del que se beneficiará el proyecto Diccionario Enciclopédico “Libertad” manejará dos tipos fundamentales: las plantillas y los activos digitales.

**Plantilla digital:** Permite definir el prototipo de un nuevo objeto o tipo de contenido, el cual puede tener indistintamente determinadas propiedades, características o funciones, siendo así la base para poder

hacer la descripción de un objeto. Puede ser vista como un contenedor para la recolección de determinado tipo de información asociado a un objeto, así como lo hacen los formularios en determinadas aplicaciones.

**Activo digital**: Es una instancia específica de una plantilla digital. No es más que una combinación entre una plantilla digital que define el prototipo del objeto y la información en sí, determinada por el prototipo definido en la plantilla asociada. Puede ser visto como el conjunto de datos o información en sí que se recoge para un formulario determinado para una instancia específica.

Estos conceptos de plantillas y activo digital pueden ser asociados a los conceptos de "clase" y "objeto" conocidos de la programación orientada a objeto, donde una plantilla se corresponde con una "clase" como prototipo para crear objetos específicos, y los activos se corresponden con los "objetos", como instancias de una plantilla o clase determinada.

**Modelo de dominio**

El tipo de artefacto más interesante utilizado en el Proceso Unificado es el modelo. Cada trabajador necesita una perspectiva diferente del sistema. Cuando se decide desarrollar un Proceso Unificado se identifican todos los trabajadores y cada una de las perspectivas que posiblemente podrían necesitar. Las perspectivas recogidas de todos los trabajadores se estructuran en unidades más grandes, es decir, modelos, de modo que cada trabajador puede tomar una perspectiva concreta del conjunto de modelos.

La construcción de un sistema es por tanto un proceso de construcción de modelos, utilizando distintos modelos para describir todas las perspectivas diferentes del sistema. La elección de los modelos para un sistema es una de las decisiones más importantes del equipo de desarrollo.

**¿Qué es un modelo?**

Un modelo es una abstracción del sistema, especificando el sistema modelado desde un cierto punto de vista y en un determinado nivel de abstracción. Un punto de vista es, por ejemplo, una vista de especificación o una vista de diseño del sistema. Los modelos son abstracciones del sistema que construyen los arquitectos y desarrolladores.

Un modelo puede ser visto además como una abstracción semánticamente cerrada del sistema. Es una vista auto-contenida en el sentido de que un usuario de un modelo no necesita para interpretarlo más información de otros modelos. Además del sistema, un modelo debe describir las interacciones entre el sistema y los que lo rodean, por tanto, aparte del sistema que se está modelando, el modelo también debería incluir elementos que describan partes relevantes de su entorno.

**Comprensión del contexto del sistema: Modelo de dominio**

El modelado del negocio es la primera fase de RUP que se encarga de hacer un estudio detallado de cada uno de los procesos y estados del negocio que se quiere automatizar, con el objetivo de comprender la estructura y dinámica de la organización, así como entender los problemas actuales e identificar las mejoras que se le puedan hacer al mismo.

Para lograr estos propósitos se debe obtener una visión que permita definir los procesos, roles y responsabilidades de la organización en los modelos de casos de uso. Es por ello que la primera iteración de la fase “Modelado de negocio” incluye la evaluación de la organización en la cual será implantada el sistema para tomar así las decisiones más correctas y eficientes sobre cómo se desarrollará este proceso.

Debido que el negocio actual presenta un bajo nivel de estructuración, sin poder identificarse las personas que desarrollan las distintas actividades en cada uno de los procesos, se decide realizar el modelo de dominio que RUP propone para estos casos.

El Modelo de Dominio o Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real, más significativos para un problema o área de interés. Se realiza si no se logra determinar el proceso de negocio con fronteras bien establecidas donde se logra ver claramente quiénes son las personas que lo inician. Los objetos del dominio representan las “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema.

El objetivo fundamental de este modelo es contribuir a la comprensión del contexto del sistema, y por lo tanto contribuir también a la comprensión de los requisitos del sistema que se desprenden de este

contexto. En otras palabras, el modelado del dominio debería contribuir a una compresión del problema que se supone que el sistema resuelve en relación a su contexto.

Representa clases u objetos conceptuales del problema, que vienen a ser las ideas u objetos físicos y el/los enlaces de unos objetos con otros, ayudando de esta forma a la elaboración del glosario de términos, facilitando la comunicación entre los desarrolladores del sistema y un mayor entendimiento del contexto en que se desarrolla el mismo, producto del empleo de un vocabulario común, lo cual es de gran importancia para poder compartir el conocimiento adquirido con otros.

**Representación del modelo de dominio para el sistema propuesto**

En el diagrama siguiente se puede ver una representación de los objetos o conceptos fundamentales que se derivan del contexto del negocio. En dicho diagrama UML se indica que Nodo y Propiedad son sub-interfaces de Elemento, en otras palabras, todo es un elemento, solo que en determinados casos pudiera ser un Nodo y en otros una Propiedad. Una propiedad tiene uno y un sólo nodo padre, por su parte un nodo puede no tener padres (en este caso solo si se refiere al nodo raíz) o un solo padre, y además de eso un número cualquiera de Elementos hijos, o sea, propiedades u otros elementos.

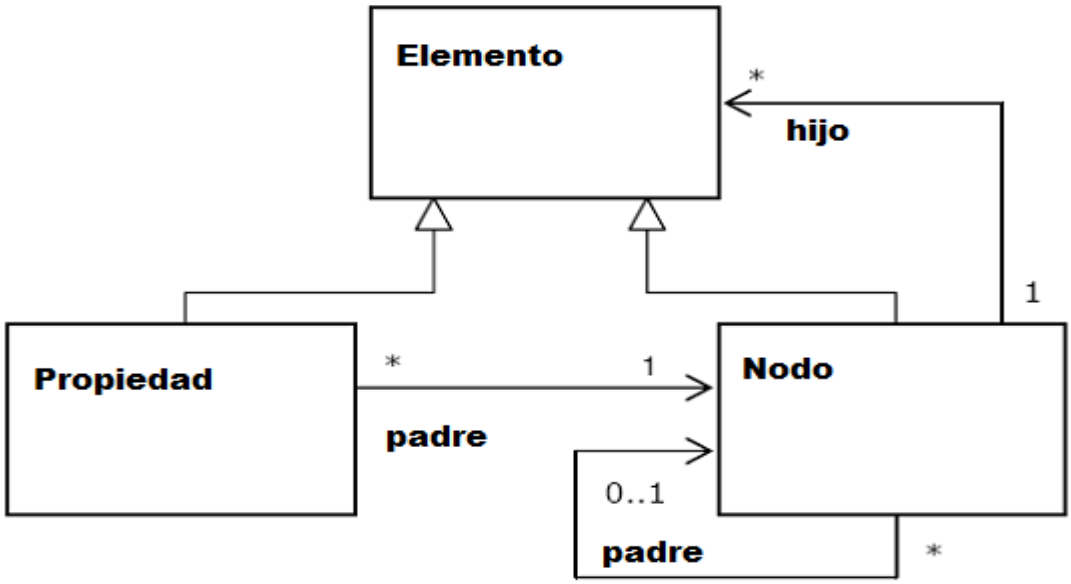



**Figura 2.1 Modelo de Dominio**

**Especificación de los requisitos de software.**

Los requerimientos de un software son condiciones o capacidades que tienen que ser alcanzadas o poseídas por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

Definen qué es lo que el software debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se establecen. Es una característica que el sistema debe tener para cubrir alguna de las peticiones de los clientes que lo motivan para resolver algún problema o lograr algún objetivo.

**Requisitos funcionales**

Los requisitos funcionales son capacidades o condiciones con las cual el sistema debe cumplir e indican su comportamiento, por lo que se debe analizar cuáles son las funcionalidades del sistema que cumplan

con los objetivos que se plantearon, enumerando para ello las acciones que la aplicación debe ser capaz de realizar.

- R1: Autenticar usuario
- R2: Gestionar contenidos
    - Crear contenidos
    - Eliminar contenidos
    - Leer contenidos
- Gestionar usuarios.
    - Crear usuarios
    - Modificar usuarios
    - Eliminar usuarios
    - Buscar usuarios

**Requisitos no funcionales**

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades son las características que hacen del software atractivo, usable, rápido o confiable. En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requerimientos funcionales, es decir una vez se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

Son fundamentales en el éxito del producto pues marcan la diferencia entre un producto bien aceptado y otro con poca aceptación. Existen numerosas categorías para clasificar los requisitos no funcionales. A continuación se muestran los requisitos que la aplicación en cuestión debe tener.

**Requisitos de software**

- El sistema se implementará con tecnología Java, por lo cual se requiere de la máquina virtual de Java a partir de su versión 1.5 en cada estación cliente.

**Requisitos de hardware**

- Debido a que para la implementación del sistema se usarán tecnologías Java, la cual se abstrae totalmente al software y hardware, y que además de eso, no es una aplicación de gran procesamiento de información, no existe ningún requisito de este tipo.

**Restricciones en el diseño e implementación**

- El lenguaje a ser usado para implementar el sistema será Java.
- Usar estándar de codificación Java.

**Requisitos de apariencia o interfaz**

- La interfaz externa de la aplicación debe ser amigable, sencilla y fácil de usar por los usuarios finales, facilitando el control de las operaciones sin necesidad de mucho entrenamiento para su uso.
- La aplicación estará estructurada de forma clara y comprensible, al mismo tiempo que permitirá la interpretación correcta e inequívoca de la información.
- El diseño responderá a la ejecución de acciones de una manera rápida, minimizando los pasos a dar en cada proceso.
- Todos los textos y mensajes de la aplicación aparecerán en idioma español.

**Requisitos de seguridad**

- La aplicación cuenta con un módulo para gestionar los usuarios y más allá de eso las operaciones que puede realizar cada uno dentro del repositorio.

**Requisitos de usabilidad**

- Su funcionamiento será intuitivo y requerirá de conocimientos mínimos para su uso.
- El sistema será flexible y fácil de usar.
- El sistema contará con combinaciones de teclas con las cuales se podrán activar de manera rápida cada una de las funcionalidades de la aplicación.
- El sistema responderá de manera rápida a cada una de las peticiones del usuario.

**Requisitos de rendimiento.**

- El sistema estará poco cargado, garantizando que las respuestas a las peticiones de los usuarios sea rápida, así como el procesamiento de la información en general.

**Requisitos de portabilidad**

- El sistema será independiente y multiplataforma, lo cual significa que el mismo podrá ser instalado y usado en cualquier sistema operativo actual.

**Requisitos legales**

- Las herramientas seleccionadas para el desarrollo del producto están respaldadas por licencias libres, bajo condiciones de software libre.

**Definición de los casos de usos.**

A continuación se irán mostrando los artefactos que se han creado durante esta primera fase, la cual servirá de base para la futura implementación del sistema.

**Definición de los actores**

En el sistema que se está modelando actúan dos actores denominados “Usuario” y “Administrador del Sistema”, que son los que interactúan con el sistema.

| Actores | Justificación |
|---|---|
| Usuario | Es la persona que se encarga de crear, eliminar o leer el/los contenidos en el repositorio. |
| Administrador del Sistema | Es la persona que se encarga de crear, eliminar, modificar o buscar los usuarios que accederán al repositorio además de cumplir con las mismas funciones que un usuario. |

**Listado de los casos de uso**

| CU-1 | Autenticar usuario |
|---|---|
| Actor | Usuario |
| Descripción | Permite que el usuario se autentifique para acceder al sistema. |
| Referencia | R1 |

| CU-2 | Gestionar contenidos |
|---|---|
| Actor | Usuario |
| Descripción | Permite que el usuario cree nuevos |

| | |
|---|---|
| | contenidos, los elimine o los puede leer desde el repositorio. |
| Referencia | R2.1,R2.2,R2.3 |

| CU-3 | Gestionar usuarios |
|---|---|
| Actor | Administrador del Sistema |
| Descripción | Permite que el administrador del sistema cree nuevos usuarios, los elimine, modifique sus permisos para acceder al repositorio y busque aspectos del mismo. |
| Referencia | R3.1,R3.2,R3.3,R3.4 |

**Diagrama de Casos de Uso.**

A continuación se presenta el diagrama de casos de uso, en el cual como se puede apreciar, se identifican los usuarios (Usuario y Administrador) previamente citados, donde el administrador del sistema también pudiera interactuar con el mismo similar a como lo hace un Usuario. Se aprecia además que cualquier usuario antes de realizar cualquier actividad con el sistema lo primero que debe hacer es autenticarse, de esta manera los casos de uso “Gestionar Usuarios “y “Gestionar Contenido” incluyen al caso de uso Autenticar usuario.

