Universidad de las Ciencias Informáticas
Facultad 1

Facultad 1

Propuesta de Arquitectura para el Sistema de Gestión de Credenciales

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Yasel Rafael Quevedo Montero

Tutora: Lic. Yeneit Delgado Kios

Ciudad de La Habana, 20 de julio de 2008

"Año 50 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yasel Rafael Quevedo Montero.	Lic. Yeneit Delgado Kios
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Tutora: Lic. Yeneit Delgado Kios
Licenciada en Ciencia de la Computación
Profesora Instructora, 3 años de graduada
Dpto. de Técnicas de Programación
Universidad de las Ciencias Informáticas
E-mail: yeneit@uci.cu

AGRADECIMIENTOS

En especial a mi mamá y a mi tía Daysi, que es otra madre para mí, ambas me guiaron siempre por el buen camino y confiaron siempre en mí.

A toda mi familia, porque de alguna forma todos me ayudaron y me apoyaron en estos 5 años.

A mi novia Yudileidis por estar siempre a mi lado.

A mis compañeros y amigos por hacer que estos años sean inolvidables y por ayudarme siempre en lo que pudieron.

A mi tutora Yeneit por guiarme en la realización de este trabajo.

A nuestro Comandante Fidel Castro y a la UCI por haberme formado como profesional.

A todas las personas que de una forma u otra contribuyeron a la realización de este trabajo.

DEDICATORIA

A mi madre, mi hermana y mi tía Daysi, por estar siempre a mi lado apoyándome incondicionalmente en todo y a pesar de todo, por darme un motivo y fuerzas para seguir adelante siempre, no sé que hubiera sido sin ustedes.

A mi padre que aunque ya no está, en estos momentos sé que estaría muy orgulloso de mí.

A mi familia que me apoyó anímica, moral, material y económicamente durante todos estos años, siempre se los voy a agradecer.

Resumen

La identificación de personas en el marco de la Universidad de las Ciencias Informáticas (UCI) toma especial importancia debido al gran número de personas que de alguna forma residen o tienen acceso a ella. Hoy en día existe un sistema automatizado en la UCI, pero que no se ajusta a las nuevas medidas de la Dirección de Informatización, de esta manera surge la necesidad de crear un nuevo Sistema de Gestión de Credenciales que tiene como objetivo concreto, rediseñar e implementar todo el proceso de acreditación de personas, haciéndolo lo suficientemente flexible.

En el presente trabajo se realiza una propuesta de arquitectura para el Sistema de Gestión de Credenciales. Durante el desarrollo del mismo se analizan las principales tendencias, tecnologías y metodologías actuales, se definen los principales requerimientos del sistema y se obtiene como resultado el diseño de la arquitectura, definiendo de esta forma el estilo arquitectónico, lenguaje de programación, entorno de desarrollo y gestor de base de datos, que deberán ser usados en la implementación del sistema.

PALABRAS CLAVES: Arquitectura de Software, Credenciales, Acreditación.

Índice

Índice de tablas

Índice de figuras

INTRODUCCIÓN

En la Universidad de las Ciencias Informáticas (UCI) se hace necesario tener acreditadas a todas las personas por la alta cantidad de personal que reside y trabaja a diario en ella. Esto lleva un proceso, el cual debe ser eficiente y lo más rápido posible debido a los nuevos ingresos (estudiantes y trabajadores) que cada año entran a la UCI y a la gran cantidad de credenciales que se imprimen a diario, ya sea por deterioro, pérdida, etc.

En toda institución la credencial de identificación juega un papel muy importante. Esta permite a las empresas e instituciones proteger a grandes rasgos cada movimiento mediante estas tarjetas, gracias al gran avance de la tecnología existente. Para una mejor seguridad en las organizaciones, la creación y el diseño de credenciales es indispensable, ya que estas permiten personalizar los datos y la foto para el reconocimiento del personal, garantizando una identificación única y segura, sin posibilidades de ser duplicadas o falsificadas.

El Sistema de Gestión de Credenciales con el que se cuenta actualmente en la UCI para realizar todo este proceso carece de funcionalidades que son imprescindibles, lo que hace engorroso e ineficiente el trabajo de la especialista de acreditación. Además las tendencias del país y la UCI a incorporarse dentro del mercado del desarrollo de software libre hacen necesario reestructurar la actual arquitectura del Sistema de Gestión de Credenciales, surgiendo así como **problema científico**: ¿Qué Arquitectura utilizar en el Sistema de Gestión de Credenciales sobre software libre, que cumpla con las necesidades del proyecto Identificación y Control de Acceso?

A partir de lo anterior se plantea la siguiente **idea a defender**: Definiendo la arquitectura del Sistema de Gestión de Credenciales es posible obtener un sistema desarrollado en software libre y que cumpla con los nuevos requerimientos.

Se define como **objeto de estudio** de este trabajo el Sistema Identificación y Control de Acceso de la UCI y su **campo de acción** lo forma la Arquitectura del Sistema de Gestión de Credenciales.

Para dar solución a la situación problémica y al problema científico se traza como **objetivo general**: Definir la arquitectura del Sistema de Gestión de Credenciales del proyecto Identificación y Control de Acceso, teniendo en cuenta su desarrollo en software libre, los nuevos requerimientos e identificar las técnicas y herramientas apropiadas para ello.

Como **objetivos específicos** tendremos:

- Definir componentes y funcionalidades reusables que agilicen el desarrollo de la aplicación.
- Desarrollar un marco arquitectónico que permita el desarrollo de la aplicación de forma paralela en cada una de las capas lógicas.
- Definir patrones de diseño útiles para el desarrollo de la aplicación.
- Definir las herramientas apropiadas para el desarrollo de la solución.

Para el desarrollo de la investigación se proponen las siguientes **preguntas científicas**:

- ¿Cuál es el tipo de arquitectura de software a emplear de modo que resuelva las necesidades del proyecto y de la UCI?
- ¿Cuáles son los patrones que pueden ser aplicables al tipo de arquitectura seleccionada?
- ¿Permite la arquitectura seleccionada implementar las funcionalidades deseadas?

Al concluir este trabajo se espera como **posible resultado**: Contar con un modelo de la arquitectura a desarrollar en el Sistema de Gestión de Credenciales con vistas a su desarrollo en software libre y de acuerdo a los nuevos requerimientos.

Dentro de las **tareas** que se proponen para dar solución a los objetivos planteados se encuentran:

- Realizar un estudio del estado del arte acerca de softwares similares al que se quiere desarrollar.
- Realizar un estudio del actual software del Sistema de Control de Acceso.
- Investigar acerca de las herramientas de software libre que podrán ser usadas.

- Intercambiar ideas con personas especializadas en el tema de las arquitecturas.
- Definir la nueva arquitectura ajustándose a las indicaciones propuestas por la Dirección de Informatización.
- Definir frameworks y entornos de desarrollo (IDEs) más apropiados para el desarrollo del sistema.
- Definir cómo será gestionada la seguridad en el sistema.
- Definir estándares de código a utilizar.
- Realizar la ingeniería de software correspondiente.
- Definir los patrones de diseño de software que se deberán utilizar.

Métodos de investigación científica empleados

Métodos teóricos

- **Deductivo:** Siguiendo reglas lógicas de deducción se llegan a nuevos conocimientos y predicciones que posteriormente son sometidas a verificaciones.

  - Su objetivo es arribar a la propuesta de arquitectura a partir de un estudio de los estilos y modelos arquitectónicos.
- **Histórico lógico:** Permite estudiar de forma analítica la trayectoria histórica de los fenómenos, su evolución y desarrollo.

  -Su objetivo en nuestro trabajo es el estudio de cómo ha evolucionado y se ha desarrollado la arquitectura de software.
- **Modelación:** Se crean abstracciones con vistas a explicar la realidad, se crea una relación entre el modelo y el objeto que se modela y permite predecir la respuesta de este trabajo.

  -Se modela la arquitectura con vistas a un mejor entendimiento de las tareas y los objetivos a cumplir.

Métodos empíricos

- **Entrevistas:** Con el fin de precisar detalles, tener una mejor idea de las herramientas que se utilizarán y de la arquitectura que se definirá.

- **Observación:** Para la percepción selectiva de las restricciones y propiedades del sistema y para el control de la evolución de la arquitectura inicial.

El presente trabajo se ha estructurado en tres capítulos:

Capítulo 1, Fundamentación teórica: donde se realiza un estudio de las tendencias, tecnologías y metodologías actuales con el fin de proponer las más adecuadas para la solución del problema. Además se da a conocer el estado del arte referente a los Sistemas de Gestión de Credenciales.

Capítulo 2, Definición de la arquitectura: en este capítulo se realiza la propuesta de la arquitectura para el Sistema Gestión de Credenciales, teniendo en cuenta lenguaje de programación, patrones de arquitectura, requisitos del sistema, etc.

Capítulo 3, Descripción de la arquitectura: en este capítulo se describe la arquitectura propuesta en el capítulo anterior.

Finalmente se exponen las conclusiones del trabajo, las recomendaciones y se hace referencia a las bibliografías utilizadas en la investigación.