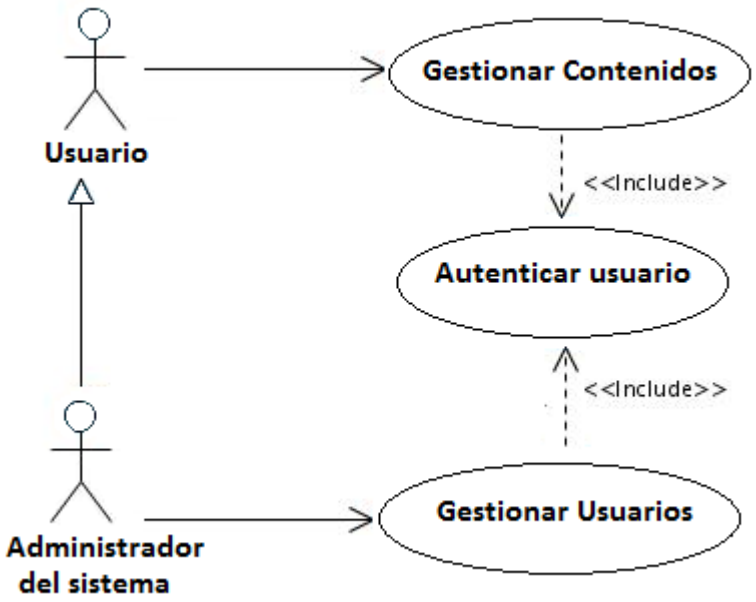



**Figura 2.2 Diagrama de Casos de Uso del Sistema**

**Descripción expandida de los casos de uso.**

| **Caso de Uso** | |
|---|---|
| CU – 1 | Autenticar Usuario |
| **Propósito** | Permitir que el usuario se autentifique en el sistema. |
| **Actores**: Usuario | |
| **Resumen**: El caso de uso se inicia cuando el usuario inicia la aplicación para realizar alguna de las operaciones permitidas sobre el repositorio. | |

| | |
|---|---|
| **Referencias** | |
| **Precondición** | |
| **Acción del actor** | **Acción del sistema** |
| 1- El usuario inicia la aplicación | 2- El sistema muestra un cuadro de diálogo, pidiendo que se ingrese el identificador de usuario y la clave del mismo. |
| 3- El usuario ingresa un valor para el identificador y otro para la contraseña. | 4- El sistema verifica si existe algún usuario con ese identificador y que le corresponda la clave introducida por el |
| | 5- El sistema le da acceso al usuario al contenido existente en el repositorio., así como a las operaciones que puede realizar |
| **Flujo Alterno 4** | |
| | 4.1 El sistema muestra un error al usuario, para que rectifique el identificador de usuario y la clave introducida. |

| **Caso de Uso** | |
|---|---|
| CU – 2 | Gestionar Contenido |
| **Propósito** | Permitir que el usuario realice determinadas operaciones sobre el contenido existente en el repositorio. |
| **Actores**: Usuario | |

| | |
|---|---|
| **Resumen**: El caso de uso se inicia cuando el usuario se autentica en el sistema y desea realizar alguna de las operaciones permitidas sobre los contenidos. En la interfaz se muestran los contenidos a los cuales el usuario tiene acceso. | |
| **Referencias** | R2.1, R2.2, R2.3 |
| **Precondición** | El usuario está autenticado en el sistema. |
| **Acción del actor** | **Acción del sistema** |
| 1- El usuario selecciona la opción:<br>❖ Adicionar un nuevo contenido en el repositorio. Sección "Adicionar Contenido".<br>❖ Eliminar un contenido del repositorio. Sección "Eliminar Contenido".<br>❖ Buscar un contenido. Sección "Buscar | |
| **Sección "Adicionar Contenido"** | |
| | 1- El sistema muestra un cuadro de diálogo de selección de un archivo determinado en el sistema. |
| 2- El usuario selecciona el archivo que desea adicionar. | 3- El sistema verifica si no existe ningún contenido con el mismo nombre en el repositorio. |
| | 4- El sistema adiciona el contenido al repositorio y muestra un mensaje informando el éxito de la operación. |
| **Flujo Alterno 3** | |

| | |
|---|---|
| | 3.1 El sistema muestra un cuadro de diálogo anunciando el error cometido. |
| **Sección" Eliminar Contenido"** | |
| | 1- El sistema muestra un cuadro de diálogo para que el usuario indique qué contenido desea eliminar. |
| 2- El usuario indica qué contenido desea eliminar. | 3- El sistema verifica si existe el contenido. |
| | 4- El sistema elimina el contenido del repositorio y muestra un mensaje informando el éxito de la operación. |
| **Flujo Alterno 3** | |
| | 3.1 El sistema muestra un cuadro de diálogo anunciando el error cometido. |
| **Sección "Buscar Contenido"** | |
| | 1- El sistema muestra un panel para que el usuario indique el nombre del elemento a buscar. |
| 2- El usuario indica el nombre del contenido a buscar. | 3- El sistema verifica si el contenido existe. |
| | 4- El sistema devuelve un listado con el conjunto de contenidos que tienen el nombre especificado por el usuario. |
| **Flujo Alterno 3** | |
| | 3.1 El sistema muestra un cuadro de diálogo indicando que no existe ningún contenido en el repositorio con ese nombre. |

| **Caso de Uso** | |
|---|---|
| CU – 3 | Gestionar Usuario |
| **Propósito** | Permitir que el administrador realice determinadas operaciones con los usuarios del sistema. |
| **Actores**: Administrador del sistema | |
| **Resumen**: El caso de uso se inicia cuando el administrador se autentica en el sistema y desea realizar alguna de las operaciones permitidas sobre los usuarios. | |
| **Referencias** | R2.1, R2.2, R2.3 |
| **Precondición** | El usuario está autenticado en el sistema. |
| **Acción del actor** | **Acción del sistema** |
| 1- El usuario selecciona la opción:<br>❖ Adicionar un nuevo usuario en el sistema. Sección “Adicionar Usuario”.<br>❖ Eliminar un usuario del sistema. Sección “Eliminar Contenido”. | |
| **Sección “Adicionar Usuario”** | |
| | 1- El sistema muestra un cuadro de diálogo donde se le pide que entre los datos del nuevo usuario. |
| 2- El administrador introduce los datos del nuevo usuario en el cuadro de diálogo. | 3- El sistema verifica si no existe ningún usuario con el mismo identificador registrado en el mismo. |

| | |
|---|---|
| | 4- El sistema adiciona el usuario y muestra un mensaje informando el éxito de la operación. |
| **Flujo Alterno 3** | |
| | 3.1 El sistema muestra un cuadro de diálogo anunciando el error cometido. |
| **Sección" Eliminar Usuario"** | |
| | 1- El sistema muestra un cuadro de diálogo para que el usuario indique qué usuario desea eliminar. |
| 2- El usuario indica el usuario que desea o debe eliminar. | 3- El sistema verifica si existe el contenido. |
| | 4- El sistema elimina al usuario y muestra un mensaje informando el éxito de la operación. |
| **Flujo Alterno 3** | |
| | 3.1 El sistema muestra un cuadro de diálogo anunciando el error cometido. |
| **Sección "Buscar Usuario"** | |
| | 1- El sistema muestra un panel para que el administrador indique el nombre del usuario a buscar. |
| 2- El administrador indica el nombre del usuario a buscar. | 3- El sistema verifica si el contenido existe. |
| | 4- El sistema un cuadro de diálogo con los datos fundamentales del usuario buscado. |
| **Flujo Alterno 3** | |

| | |
|---|---|
| | 3.1 El sistema muestra un cuadro de diálogo indicando que no existe ningún usuario con ese nombre. |

**Conclusiones.**

En este capítulo se analizaron las características principales del sistema para la creación de nuevos repositorios de contenido para el Diccionario Enciclopédico “Libertad”. Se definió el modelo conceptual o de dominio para comprender la estructura y la dinámica de los procesos involucrados; se plantearon además los requisitos funcionales y no funcionales del sistema, los actores y finalmente la descripción textual de cada uno de los casos de uso del sistema.

Luego de haber concluido el mismo se han obtenido los resultados necesarios para proseguir con la próxima fase dentro del ciclo de vida del sistema. A través de los Modelos y diagramas expuestos en él se ha podido definir el ámbito del sistema, así como hacer una propuesta de las interfaces de usuario enfocadas 100% al usuario final, una base para la estimación del tiempo y recursos necesarios para proseguir con el proyecto y finalmente ha permitido sentar las bases para profundizar en los casos de uso detallándolos de manera que permitan reflejar la vista interna del sistema con el lenguaje de los desarrolladores.

# 3. Diseño del Sistema

En el actual capítulo se modelará el sistema y se encontrará su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Para el cumplimiento de dicha tarea se expondrán los artefactos y diagramas de clases de diseño, así como la realización de los casos de uso correspondientes al flujo de trabajo de diseño.

Los principales objetivos que se propone alcanzar con el desarrollo de este capítulo son:

. Adquirir una comprensión detallada de los requisitos no funcionales y las restricciones relacionadas con el lenguaje de programación, componentes reutilizables, sistemas operativos y tecnologías de interfaz de usuarios, entre otras.

.Crear una entrada apropiada y un punto de partida para las actividades de implementación capturando los requisitos, interfaces y clases necesarias.

.Descomponer los trabajos de implementación en partes más manejables que puedan ser manejadas por diferentes personas.

**Diseño del sistema**

El modelo de diseño es utilizado para modelar los aspectos dinámicos del sistema. Consta de un conjunto de objetos que describen las realizaciones de los casos de uso y sus relaciones, incluyendo además los mensajes que pueden enviarse entre ellos. Se centra en los impactos que producen en el sistema los requisitos funcionales y no funcionales.

De manera general el modelo de diseño constituye una abstracción al modelo de implementación y del código fuente, o sea, es la entrada principal o el punto de partida para las posteriores actividades de implementación del sistema que se desea desarrollar.

El mismo contiene básicamente los paquetes y/o subsistemas de diseño representados en una

breve jerarquía, los diagramas de clases de diseño y los de interacción(secuencia y/o colaboración), conocidos también como las realizaciones de los casos de uso, y finalmente las clases, interfaces y relaciones entre ellas contenidas en los paquetes.

**Realización de los casos de uso**

Las realizaciones de los casos de uso son las colaboraciones en el modelo de diseño que describen cómo se realizará uno o varios casos de uso específicos y cómo se ejecutan en término de casos de uso del diseño, en el cual intervienen el diagrama de clases y los diagramas de interacción.

Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento. Normalmente, un diagrama de interacción capta el comportamiento de un solo caso de uso. El diagrama muestra cierto número de ejemplos de objetos y los mensajes que se pasan entre estos objetos dentro del caso de uso. Existen dos tipos de diagramas de interacción que son: diagramas de secuencia y diagramas de colaboración. (Mairena, 2009)

En este capítulo se mostrarán los diagramas de interacción que modelan o responden a las funcionalidades que brindará el sistema.

Estos diagramas capturan las colaboraciones entre objetos que realizan el caso de uso, además muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. El diagrama de interacción, representa la forma en cómo un Cliente (Actor) u Objetos (Clases) se comunican entre sí en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente. Dicho diagrama puede ser obtenido de dos partes, desde el Diagrama Estático de Clases o el de Casos de Uso, en este caso se obtuvo del este último. (Gónzalez, J, 2009)

El diagrama de interacción, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos

objetos. El tiempo fluye de arriba abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren. (Ferré, X y Sánchez, M. 2005)

Los principales objetivos que se quieren lograr con la presentación de estos diagramas son:

. Dar una visión bien clara del flujo de control en el contexto en el que se desarrollan.