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realizará un estudio detallado de los principales conceptos relacionados con la problemática, así como un breve estudio de las alternativas disponibles para la representación de la arquitectura. También se dará a conocer el estado del arte correspondiente a los sistemas de Gestión de Credenciales.

1.1.1 ¿Qué es la Arquitectura de Software?

No es novedad que ninguna definición de la arquitectura de software es respaldada unánimemente por la totalidad de los arquitectos. El número de definiciones circulantes alcanza un orden de tres dígitos casi llegando a cuatro.

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

La arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades; define de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea.

La arquitectura de software es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad. (1)

1.2 Estado del arte

En el mundo existe un gran número de sistemas que, de una forma u otra, se relacionan con la acreditación de personas y presentan un gran prestigio. Algunos de los más conocidos son los que se muestran a continuación.

1.2.1 CardFive

Es una aplicación potente y fácil de usar e instalar, con todas las características que se necesitan para generar tarjetas, con total integración y versatilidad en la interfaz. Con este programa multifunción, se puede acceder a una base de datos de diseños, codificar bandas magnéticas, procesar tarjetas inteligentes (smart cards), crear efectos y controlar el manejo de marcas de agua, hologramas y bandas para rascar (scratch ribbon).

Presenta poderosas funciones de edición de imágenes que permiten integrar cualquier formato de imagen desde cámaras digitales, en tiempo real, a través de video de Windows, o cualquier otra imagen en su base de datos.

Cardfive ofrece todas las características requeridas para crear y diseñar tarjetas usando la más alta tecnología, simplicidad de instalación y operabilidad. Puede imprimir usando el lenguaje DCL (Direct Command Language), usado por las marcas más afamadas de impresoras de tarjetas, y usado virtualmente por cualquier impresora a través de controladores, incluyendo impresoras de páginas. Permite utilizar los datos automáticamente mediante el acceso a una base de datos existente que posee la información necesaria para la impresión en la tarjeta plástica.

1.2.2 Evolis eMedia Profesional Software

Permite al usuario el diseño personalizado de credenciales para su posterior salida en las impresoras de tarjetas de PVC. En sus distintas versiones permite utilizar los datos automáticamente mediante el acceso a una base de datos existente, esta posee la

información necesaria para la impresión en la tarjeta plástica. Tiene más de 10 años de experiencia y ha sido usado en más de 70 países.

1.2.3 DataCard Id Work Id Cards Software

El software de identificación de ID Works permite mejorar cada aspecto del diseño de tarjeta y reportes, así como la producción. Ofrece gran poder y es extremadamente fácil de usar. Además de un diseño flexible y modular, permite seleccionar solamente los componentes específicos que se necesitan. DataCard es una compañía con más de 30 años de experiencia.

1.3 Tendencias, tecnologías y metodologías actuales

1.3.1 Metodologías de desarrollo del software

En un proyecto de desarrollo de software la metodología define “quién” está haciendo “qué”, “cuándo” y “cómo” para alcanzar un determinado objetivo. Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto. Las piedras angulares del proceso de desarrollo del software son: el proyecto, las personas y el producto; siendo las características del cliente, el entorno de desarrollo y las condiciones del negocio, elementos que influyen en el proceso.

1.3.1.1 Desarrollo basado en Rational Unified Process (RUP)

La metodología RUP, llamada así por sus siglas en inglés (Rational Unified Process), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Divide en cuatro fases el desarrollo del software:

- **Inicio**: El objetivo en esta etapa es determinar la visión del proyecto.
- **Elaboración**: En esta etapa el objetivo es determinar la arquitectura óptima.

- **Construcción**: En esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.
- **Transición**: El objetivo es llegar a obtener el release del producto.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

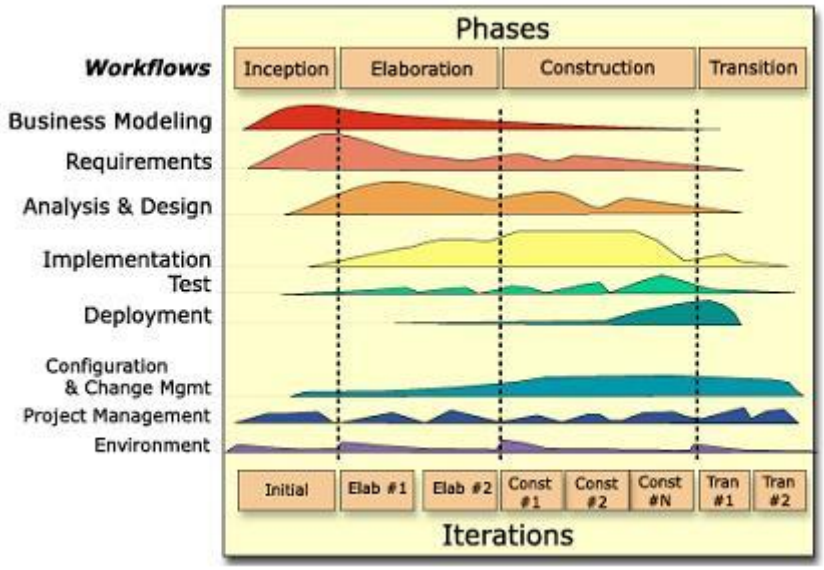

Figura 1 Fases e Iteraciones de la Metodología RUP

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevado bajo dos disciplinas:

Disciplina de Desarrollo

- **Ingeniería de Negocios**: Entendiendo las necesidades del negocio.
- **Requerimientos**: Trasladando las necesidades del negocio a un sistema automatizado.
- **Análisis y Diseño**: Trasladando los requerimientos dentro de la arquitectura de software.
- **Implementación**: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

- **Pruebas**: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

Disciplina de Soporte

- **Configuración y administración del cambio**: Guardando todas las versiones del proyecto.
- **Administrando el proyecto**: Administrando horarios y recursos.
- **Ambiente**: Administrando el ambiente de desarrollo.
- **Distribución**: Haciendo todo lo necesario para la salida del proyecto.

Los elementos de RUP son:

- **Actividades**: Son los procesos que se llegan a determinar en cada iteración.
- **Trabajadores**: Son las personas involucradas en cada proceso.
- **Artefactos**: Puede ser un documento, un modelo, o un elemento de modelo.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

Flujos de Trabajo

- **Modelamiento del negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.

- **Prueba (Testeo):** Busca los defectos a lo largo del ciclo de vida.
- **Instalación:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- **Administración del proyecto:** Involucra actividades con las que se busca desarrollar un producto que satisfaga las necesidades de los clientes.
- **Administración de configuración y cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a utilización/actualización concurrente de elementos, control de versiones, etc.
- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Cada flujo de trabajo tiene como resultado un modelo propuesto por RUP:

- Modelo de Casos de Uso del Negocio.
- Modelo de Objetos del Negocio.
- Modelo de Casos de Uso.
- Modelo de Diseño.
- Modelo de Despliegue.
- Modelo de Datos.
- Modelo de Implementación.
- Modelo de Pruebas.

Los modelos de casos de uso, diseño, despliegue e implementación son los más importantes para describir la arquitectura de un sistema, teniendo en cuenta que son los que proporcionan el desarrollo de las vistas de arquitectura.

El ciclo de vida de RUP tiene tres características fundamentales:

- **Guiado por casos de uso**: Los casos de uso reflejan las necesidades de los futuros usuarios, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. 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.

- **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 casos de uso relevantes desde el punto de vista de la arquitectura (casos de uso arquitectónicamente significativos). El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.
- **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 miniproyectos. Cada miniproyecto es 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. Cada iteración se realiza de forma planificada, es por eso que se dice que son miniproyectos.

El uso de esta metodología asegura que se produzca, desde sus primeras fases de desarrollo, un producto de calidad que cumpla con las características de funcionalidad, usabilidad y fiabilidad. Una particularidad de esta metodología es que en cada ciclo de iteración se hace exigente el uso de artefactos, siendo por este motivo una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.3.1.2 Extreme Programming (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo, pequeño equipo y “cuyo plazo de entrega era ayer”. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

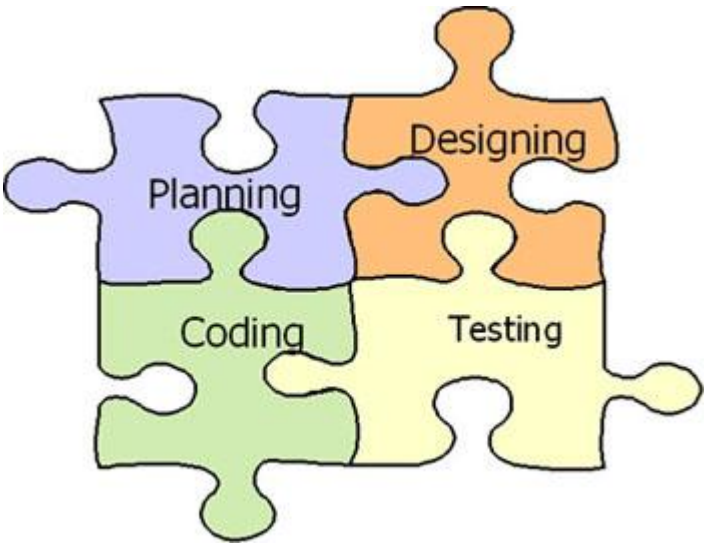

Figura 2 Metodología Extreme Programming

La metodología se basa en:

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos, de tal manera que se pueden hacer pruebas de las fallas que pudieran ocurrir en el futuro. Es como si se adelantara a obtener los posibles errores.
- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo.

Derechos del Cliente

- Decidir qué se implementa.
- Saber el estado real y el progreso del proyecto.
- Añadir, cambiar o quitar requerimientos en cualquier momento.
- Obtener lo máximo de cada semana de trabajo.
- Obtener un sistema funcionando cada 3 ó 4 meses.

Derechos del Desarrollador

- Decidir cómo se implementan los procesos.
- Crear el sistema con la mejor calidad posible.
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos.
- Estimar el esfuerzo para implementar el sistema.
- Cambiar los requerimientos en base a nuevos descubrimientos.

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente, del equipo de desarrollo, el cliente y los usuarios finales.

1.3.1.3 Microsoft Solution Framework (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Figura 3 Metodología MSF

MSF tiene las siguientes características:

- **Adaptable:** se parece a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos pequeños, de 3 ó 4 personas, así como también proyectos que requieran 50 personas o más.
- **Flexible:** se utiliza en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas en cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

- **Modelo de Arquitectura del Proyecto:** Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.
- **Modelo de Equipo**: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- **Modelo de Proceso*:*** Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases,

las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

- **Modelo de Gestión del Riesgo:** Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- **Modelo de Diseño del Proceso:** Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.
- **Modelo de Aplicación*:*** Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores. (2)

1.3.2 Tecnologías

1.3.2.1 Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos de alto nivel, gran rendimiento, sencillo, con un gran nivel de seguridad, multiplataforma y contiene las herramientas necesarias para desarrollar cualquier tipo de aplicación. El lenguaje en sí mismo toma muchas de sus sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

La sintaxis de Java se deriva en gran medida de C++, pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase.

A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener.

La independencia de su plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware.

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 las computadoras de usuarios de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación.

El diseño, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática como son:

- Dispositivos móviles y sistemas empotrados.
- Navegadores Web.
- Sistemas de servidor.
- Aplicaciones de escritorio.

1.3.2.2 Lenguaje de programación C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C y C++ y utiliza el modelo de objetos de la plataforma.NET, el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

1.3.2.3 Lenguaje de programación C++

El C++ es un lenguaje de programación que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

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*).
- Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel.
- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- 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, sin embargo es a su vez uno de los que menos automatismos trae porque obliga a hacerlo casi todo manualmente al igual que C, lo que dificulta mucho su aprendizaje.

1.3.3 Estilos Arquitectónicos

Un estilo arquitectónico o variante arquitectónica define a una familia de sistemas informáticos en términos de su organización estructural. Un estilo arquitectónico describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción. Permite sintetizar estructuras de soluciones que luego serán refinadas a través del diseño.

A continuación se mencionan algunos de los estilos arquitectónicos más utilizados en la actualidad.

1.3.3.1 Arquitectura en Capas

Los componentes de cada capa consisten en grupos de clases, las interacciones entre las capas ocurren generalmente por invocación de métodos, por definición los niveles de abajo no deben poder utilizar funcionalidad ofrecida por los niveles superiores.

Ventajas:

- Modularidad del sistema.
- Facilita la localización de errores.
- Mejora el soporte del sistema.
- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- El estilo admite muy naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. (3)

Desventaja:

- Puede ser difícil definir qué componentes ubicar en cada una de las capas.

1.3.3.2 Tubería y filtros

El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

El estilo arquitectónico tubería y filtros se puede utilizar cuando:

- Se puede especificar la secuencia de un número conocido de pasos.

- No se requiere esperar la respuesta asincrónica de cada paso.
- Se busca que todos los componentes situados corriente abajo sean capaces de inspeccionar y actuar sobre los datos que vienen de corriente arriba (pero no viceversa).

Ventajas:

- Es simple de entender e implementar.
- Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea.
- Fuerza un procesamiento secuencial.
- Los filtros se pueden empaquetar, y hacer paralelos o distribuidos.

Desventajas:

- El patrón puede resultar demasiado simplista, especialmente para orquestación de servicios que podrían ramificar la ejecución de la lógica de negocios de formas complicadas.
- No maneja con demasiada eficiencia construcciones condicionales, bucles y otras lógicas de control de flujo.
- Una desventaja adicional referida en la literatura sobre estilos concierne a que eventualmente pueden llegar a requerirse buffers de tamaño indefinido, por ejemplo en las tuberías de clasificación de datos.
- El estilo no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla.
- La independencia de los filtros implica que es muy posible la duplicación de funciones de preparación que son efectuadas por otros filtros (por ejemplo, el control de corrección de un objeto de fecha).