. Permitir mejor identificación de los objetos.

. Permitir observar adecuadamente la interacción de un objeto con respecto a los demás.

A continuación se muestran los diagramas de interacción

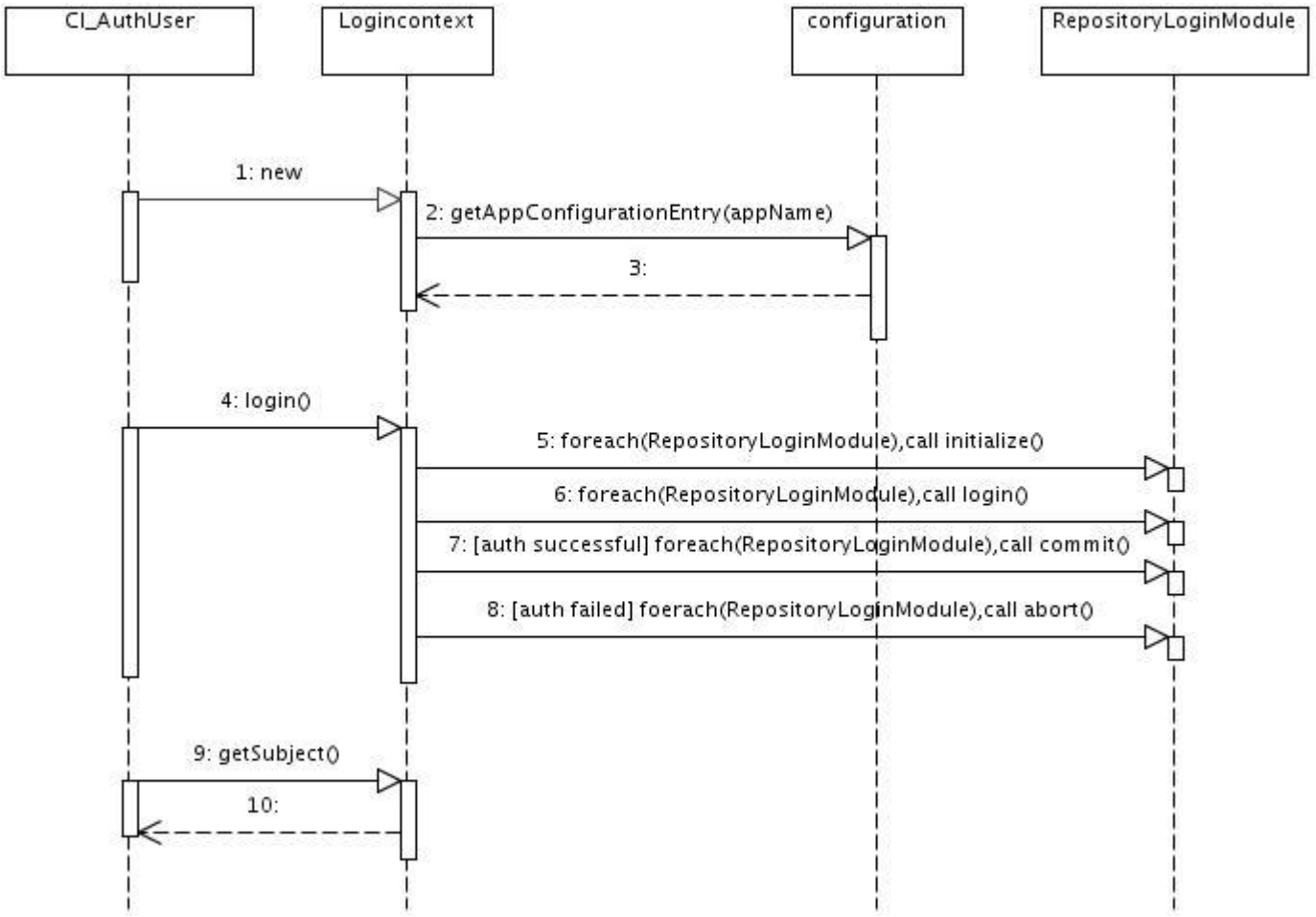



**Figura 3.1 Diagrama de interacción Caso de Uso AuthUser**

El diagrama de la figura 3.1 muestra como el sistema responde a la funcionalidad de autenticarse cada usuario para poder acceder al sistema.

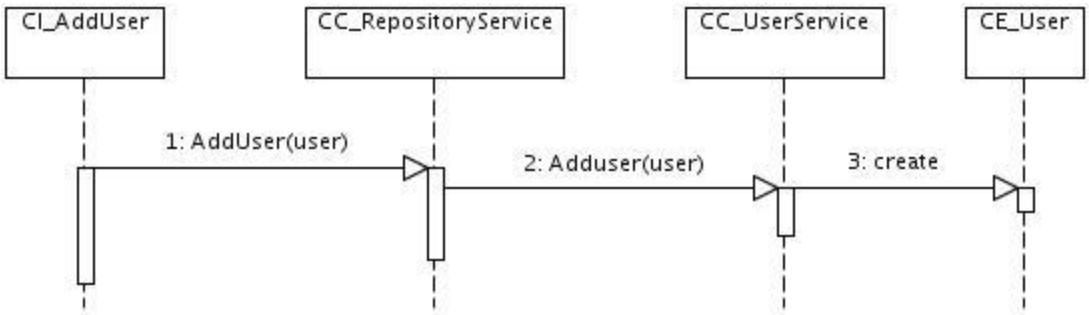



**Figura 3.2 Diagrama de interacción Caso de Uso AddUser**

El diagrama de la figura 3.2 muestra como el sistema responde a la funcionalidad de adicionar al repositorio los usuarios que podrán tener acceso a él.

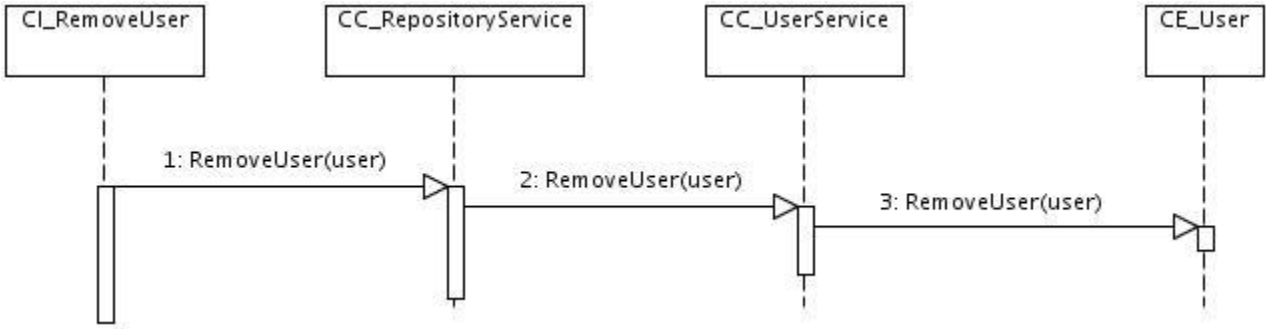



**Figura 3.3 Diagrama de interacción Caso de Uso RemoveUser**

El diagrama de la figura 3.3 muestra como el sistema responde a la funcionalidad de eliminar a un usuario del repositorio.

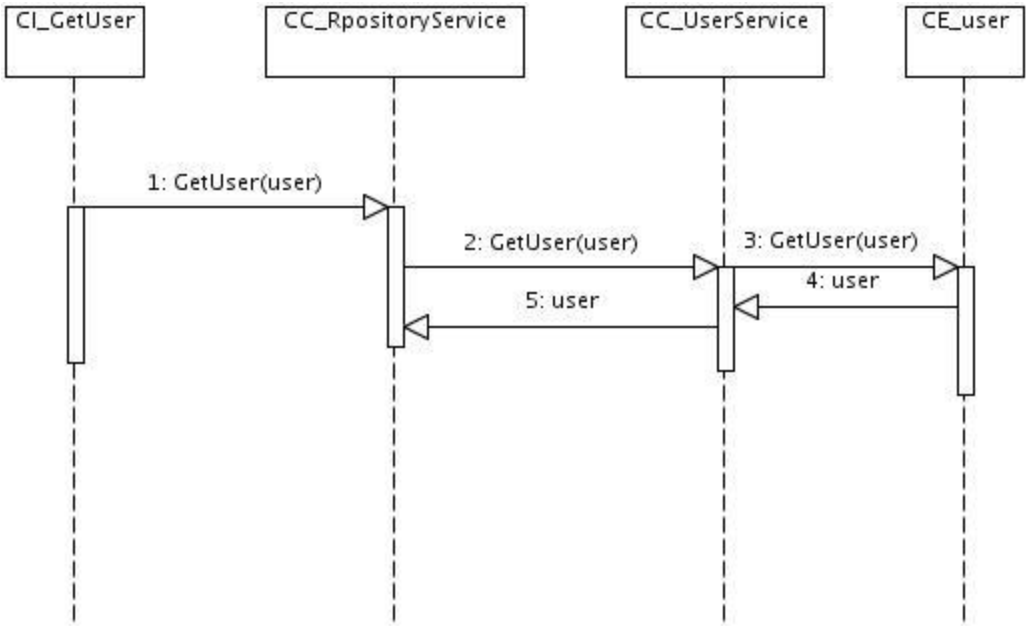



**Figura 3.4 Diagrama de interacción Caso de Uso GetUser**

El diagrama de la figura 3.4 muestra como el sistema responde a la funcionalidad de acceder a algún usuario del repositorio.

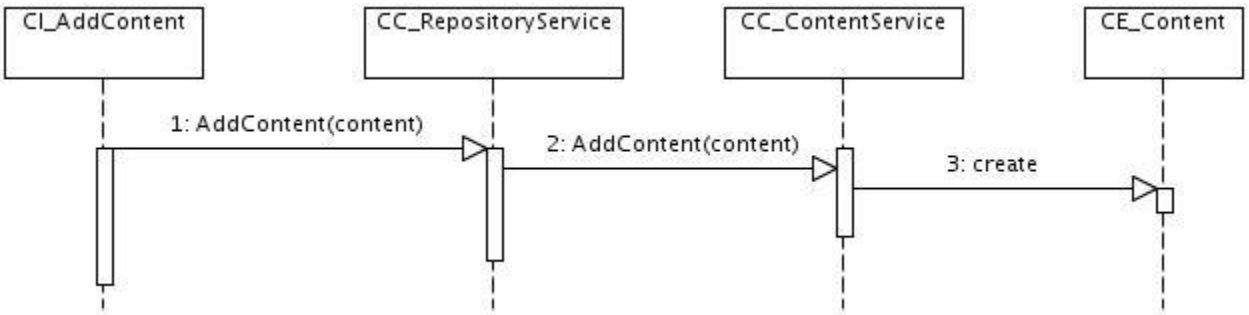



**Figura 3.5 Diagrama de interacción Caso de Uso AddContent**

El diagrama de la figura 3.5 muestra como el sistema responde a la funcionalidad de adicionar algún contenido al repositorio.

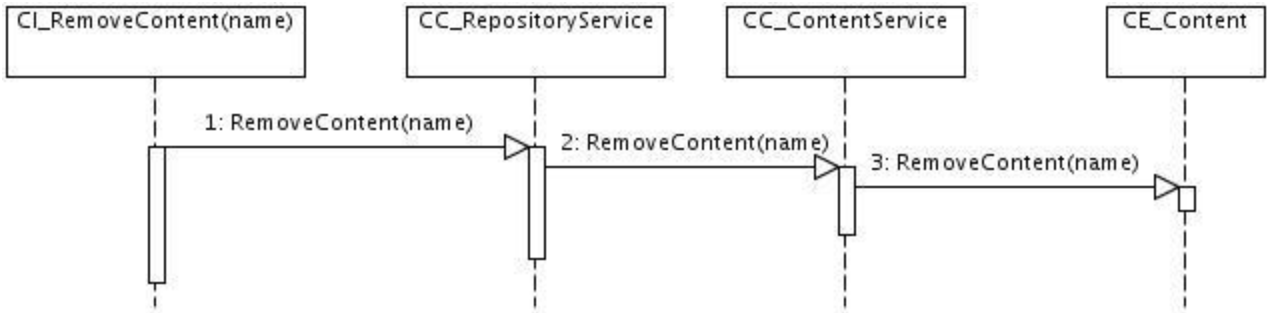



**Figura 3.6 Diagrama de interacción Caso de Uso RemoveContent**

El diagrama de la figura 3.6 muestra como el sistema responde a la funcionalidad de eliminar un tipo de contenido del repositorio.

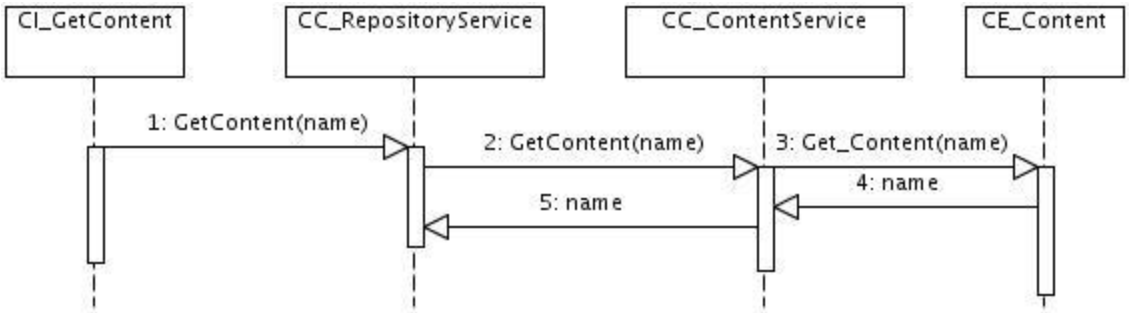



**Figura 3.7 Diagrama de interacción Caso de Uso GetContent**

El diagrama de la figura 3.7 muestra como el sistema responde a la funcionalidad de acceder a un tipo de repositorio.

**Diagramas de clases del diseño.**

Una clase del diseño es una abstracción sin costuras de una clase o construcción similar en la implementación del sistema. El lenguaje que se utiliza en dichas clases es el mismo que el de programación en el cual se implementará el sistema; se especifican atributos y operaciones; se pueden realizar interfaces si tienen sentido para la programación y los métodos tienen correspondencia directa con los métodos en la implementación.

Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra las clases que componen el sistema y las relaciones que existen entre ellos. Este diagrama se utiliza para modelar la vista de diseño estructural de un sistema. Los diagramas de clases además, pueden contener paquetes. Son la base para los posteriores diagramas de componentes y los de despliegue. Su importancia radica en que permitirá construir sistemas ejecutables aplicando ingeniería directa e inversa.

Para un mejor entendimiento del funcionamiento de las clases y las relaciones que se establecen entre las mismas en el sistema propuesto se ha optado por hacer una separación de las clases en paquetes, los cuales contendrán las clases que de cierta manera están más relacionadas. A continuación se muestran los diagramas separados por paquetes.

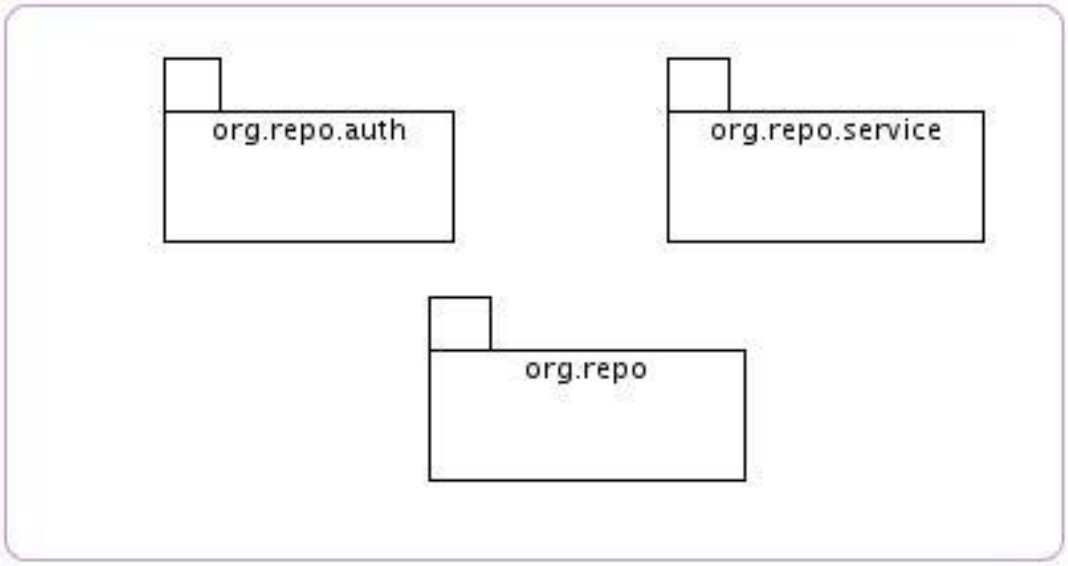



**Figura 3.8 Diagrama de Clases del Diseño**

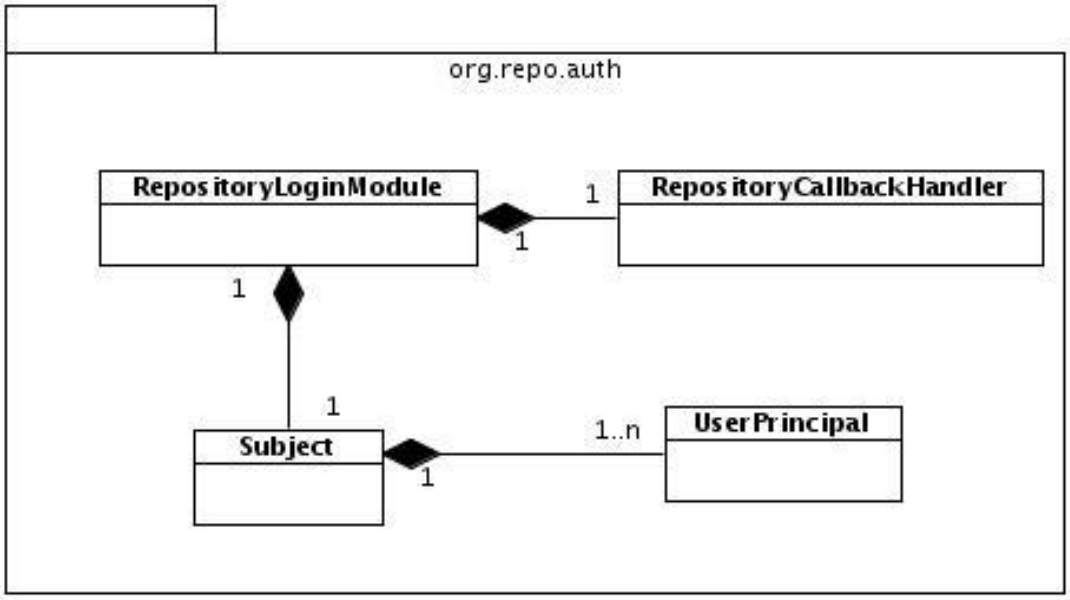



**Figura 3.9 Paquete org.repo.auth**

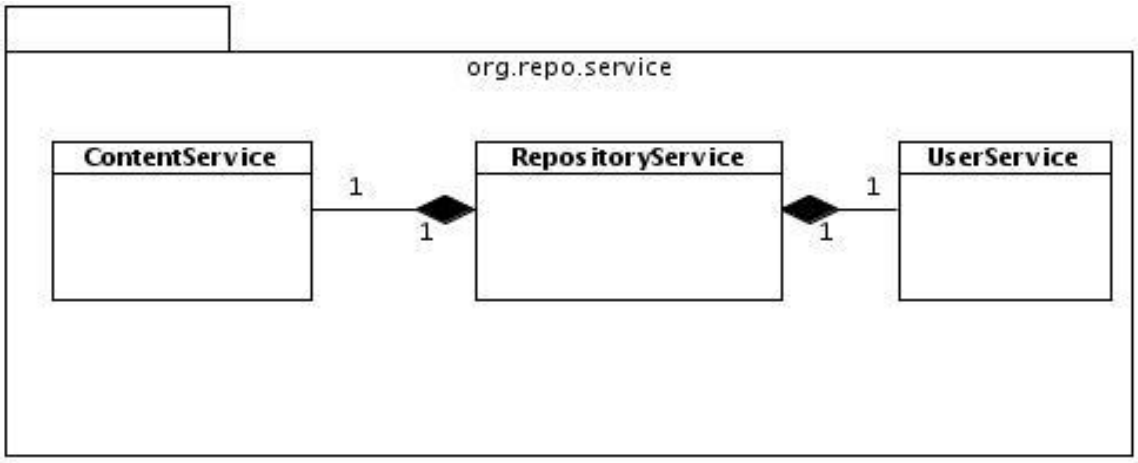



**Figura 3.10 Paquete org.repo.service**

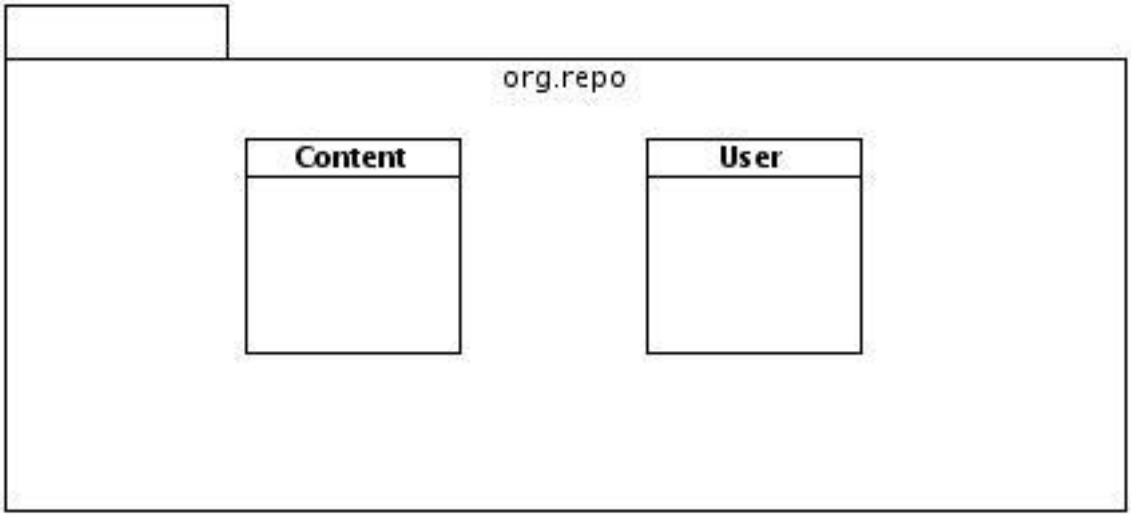



**Figura 3.11 Paquete org.repo**

Finalmente el diagrama de clases queda estructurado de la manera que se muestra en la figura que aparece a continuación. En él se muestran las principales clases e interfaces con las cuales se modela el sistema y las relaciones que se establecen entre ellas, las cuales serán descritas luego para lograr el mejor entendimiento de la razón de ser de las mismas.