1.3.3.3 Arquitecturas Orientadas a Servicios (SOA)

Estas arquitecturas van de la mano de la expansión de los Servicios Web [1](en inglés Web Services) basados en XML y el elemento básico de una arquitectura SOA es el servicio. La Arquitectura Orientada a Servicios (en inglés Service-Oriented Architecture o SOA), define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

Desde el punto de vista arquitectónico, estas son las características del estilo:

- Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.
- Los componentes del estilo (o sea, los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran.
- Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente. En general no se pretende que un servicio recuerde nada entre un requerimiento y el siguiente.

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. Esta situación contrasta con lo que es el caso en estilos tubería y filtros, donde los filtros no necesitan poseer información sobre los otros filtros que constituyen el sistema. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan. También se presentan problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A. (3)

[1] Ver epígrafe 1.3.6.

1.3.4 Ambientes de Desarrollo

El ambiente de desarrollo (en inglés Development Environment) es algo imprescindible en la producción de software. Es donde se definen el conjunto de herramientas y tecnologías, versiones a usar y su integración, que intervienen en un proceso de desarrollo de software.

Ambiente o entorno de desarrollo integrado, (en inglés Integrated Development Environment, *IDE*) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación, o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, que mediante plugins se le puede añadir soporte de lenguajes adicionales.

1.3.4.1 Eclipse

Es un entorno de desarrollo integrado de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las “Aplicaciones de Cliente Liviano" basadas en navegadores.

Eclipse emplea módulos o plugins para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de

software. Adicionalmente al permitirle a Eclipse extenderse usando otros lenguajes de programación como son C, C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos.

Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador interno del lenguaje y un modelo completo de los archivos fuente. Esto permite técnicas avanzadas de refactorización y análisis de código.

Su principal aplicación es JDT, que es la herramienta para crear aplicaciones en Java, además como su plataforma está construida en base a plugins, permite que otras aplicaciones puedan ser integradas a Eclipse en forma de estos y los mismos son reconocidos automáticamente por el IDE al iniciarse.

1.3.4.2 C++Builder

Borland C++Builder está basado en las librerías VCL (Visual Component Library), creadas por y para Delphi, de hecho la mayoría de sus objetos son simples envoltorios (wrappers) en C++ de los objetos Delphi. Las aplicaciones creadas con C++Builder son basadas en eventos. Soporta conexiones a bases de datos a través de ODBC, ADO, y dbExpress. El entorno de desarrollo es completamente visual, y permite el desarrollo de aplicaciones de escritorio, aplicaciones webs (incluyendo AJAX), servicios de sistema, Servicios Web.

1.3.4.3 SharpDevelop

SharpDevelop es un entorno de desarrollo integrado para la plataforma .NET. Soporta las versiones del framework de Microsoft y de Ximian (MONO). Soporta desarrollo de interfaces, clases, namespaces y proyectos en C# y VB.NET, además de permitir importar los proyectos creados con Microsoft Visual Studio .NET.

1.3.4.4 NetBeans

NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de plugins para extender el IDE NetBeans. Es un producto libre y gratuito sin restricciones de uso.

1.3.5 Gestores de Bases de Datos

Los Sistemas Gestores de Bases de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.

1.3.5.1 PostgreSQL

Es un servidor de base de datos relacional orientada a objetos de software libre, sus principales características son:

- **Alta concurrencia**: Mediante un sistema denominado acceso concurrente multiversión, PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.
- **Amplia variedad de tipos nativos:** PostgreSQL provee nativamente soporte para números de precisión arbitraria, texto de largo ilimitado, figuras geométricas, direcciones IP, bloques de direcciones estilo CIDR, direcciones MAC, arreglos. Adicionalmente los usuarios pueden crear sus propios tipos de datos.

- **Bloques de código que se ejecutan en el servidor**: Pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos da, desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional. Las funciones pueden ser definidas para ejecutarse con los derechos del usuario ejecutor o con los derechos de un usuario previamente definido.

1.3.5.2 MySQL

Es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. El servidor de base de datos MySQL es muy rápido, fiable y fácil de usar. Trabaja en entornos cliente/servidor o incrustados. Además, funciona en diferentes plataformas y fue escrito en C y en C++, Es muy utilizado en aplicaciones Web como MediaWiki, Drupal o phpBB, etc. Su popularidad en aplicación Web está muy ligada a PHP, que a menudo aparece en combinación con MySQL. Es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones Web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para estos tipos de aplicaciones.

1.3.6 Herramientas de Modelado

1.3.6.1 Visual Paradigm

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.