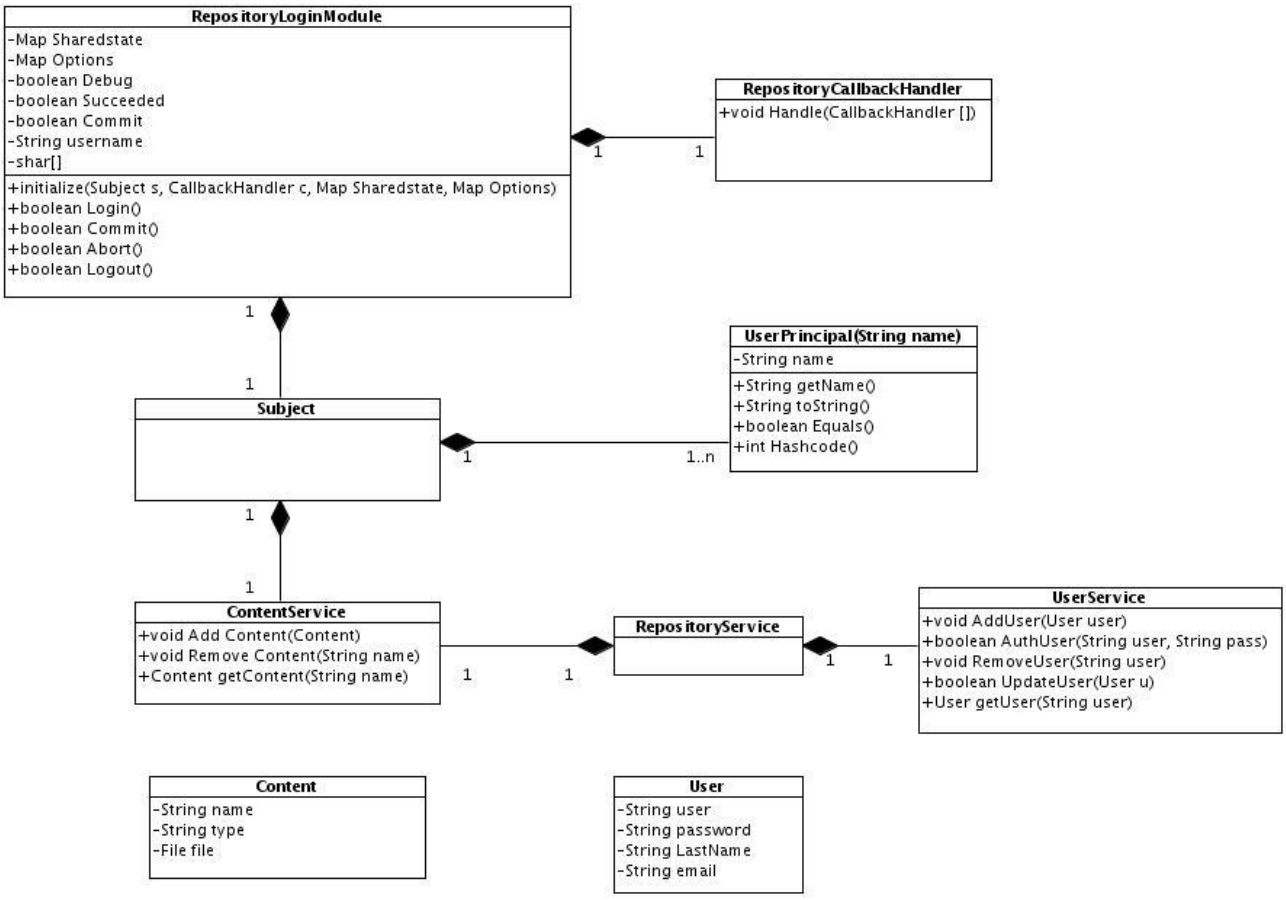



**Figura 3.12 Diagrama de Clases del Sistema**

### Descripción de las principales clases

A continuación se mostrará una descripción de las principales clases que modelan el sistema.

| **Nombre: RepositoryLoginModule** |
|---|
| **Tipo de clase** |

| **Atributo** | **Tipo** |
|---|---|
| Sharedstate | Map |
| Options | Map |
| Debug | boolean |
| Succeded | boolean |
| Commit | boolean |
| username | String |
| password | shar |
| **Principales Responsabilidades** | |
| Nombre | Initialize(Subject s, Call c) |
| Descripción | Inicializar las variables s y c. |
| Nombre | Login() |
| Descripción | Autenticar a los usuarios que accedan al sistema. |
| Nombre | Commit() |
| Descripción | Inicializar los datos de un usuario. |
| Nombre | Abort |
| Descripción | Abortar una operación inicializada. |
| Nombre | Logout |

| | |
|---|---|
| Descripción | Cerrar la sección que inicializo un usuario. |

**Tabla 3.1 Descripción de la clase RepositoryLoginModule**

| **Nombre: UserService** | |
|---|---|
| **Tipo de clase:** | |
| **Atributo** | **Tipo** |
| | |
| **Principales Responsabilidades** | |
| Nombre | AddUser(User user) |
| Descripción | Adicionar nuevos usuarios al repositorio |
| Nombre | AuthUser(String user, String pass) |
| Descripción | Autenticar a cada usuario que acceda al sistema |
| Nombre | Remove(String user) |
| Descripción | Eliminar a un usuario del repositorio |
| Nombre | GetUser(String user) |
| Descripción | Devolver un usuario |
| Nombre | UpdateUser(User u) |
| Descripción | Actuailizar un obejto seleccionado. |

**Tabla 3.2 Descripción de la clase UserService**

| **Nombre: UserPrincipal** | |
|---|---|
| **Tipo de clase:** | |
| **Atributo** | **Tipo** |
| name | String |
| **Principales Responsabilidades** | |
| Nombre | GetName() |
| Descripción | Devuelve el nombre |
| Nombre | ToString() |
| Descripción | |
| Nombre | Equeals() |
| Descripción | |
| Nombre | HasCode() |
| Descripción | |

**Tabla 3.3 Descripción de la clase UserPrincipal**

| **Nombre: ContentService** |
|---|
| **Tipo de clase:** |

| **Atributo** | **Tipo** |
|---|---|
| | |
| **Principales responsabilidades** | |
| Nombre | AddContent(content) |
| Descripción | Adinionar un contenido al respoditorio |
| Nombre | RemoveContent(String name) |
| Descripción | Eliminar un contenido del repositorio |
| Nombre | GetContent(String name) |
| Descripción | Devuelve un contenido dado su nombre |

**Tabla 3.4 Descripción de la clase ContentService**

| **Nombre: RepositoryCallbackHandler** | |
|---|---|
| **Tipo de clase:** | |
| **Atributo** | **Tipo** |
| | |
| **Principales Responsabilidades** | |
| Nombre | **Handle(callbackhandler[ ] c)** |
| Descripción | Capturar las credenciales del usuario |

**Tabla 3.5 Descripcion de la clase RepositoryCallbackHandler**

| **Nombre: Content** | |
|---|---|
| **Tipo de clase:** | |
| **Atributo** | **Tipo** |
| name | String |
| type | String |
| file | File |
| **Principales Responsabilidades** | |
| Nombre | GetName() |
| Descripción | Devuelve el nombre del contenido |
| Nombre | GetType() |
| Descripción | Devuelve el tipo de contenido |
| Nombre | GetFile() |
| Descripción | Devuelve el nombre del contenido |
| Nombre | SetName() |
| Descripción | Cambia el nombre de un contenido por otro |
| Nombre | SetType() |
| Descripción | Cambia un tipo de contenido por otro tipo |

| | |
|---|---|
| Nombre | SetFile() |
| Descripción | Cambia un tipo de contenido por otro tipo |

**Tabla 3.6 Descripcion de la clase Content**

| **Nombre: User** | |
|---|---|
| **Tipo de clase:** | |
| **Atributo** | **Tipo** |
| user | String |
| password | String |
| LasName | String |
| email | String |
| **Principales Responsabilidades** | |
| Nombre | GetUser() |
| Descripción | Devuelve un usuario |
| Nombre | GetPassword() |
| Descripción | Devuelve el password |
| Nombre | GetLasName() |
| Descripción | Devuelve el nombre del usuario |

| | |
|---|---|
| Nombre | GetEmail() |
| Descripción | Devuelve el correo de un usuario |
| Nombre | SetName() |
| Descripción | Cambia el nombre de un usuario por otro |
| Nombre | SetEmail() |
| Descripción | Cambia el correo de un usuario |

**Tabla 3.7 Descripción de la clase User**

**Conclusiones.**

Concluido el capítulo se han dejado claras las premisas necesarias para comenzar a implementar el sistema. Para ello se han transformando los requisitos tanto funcionales como no funcionales a una especificación que describa como implementar el sistema. Durante el transcurso del mismo se ha obtenido una visión sobre lo que debe hacer el sistema a desarrollar, centrándose en los requisitos funcionales y por otro lado el cómo el sistema cumple con los objetivos propuestos, enfocándose así en los requisitos no funcionales. Una vez logrado esto puede decirse que se ha refinado y definido la arquitectura del sistema lo que permitirá adaptar dicho diseño para que sea consistente con el entorno de implementación.

# 4. Implementación

Una vez concluido el diseño del sistema ya se ha capturado la mayor parte de la arquitectura del mismo y se ha creado un plano del modelo de implementación. En el flujo que se comienza a desarrollar en este capítulo, **"implementación"**, se describe cómo los elementos de diseño se implementan en términos de componentes y cómo los mismos se organizan en nodos específicos en el diagrama de despliegue.

Uno de los artefactos más importantes que se generarán en este capítulo es el modelo de implementación, el cual está conformado por el diagrama de despliegue y el diagrama de componentes. Por otra parte, otro de los propósitos de la implementación es desarrollar la arquitectura y el sistema como un todo.

**Diagrama de Despliegue.**

El modelo de despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre dichos elementos. El mismo incluye un artefacto fundamental, el diagrama de despliegue, el cual es usado para visualizar la distribución de los componentes de software en los nodos físicos.

En él se especifican tres elementos fundamentales: los nodos, los dispositivos y los conectores.

Los nodos: Describen los elementos de procesamiento con al menos un procesador, memoria y cualquier otro dispositivo.

Los dispositivos: Son nodos que no tienen capacidad de procesamiento en el nivel de abstracción que se modela.

Los conectores: expresan el tipo de protocolo utilizado en el resto de los elementos del modelo.

A continuación se muestra el diagrama de despliegue correspondiente al sistema que se desea implementar.

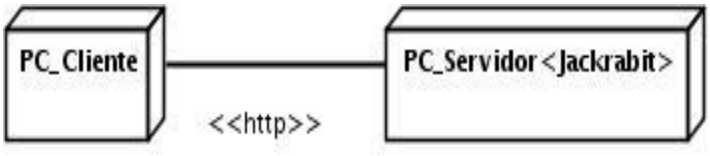



**Figura 4.1 Diagrama de Despliegue**

Como se muestra en la imagen para el uso del sistema solamente son necesarios dos nodos, una PC cliente que es en la cual se ejecuta la aplicación como tal y otra PC en la que se esté ejecutando el servidor de Jackrabbit.

**Diagrama de Componentes.**

Los componentes son partes modulares del sistema, que pueden desplegarse y reemplazarse.

Los mismos encapsulan implementación y un conjunto de interfaces proporcionando la realización de los mismos. Por lo general los componentes contienen clases y pueden ser implementados por uno o más artefactos.

El diagrama de componentes entonces, muestra un conjunto de componentes relacionados entre sí. Su uso fundamental es estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones de los elementos de implementación, modelando de esta manera la vista estática del sistema.

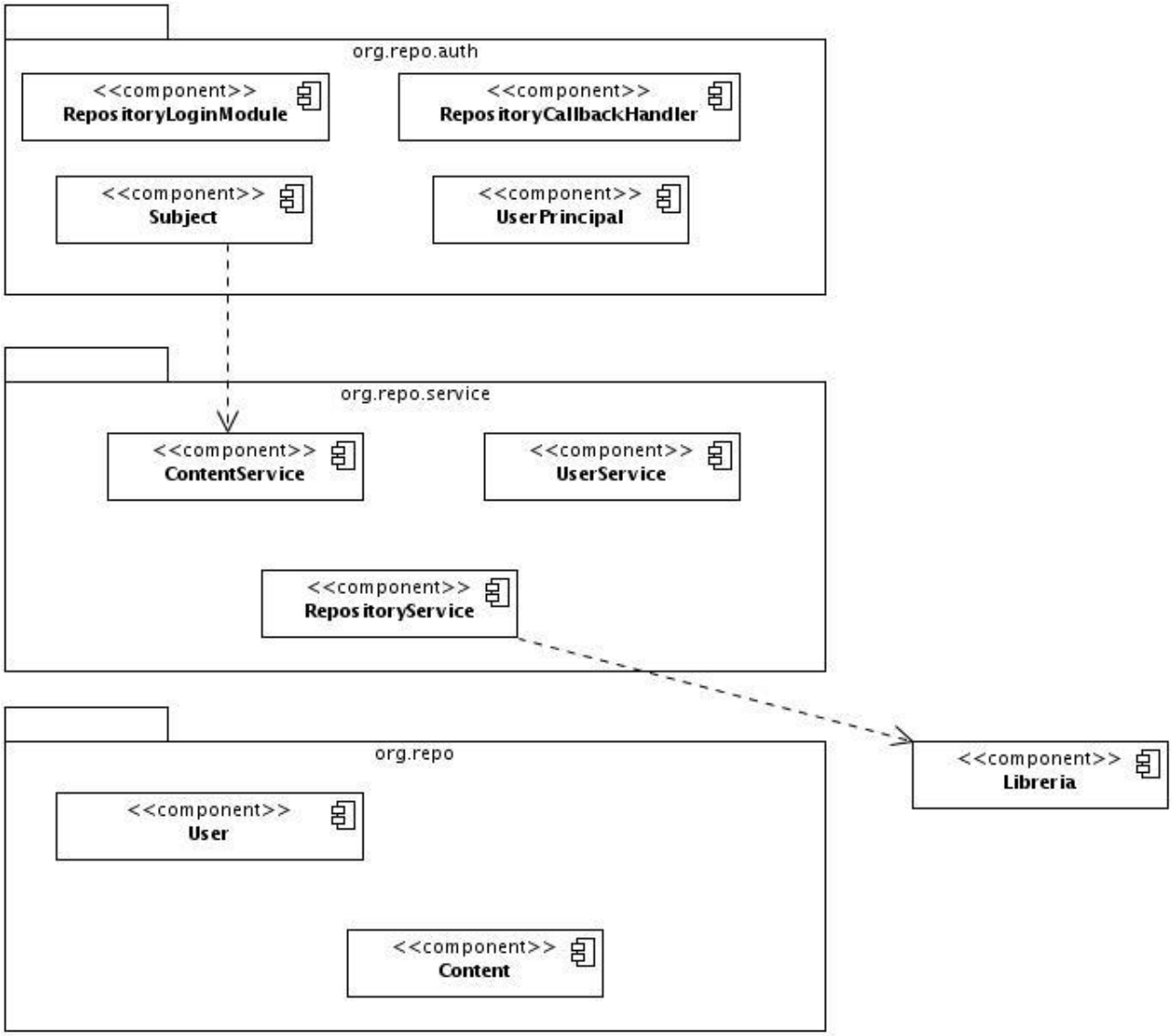



**Figura 4.2 Diagrama de Componentes**

**Estándar de Codificación.**

Los estándares de codificación son importantes para los programadores por un número de razones:

.El 80% del coste del código de un programa va a su mantenimiento.

.Las convenciones de código mejoran la lectura del software, permitiendo entender el código mucho más rápido y más a fondo.

.Si distribuyes el código fuente de tu producto, necesitarás asegurarte que el mismo sea tan presentable como la misma aplicación.

Durante la codificación es importante que se consideren permanentemente los siguientes criterios de calidad:

| Criterio | Objetivo |
|---|---|
| Facilidad de Comunicación | Proporcionar al usuario entradas y salidas fácilmente asimilables. |
| Auto descripción | Proporcionar en el código , explicaciones sobre la implantación realizada |
| Simplicidad | La implantación realizada debe hacerse de la forma más comprensible posible |

De manera general se proponen las siguientes nomenclaturas para desarrollar el sistema:

**Principios generales:**

Los nombres de cada uno de los elementos del programa deben ser significativos; su nombre debe explicar en lo posible el uso del elemento.

La mayoría de los elementos se deben nombrar usando sustantivos (posiblemente compuestos), o formas verbales en imperativo.

La forma de construir los nombres será colocando primero el verbo o el sustantivo, seguido de cada uno de sus complementos con la primera letra en mayúscula.

**Nombramiento de los elementos:**

| Elemento | Regla de nombramiento |
|---|---|
| Clases, interfaces y archivos fuente | Nombre sustantivo singular, con la primera letra en mayúscula y las demás en minúsculas. |
| Constantes | Nombre sustantivo en mayúsculas. Para separar palabras se usara el guion bajo. |
| Atributos | Nombre sustantivo en minúsculas |
| Métodos | Nombre sustantivo en minúsculas, si retorna un valor.<br>Nombre verbal en minúsculas, si no retorna valor.<br>Para los métodos analizadores y modificadores de atributos, más conocidos como métodos de acceso, alternativamente puede utilizarse el estándar getAtributo() y |

| | |
|---|---|
| | setAtributo(). |
| Paquetes y directorios | Nombre sustantivo. |

**Estructura de los archivos.**

Todas las fuentes deben tener la estructura básica general.

```
//===================================================================

// ARCHIVO

// FECHA CREACIÓN:

// AUTOR:

// .... Comentarios generales

//===================================================================
```

```
package xxxx;

//============================================================================

// BIBLIOTECAS REQUERIDAS

//============================================================================

import xxx ;

....
```

**Cada clase o interfaz debe tener la siguiente estructura básica general**.

```
//============================================================================

// CLASE NombreDeLaClase
```

```
//=========================================================================

....

//-------------------------------------------------------------------

// ATRIBUTOS DE INSTANCIA

//-------------------------------------------------------------------

....

//-------------------------------------------------------------------

// ATRIBUTOS DE CLASE (ESTATICOS)

//-------------------------------------------------------------------
```

```
....

//-------------------------------------------------------------------

// VALIDACIÓN DE INVARIANTE

//-------------------------------------------------------------------

private boolean invarianteOk ()

//-------------------------------------------------------------------

// METODOS CONSTRUCTORES

//-------------------------------------------------------------------

....
```

```
//-------------------------------------------------------------------

// MÉTODOS DE INSTANCIA

//-------------------------------------------------------------------

....

//-------------------------------------------------------------------

// METODOS DE CLASE (ESTATICOS)

//-------------------------------------------------------------------

....

//-------------------------------------------------------------------
```

```
// INICIALIZADOR DE CLASE (ESTATICO)


//-------------------------------------------------------------------


....
```

Si alguna de las secciones no aparece pueden eliminarse las líneas de encabezado.

**Tamaño de las líneas**

Las líneas deben ser de máximo 99 caracteres. Si es necesario partir la línea, la siguiente debe alinearse dejando doble sangría.

**Comentarios**

El código debe comentarse utilizando la sintaxis apropiada para uso de javadoc, teniendo en cuenta que para la producción de la documentación deben incorporarse los tags particulares que no hagan parte del estándar.

Comentarios de clases e interfaces

```
**


* Descripción de la clase
```

```
 * @author Nombre del desarrollador 1

 * @author Nombre del desarrollador 2

 * @version número versión y fecha

 * @inv Invariante de clase

*/
```

**Comentarios de atributos.**

```
/** Descripción del atributo */
```

Comentarios de métodos constructores

```
/**
```

```
   * Descripción de lo que hace el método

   * @param p Descripción del parámetro.

   * @pre precondición uno

   * @pre precondición dos

   * @pos poscondición uno

   * @pos poscondición uno

   * @exception TipoExcepción, Condiciones que causan la excepción

*/
```

**Comentarios de métodos analizadores o métodos de acceso.**

```
/**

   * Descripción de lo que hace el método

   * @param p Descripción del parámetro.

   * @pre precondición uno

   * @return valor a retornar

   * @exception TipoExcepción, Condiciones que causan la excepción

*/
```

**Comentarios de métodos modificadores.**

```
/**
```

```
   * Descripción de lo que hace el método

   * @param p Descripción del parámetro.

   * @pre precondición uno

   * @pos poscondición uno

   * @exception TipoExcepción, Condiciones que causan la excepción

*/
```

**Sangría y ablocamiento.**

Todos los archivos fuentes deben seguir el estándar de sangría. Cada entrada corresponde a 4 espacios. No debe usarse el carácter de tabulación, pues es dependiente de la configuración del editor. Todos los constructores que admiten bloques de instrucciones delimitados por {...}, deben usarlos aún si éstos tienen una sola instrucción.

**Definición de clases e interfaces**

```
..... {

        ... ..... ;

        ... ..... ;

        ... ..... ;

        ... ..... ;

} ..... ;
```

**Definición de métodos**

```
.... ... ..... ( .... , .... , .... ){

      ..... ;
```

```
      ..... ;

}

.... ... ..... ( .... , .... , .... )

throws ...,.... {

      ..... ;

      ..... ;

}
```

**Estructuras condicionales**

```
if ( ..... ) {
```

```
    ..... ;

    ..... ;

    ..... ;

} else if ( ..... ) {

    ..... ;

    ..... ;

    ..... ;

} else {

    ..... ;
```

```
    ..... ;

    ..... ;

}
```

```
switch ( ..... ) {

    case ..... :

        ..... ;

        ..... ;

        ..... ;
```

```
break;

case ..... :

    ..... ;

    ..... ;

    ..... ;

break;

case ..... :

    ..... ;

    ..... ;
```

```
            ..... ;

    default:

            ..... ;

            ..... ;

            ..... ;

}
```

**Estructuras de iteración.**

```
while ( ..... ) {

      ..... ;
```

```
      ..... ;

      ..... ;

}
```

```
for ( ..... ; ..... ; ..... ) {

      ..... ;

      ..... ;

      ..... ;

}
```

```
do {

      ..... ;

      ..... ;

      ..... ;

} while ( ..... ) ;
```

**Estructuras de excepción.**

```
try {

    .....;

    .....;
```

```
} catch (....) {

    ....;

} catch (....) {

    ....;

} finally {

    .... ;

}
```

**Pruebas al Sistema.**

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y alguna evaluación es hecha de algún aspecto del sistema o componente.

La creciente inclusión del software como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo, han motivado la creación de pruebas más minuciosas y bien planificadas. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

**Prueba de caja negra.**

La prueba de Caja Negra se centra principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa según si lo que se desea es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa funcione bien.

**Casos de prueba de integración.**

Los casos de prueba son un conjunto de condiciones mediante las cuales se comprueba si los requisitos funcionan correctamente. Las descripciones de los casos de prueba de integración realizadas a continuación contienen el nombre del caso de uso, en el caso de prueba, se describe el estado de la información almacenada, el resultado que se obtiene y las condiciones que se deben cumplir en el momento que se ejecuta el caso de uso.

| Entrada | Resultado | Condiciones |
|---|---|---|
| Autenticar un usuario de forma correcta. | Accede a la aplicación sin ningún problema. | Que los ingresados datos sean correctos. |
| Autenticar un usuario de manera incorrecta. | Muestra un cartel de error. | Que los ingresados datos sean incorrectos. |

**Tabla 4.1 Caso de Prueba "Autenticar usuario"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Adicionar un usuario de manera correcta. | Se adiciona al sistema el usuario, con todos los datos solicitados por la aplicación. | Se ingresen correctamente todos los datos solicitados. |
| Adicionar un usuario de manera incorrecta. | Muestra un cartel solicitando cambiar los datos ingresados incorrectamente. | Que el usuario exista ya en el repositorio o que las claves coincidan. |

**Tabla 4.2 Caso de Prueba "Adicionar usuario"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Eliminar un usuario de manera correcta. | Se elimina correctamente el usuario seleccionado. | Que sea el admin del sistema. |
| Eliminar un usuario de manera incorrecta. | Muestra un cartel de error ya que no es el admin del sistema. | Que no sea admin del sistema. |

**Tabla 4.3 Caso de Prueba "Eliminar usuario"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Crear carpeta de manera correcta. | Crea la carpeta correctamente dentro del repositorio. | Tener los permisos requeridos para crear una carpeta. |
| Crear carpeta de manera incorrecta. | Muestra un error de que ya existe una carpeta con ese nombre. | Si ya existe la carpeta dentro del repositorio. |

**Tabla 4.5 Caso de Prueba "Crear carpeta"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Adicionar documentos de manera correcta. | Se adiciona correctamente el documento seleccionado. | Tener los permisos requeridos para adicionar el documento seleccionado. |
| Adicionar documentos de manera incorrecta. | Muestra un cartel de error, que no tiene permiso para adicionar el documento seleccionado. | No tiene permisos para adicionar el documento seleccionado.. |

**Tabla 4.6 Caso de Prueba "Adicionar documentos"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Descargar documentos de manera correcta. | Se descarga correctamente el documento seleccionado. | Tener los permisos requeridos para |

| | | |
|---|---|---|
| | | descargar el documento. |
| Descargar documentos de manera incorrecta. | Muestra un cartel de error, que no tiene permiso para descargar el documento seleccionado. | No tiene permisos para descargar documentos. |

**Tabla 4.7 Caso de Prueba "Descargar documento"**

| Entrada | Resultado | Condiciones |
|---|---|---|
| Eliminar documentos de manera correcta. | Se elimina correctamente el documento seleccionado. | Tener los permisos requeridos para eliminar el documento. |
| Eliminar documentos de manera incorrecta. | Muestra un cartel de error, que no tiene permiso para descargar el documento seleccionado. | No tiene permisos para eliminar documentos. |

**Tabla 4.8 Caso de Prueba "Eliminar documento"**

**Conclusiones.**

Una vez concluido este capítulo se ha refinado la vista de la arquitectura del modelo de despliegue, donde los componentes son asignados a nodos. El modelo de implementación es la entrada principal para las etapas de prueba que siguen directamente luego de dicha etapa. Una vez concluida la misma ya el sistema se puede decir que está casi listo para las posteriores actividades de despliegue.