Características:

- Soporte de UML versión 2.1.
- Modelado colaborativo con CVS y Subversion.
- Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería de ida y vuelta.
- Ingeniería inversa, código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, etc.
- Generación de código, modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso, entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Generación de componentes para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos.
- Soporte ORM, generación de objetos Java desde la base de datos.
- Generación de bases de datos, transformación de diagramas Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos, desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas, reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XMI.
- Integración con Visio, dibujo de diagramas UML con plantillas (stencils) de MS Visio.
- Editor de figuras.

1.3.6.2 Rational Rose

Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases.

La interfaz de Rational Rose (en adelante solo Rose) Está formada por los siguientes elementos principales:

- Browser ó Navegador, que permite navegar rápidamente a través de las distintas vistas del modelo y permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable.
- Ventana de documentación, para manejar los documentos del ítem seleccionado en cualquiera de los diagramas.
- Barra de herramientas, para acceder rápidamente a las acciones comunes a ejecutar para cada uno de los diagramas del modelo.

1.3.7 Sistemas de Control de Versiones

1.3.7.1 Subversion (SVN)

Es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular Concurrent Versions System [2](CVS), el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. Se sigue la historia de los archivos y directorios a través de copias y renombrados, las

[2] Ver epígrafe 1.3.7.2

modificaciones (incluyendo cambios a varios archivos) son atómicas. Permite selectivamente el bloqueo de archivos y es unos de los controladores más utilizados en proyectos de software libre.

1.3.7.2 Concurrent Versions System (CVS)

CVS es una aplicación informática que implementa un sistema de control de versiones, mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren.

Utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. Esto se hace para que eventualmente puedan trabajar con esa copia y más tarde ingresar sus cambios. Varios clientes pueden sacar copias del proyecto al mismo tiempo, los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña, solamente el ingreso de cambios requiere una contraseña.

Puede mantener distintas ramas de un proyecto. Por ejemplo, una versión difundida de un proyecto de programa puede formar una rama y ser utilizada para corregir errores. Todo esto se puede llevar a cabo mientras la versión que se encuentra actualmente en desarrollo y posee cambios mayores con nuevas características se encuentre en otra línea formando otra rama separada.

1.3.8 Frameworks de desarrollo

Un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, este puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional.

Fuera de las aplicaciones en la informática, un framework puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema complejo.

1.3.8.1 Hibernate

Es una herramienta de Mapeo objeto-relacional para la plataforma Java y disponible también para .Net con el nombre de NHibernate que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Como todas las herramientas de su tipo, busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos: el usado en la memoria de la computadora (orientado a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen, con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO (Programación Orientada a Objetos). Convierte los datos entre los tipos utilizados por Java y los definidos por SQL, genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

1.3.8.2 Struts

Es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón Modelo-Vista-Controlador (MVC), bajo la plataforma J2EE (Java 2, Enterprise Edition). Permite reducir el tiempo de desarrollo, su carácter de software libre y su compatibilidad con todas las plataformas en que Java Enterprise esté disponible, lo convierte en una herramienta altamente disponible.

Entre las características principales de Struts se pueden mencionar:

- Tiene una configuración de control centralizada.
- Presenta componentes de aplicación que son un mecanismo para compartir información bidireccionalmente entre el usuario de la aplicación y las acciones del modelo.
- Presenta librerías de entidades para facilitar la mayoría de las operaciones que generalmente realizan las páginas JSP.
- Contiene herramientas para validación de campos de plantillas bajo varios esquemas que van desde validaciones locales en la página hasta las validaciones de fondo hechas a nivel de las acciones.
- Permite que el desarrollador se concentre en el diseño de aplicaciones complejas como una serie simple de componentes del modelo y de la vista, intercomunicados por un control centralizado, diseñando de esta manera puede obtenerse una aplicación más consistente y más fácil de mantener.

1.3.9 Protocolo de acceso a datos SOAP

El protocolo simple de acceso a datos (SOAP), se encuentra en el núcleo de los Servicios Web. Este protocolo proporciona un estándar para empaquetar mensajes y es el primero de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo, incluso por aquellas que raramente cooperan entre si, tales como: Microsoft, IBM, SUN, Microsystems, SAP y Ariba.

Algunas de las Ventajas de SOAP son:

- **No está asociado con ningún lenguaje**: los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista.
- **No se encuentra fuertemente asociado a ningún protocolo de transporte**: La especificación de SOAP no describe como se deberían asociar los mensajes de este con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- **No está atado a ninguna infraestructura de objeto distribuido**: La mayoría de los sistemas de objetos distribuidos se pueden extender y ya lo están alguno de ellos para que admitan SOAP.
- **Aprovecha los estándares existentes en la industria**: Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas, optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema, no define un medio de trasporte de los mensajes, los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes.
- **Permite la interoperabilidad entre múltiples entornos**: se desarrollo sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas.