# Conclusiones

Con la realización del presente trabajo se arribaron a las siguientes conclusiones:

El repositorio de contenido desarrollado facilita la recopilación de la información que contendrá el Diccionario Enciclopédico "Libertad" de forma centralizada y organizada. Se logró un producto de fácil manejo para cualquier usuario que interactúe con él mismo.

La metodología de desarrollo de software RUP contribuyó a un correcto análisis y diseño de cada una de las funcionalidades de la aplicación, la misma permitió que se concretaran cada una de las fases del trabajo. Para ello se hizo un estudio preliminar de la situación actual del Diccionario Enciclopédico "Libertad" desde el punto de vista de la forma en que se almacenaba la información recopilada.

# Recomendaciones

Luego de haber concluido el presente trabajo se recomienda:

1. Continuar profundizando en el estudio de la JSR-170 y la especificación Jackrabbit para implementarles nuevas funcionalidades al repositorio de contenidos que mejoren los servicios ofertados, como por ejemplo el versionado y clasificación de los documentos y la búsqueda avanzada de los mismos.
2. Desplegar el sistema en otras instituciones que trabajen en la recopilación de datos.
3. La utilización de este trabajo como bibliografía y material de apoyo para las investigaciones futuras en esta área de desarrollo.

# Bibliografía

Ajaxman_net. 2008. [Java]Pensando en java: Que es J2SE, J2EE, J2ME y Java card. [En línea] 28 de Marzo de 2008. [Citado el: 12 de Febrero de 2010.] [Disponible en]: http://www.ajaxman.net/657/javapensando-en-java-que-es-j2se-j2ee-j2me-y-java-card/.

Barros, L. 2009. Seminario de Herramientas. [En línea] 2009. [Citado el: 5 de Febrero, 2010.] [Disponible en]: http://www.ctr.unican.es/asignaturas/procodis_3_II/Doc/SeminarioHerramientas.pdf.

buenas_tareas. 2009. HERRAMIENTAS CASE. [En línea] 2009. [Citado el: 20 de Enero, 2010.] [Disponible en]: http://www.buenastareas.com/ensayos/Ingeniero-De-Sistemas/326796.html.

Canal_Visual_Basic.net. 2008. Manuales .NET. Canal Visual Basic.net. [En línea] 19 de Junio de 2008. [Citado el: 15 de Diciembre de 2009.] [Disponible en]: http://www.canalvisualbasic.net/manual-net/c-sharp/#cSharp.

Casals, V. 2008. Repositorios-Documentos. Ciudad de la Habana : s.n., 2008.

CENTROCAMALEON.COM. 2010. HERRAMIENTA CASE. [En línea] 21 de Enero, 2010. [Citado el: 10 de Febrero, 2010.] [Disponible en]: http://www.centrocamaleon.com/v1/?p=233.

Ciberaula. 2007. ¿Qué es Java? Ciberaula. [En línea] 15 de Enero de 2007. [Citado el: 16 de Diciembre de 2009.] [Disponible en]: http://java.ciberaula.com/articulo/que_es_java.

Ciencia_y_Técnica_Administrativa. [En línea] [Citado el: 15de Enero, 2010.] [Disponible en]: http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm.

Creación_de_ Software,2009. Java Enterprise Edition. [En línea] [Citado el: 15 de Febrero de 2010.] [Disponible en]: http://creaciondesoftware.es/CreacionSoftware/Tecnologia.html.

Ferré, X y Sánchez, M. 2005. Desarrollo Orientado a Objetos con UML. [En línea] 2005. [Citado el: 10 de Marzo de 2010.] [Disponible en]: http://www.clikear.com/manuales/uml/diagramasinteraccion.aspx.

Flores, G y Sánchez, N. 2007. Los repositorios institucionales: análisis de la situación internacional y principios generales para Cuba. Los repositorios institucionales. [En línea] 2007. [Citado el: 25 de Noviembre de 2009.] [Disponible en]: http://bvs.sld.cu/revistas/aci/vol16_6_07/aci061207.htm.

Fundación_Héctor_A_García. Breve Historia de las Enciclopedias. [En línea] 2005. [Citado el: 20de Noviembre, 2009.] [Disponible en]: http://www.proyectosalonhogar.com/breve_historia_de_las_enciclopedias.htm.

Gattaca_ S.A. 2006. Presentación de Metodología MSF. [En línea] 2006. [Citado el: 10 de Enero de 2010.] [Disponible en]: http://www.e-gattaca.com/eContent/library/documents/DocNewsNo50DocumentNo6.PDF.

Gomez D, Aranda E y Fabrega J. Programación Extrema. [En línea] [Citado el: 17 de Diciembre de 2009.] [Disponible en]: http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf.

Gónzalez, J, 2009. ¿Qué es UML? [En línea] 2009. [Citado el: 12 de Enero de 2010.] [Disponible en ]: http://www.docirs.cl/uml.htm#diagrama.

HELLFREDMANSON. 2009. JAVA Y C++ . HELLFREDMANSON. [En línea] 4 de Agosto de 2009. [Citado el: 16 de Diciembre de 2009.] [Disponible en]: http://hellfredmanson.over-blog.es/article-34580352.html.

Hippo_Community_Site. 1999. Hippo Community. [En línea] 1999. [Citado el: 26 de Junio de 2010.] [Disponible en]: http://www.onehippo.org/.

Huaroto, L. 2007. Repositorio Digitales de Documentos. [En línea] Diciembre de 2007. [Citado el: 30 de Noviembre de 2009.] [Disponible en]: http://www.slideshare.net/lhuaroto/repositorio-digital.

IBM. 2005. Introducing the Java Content Repository API. [El línea] Agosto 23, 2005. [Citado: 19 de Enero, 2010.] [Disponible en]: http://www.ibm.com/developerworks/java/library/j-jcr/.

Instituto_Politécnico_Nacional. 2005. Las TIC's en el IPN. [En línea] 2005. [Citado en: 1de Febrero, 2010.] [Disponible en]: http://www.dcyc.ipn.mx/dcyc/glosario/L.aspx.

Java_Community_Process. 2006. JSR 170: Content Repository for JavaTM technology API. [En línea] 2006. [Citado el: 26 de Junio de 2010.] [Disponible en]: http://www.jcp.org/en/jsr/detail?id=170.

Letelier P y Penadés M. 2006. Métodologías ágiles para el desarrollo de software:eXtreme Programming (XP). [En línea] 2006. [Citado el: 11de Enero, 2010.] [Disponible en]: http://www.willydev.net/descargas/masyxp.pdf.

Magnolia. 2003. Magnolia CMS. [En línea] 2003. [Citado el: 26 de Junio de 2010.] [Disponible en]: http://www.magnolia-cms.com/home.html.

Mairena. 2009. UML DIAGRAMAS. [En línea] 8 de Agosto, 2009. [Citado el: 12 de Enero, 2010.] [Disponible en]: http://www.gratisblog.com/tecnologiamairena/i130272-uml__diagramas.htm.

Navarro, Clara. 2009. Los distintos tipos de diccionarios. suite101.net. [En línea] 19 de Noviembre de 2009. [Citado el: 4 de Diciembre de 2009.] [Disponible en]: http://filologia.suite101.net/article.cfm/los_distintos_tipos_de_diccionarios.

Noticiasdot_com. 2004. Sun anuncia versión beta de la nueva plataforma Java . [En línea] 19 de Febrero de 2004. [Citado el: 12 de Febrero de 2010.] [Disponible en]: http://www.noticiasdot.com/publicaciones/2004/0204/1902/noticias190204/noticias190204-5.htm.

Nuxeo_The_Community. 2006. Welcome to the Nuxeo Community Site. [En línea] Diciembre de 2006. [Citado el: 26 de Junio de 2010.] [Disponible en]: http://www.nuxeo.org/xwiki/bin/view/Main/.

OpemKM_Knowledge_Management. 2006. Collaborate and comunicate. [En línea] 2006. [Citado el: 26 de Junio de 2010.] [Disponible en]: http://www.openkm.com/.

Patil, S. 2006. What is Java Content Repository. [En línea] Abril 10, 2006. [Citado el: 18 de Enero, 2010.][Disponible en]: http://onjava.com/pub/a/onjava/2006/10/04/what-is-java-content-repository.html.

Rivera, P. 2003. La enseñanza de la Historia de la contemporaneidad y la Educación para la Paz y los Derechos Humanos. [En línea] 2003. [Citado el: 19de Noviembre, 2009.] [Disponible en]: http://www.ciget.pinar.cu/No.2003-4/historia.htm.

RUP. 2005. Rational Unified Process Version 7.0.1. 2005.

Sánchez, J. 2005. Metodología para el Desarrollo de Software. [En línea] 2005. [Citado el: 19de Diciembre, 2009.] [Disponible en]: http://www.lcc.uma.es/~jignacio/index_archivos/TEMA4.pdf.

Sitio_de_descarga_de_Software. 2007. Visual Paradigm para UML (CE) en Windows (Visual . [En línea] 5 de marzo, 2007. [Citado en: 12 de Enero, 2010.] [Disponible en]: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28Iglesia_Anglicana%29_para_Windows_14718_p/.

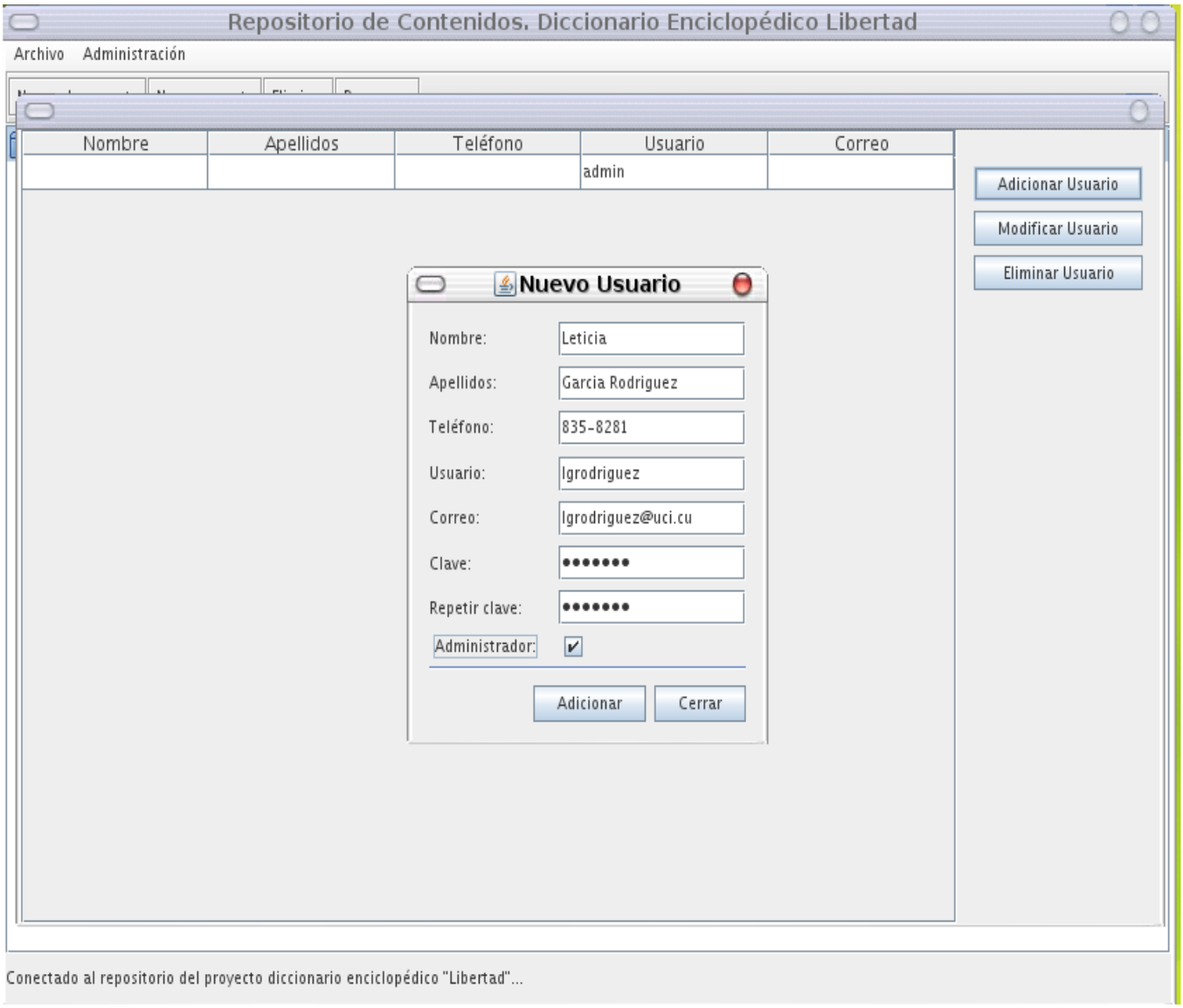
Taringa.net. 2007. Aprender lenguaje C++. Taringa.net. [En línea] 6 de Octubre de 2007. [Citado el: 15 de Diciembre de 2009.]

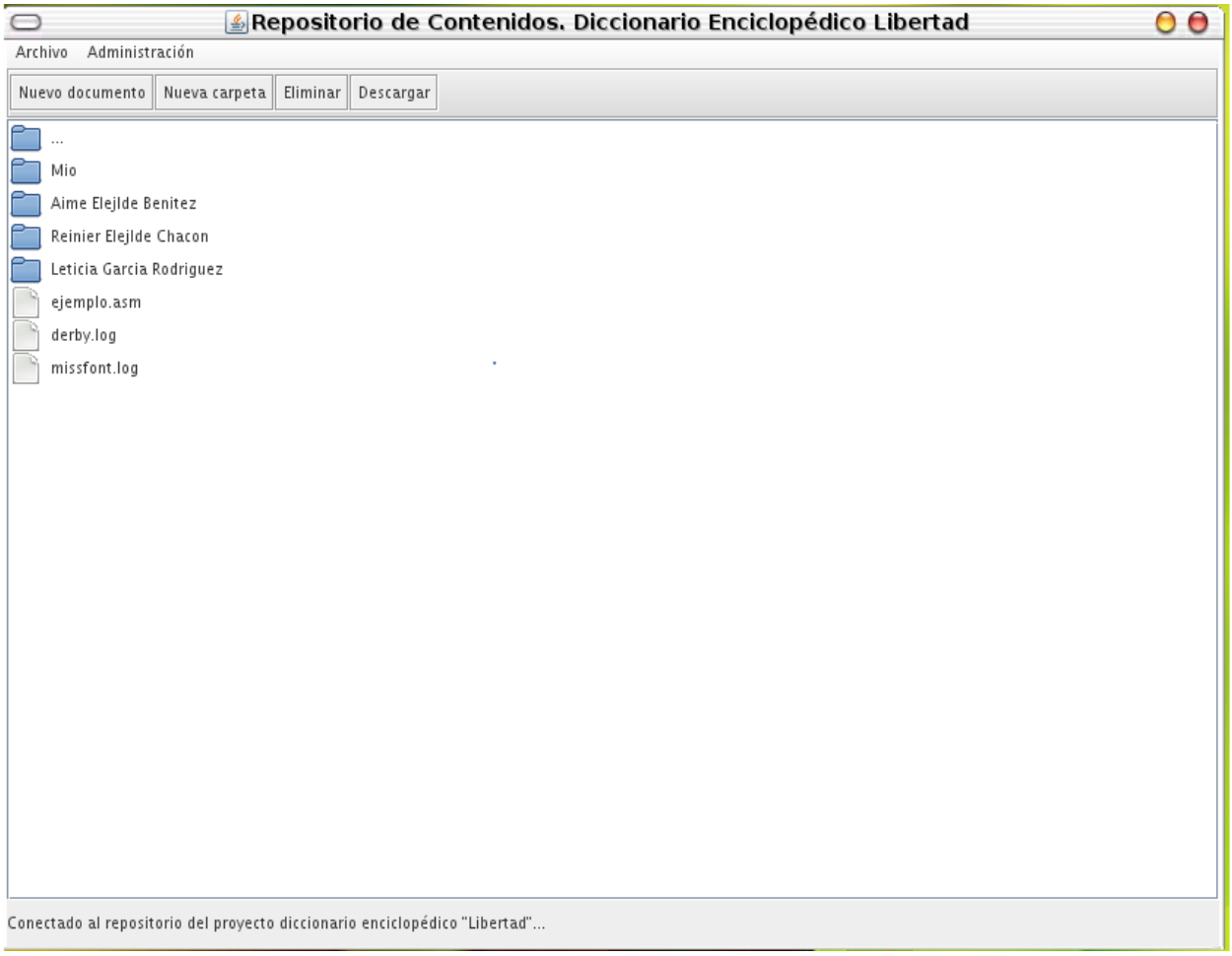
The_Apache_Software_Foundation. 2004. Apache Jackrabbit. [En línea] 2004. [Citado el: 20 de Enero de 2010.] [Disponible en]: http://jackrabbit.apache.org/node-types.html.

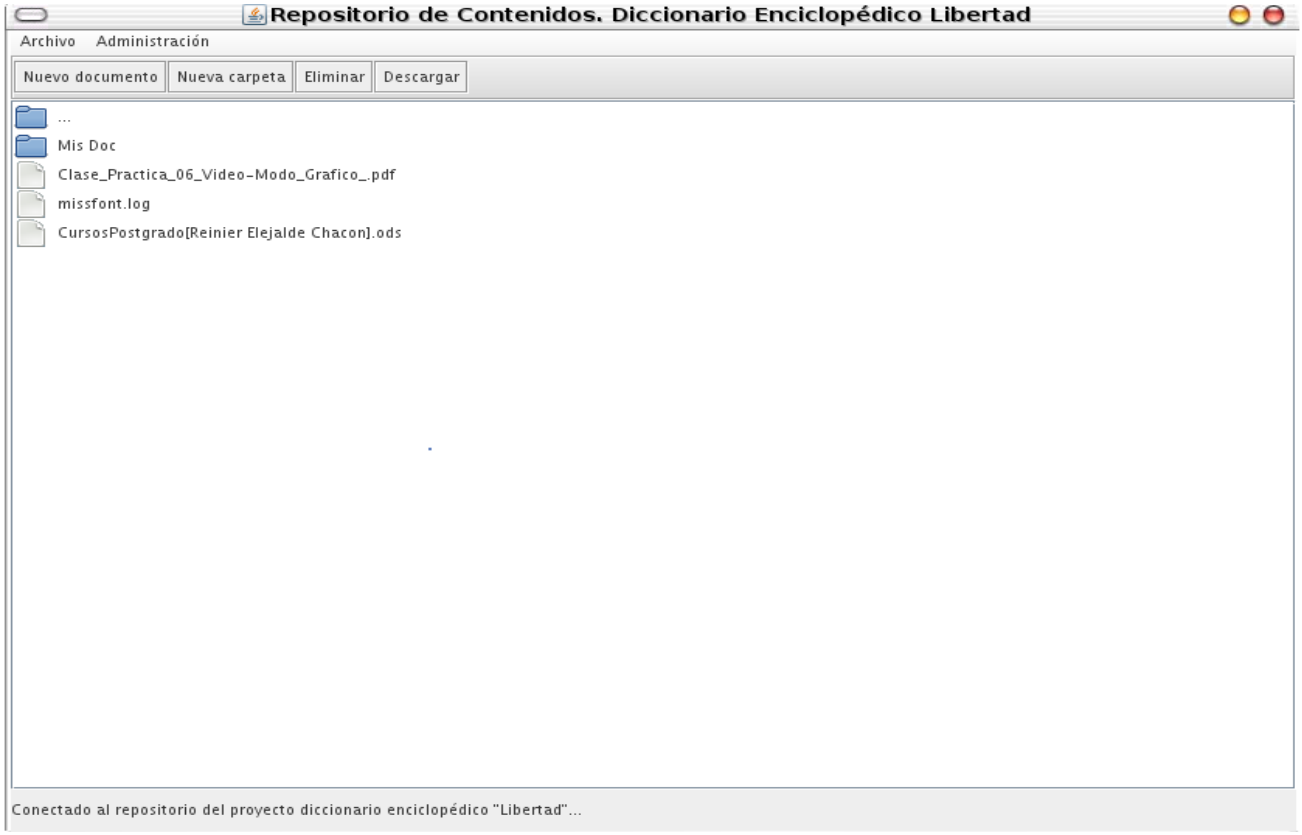
Torres, C. 2009. SISTEMAS OPERATIVOS LIBRE DE LINUX. [En línea] 21 de Abril de 2009. [Citado el: 29 de Junio de 2010.] [Disponible en]: http://lomejorenpc.blogspot.com/2009_04_01_archive.html.

www.canalhanoi.com. 2007. Lenguajes de Programación mas Comunes. www.canalhanoi.com. [En línea] Mayo de 2007. [Citado el: 15 de Diciembre de 2009.] [Disponible en]: http://canalhanoi.iespana.es/programacion/lengucomunes.htm#LENGUAJE%20C.

# Anexos


Repositorio de Contenidos. Diccionario Enciclopédico Libertad
Archivo
Administración
Nombre
Apellidos
Teléfono
Usuario
Correo
admin
Adicionar Usuario
Modificar Usuario
Eliminar Usuario
Nuevo Usuario
Nombre:
Leticia
Apellidos:
Garcia Rodriguez
Teléfono:
835-8281
Usuario:
lgrodriguez
Correo:
lgrodriguez@uci.cu
Clave:
Repetir clave:
Administrador:
Adicionar
Cerrar
Conectado al repositorio del proyecto diccionario enciclopédico "Libertad"...


Repositorio de Contenidos. Diccionario Enciclopédico Libertad
Archivo
Administración
Nuevo documento
Nueva carpeta
Eliminar
Descargar
...
Mio
Aime Elejlde Benitez
Reinier Elejlde Chacon
Leticia Garcia Rodriguez
ejemplo.asm
derby.log
missfont.log
Conectado al repositorio del proyecto diccionario enciclopédico "Libertad"...


Repositorio de Contenidos. Diccionario Enciclopédico Libertad
Archivo
Administración
Nuevo documento
Nueva carpeta
Eliminar
Descargar
...
Mis Doc
Clase_Practica_06_Video-Modo_Grafico_.pdf
missfont.log
CursosPostgrado[Reinier Elejalde Chacon].ods
Conectado al repositorio del proyecto diccionario enciclopédico "Libertad"...