1.3.10 Servicios Web

Existen múltiples definiciones sobre lo que son los Servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web, estas aplicaciones o tecnologías intercambian datos entre sí.

Ventajas:

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los Servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta y se garantiza la plena interoperabilidad entre aplicaciones.

1.3.11 Lenguaje Unificado de Modelado (UML)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

1.3.12 Patrones de Diseño

Un patrón de diseño es:

- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.
- Un lenguaje de programación de alto nivel.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.

Los patrones de diseño pretenden:

- Proporcionar elementos reusables en el diseño de aplicaciones.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores, condensando conocimiento ya existente.

Los patrones de diseño se utilizan cuando se percibe un problema en el proyecto, algo que debería resultar sencillo no lo es. No es obligatorio utilizar los patrones siempre, solo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable.

1.4 Conclusiones

En este capítulo se han abordado los temas relacionados con el estado de arte, conceptos fundamentales, metodología, tecnologías y herramientas que se pueden emplear en la arquitectura que se propondrá en este trabajo, con el objetivo de dar una solución a la situación existente.

CAPÍTULO 2 DEFINICIÓN DE LA ARQUITECTURA

2.1 Introducción

La arquitectura de software es el esqueleto o base de una aplicación, en esta se analiza la aplicación desde varios puntos de vista. En la arquitectura aparecen los artefactos más importantes para establecer un esquema de cómo deben ser los próximos artefactos a construir. La arquitectura permite guiar al equipo de desarrollo a través del ciclo de vida del sistema.

En este capítulo se propone la metodología, herramientas y patrones a utilizar en la implementación del sistema.

Lo que se persigue obtener con la elaboración de dicho documento es:

- Una mejor comprensión del sistema.
- La organización del desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.
- Capacitar a los desarrolladores, directivos, clientes y otros usuarios para que comprendan con suficiente claridad el proceso y participen en este.

2.2 Descripción del Sistema Propuesto

La propuesta de solución de este trabajo es un sistema capaz de gestionar grupos de credenciales, el especialista de acreditación podrá seleccionar un grupo y que se muestren las credenciales que están en el mismo, pudiendo filtrar por cada uno de los campos que componen la credencial, ver el historial de credenciales que ha tenido una persona hasta el momento, cargar la foto para la confección de las credenciales tanto de una base de datos como de una localización física, seleccionar una credencial, mostrándose una vista previa de cómo va a quedar esta después de imprimirla y así poder corregir cualquier detalle.

Con la elaboración de esta solución se estarían agregando funcionalidades que el Sistema Gestión de Credenciales anteriormente carecía que son muy necesarias para llevar un mejor control de la gestión e impresión de credenciales en la UCI y será desarrollado con herramientas de software libre logrando convertirlo en un sistema completamente multiplataforma.

Las otras soluciones que se han encontrado a lo largo de este trabajo presentan gran prestigio mundialmente pero no permiten manejar el proceso de impresión de credenciales en entornos complejos, como el de nuestra Universidad, además estos son muy caros y solamente funcionan obteniendo la información de una base de datos que se le indique, pero cuando la información se encuentre en diferentes bases de datos o se necesita utilizar Servicios Web como en nuestro caso, estos sistemas no son capaces de realizar esta tarea eficientemente, ellos se centran más en el diseño de las credenciales. Por eso se sigue adelante con la propuesta de solución.

2.3 Propuesta de la arquitectura ajustándose a las indicaciones de la dirección de Informatización.

Esta propuesta se basará en las tendencias del país y de la UCI a incorporarse dentro del mercado del desarrollo de software libre. A continuación se exponen las metodologías, tecnologías, herramientas, patrones y lenguajes que serán utilizados en la construcción de la aplicación propuesta. Además se dan elementos del por qué fueron escogidas precisamente estas y no otras.

2.3.1 ¿Qué metodología usar?

Se usará RUP porque aparte de que constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, tiene características que las otras no poseen lo que la hace más cómoda para el desarrollo de un software.

Divide en fases el desarrollo del software (Inicio, Elaboración, Construcción, Transición), cada una de estas fases es desarrollada mediante un ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

Se han agrupado las actividades en grupos lógicos, se va desarrollando el software por iteraciones e incrementos, es guiado por casos de uso, los que reflejan las necesidades de los futuros usuarios, centrado en la arquitectura, la cual muestra la visión común del sistema completo.

Permite que en la medida que se vaya construyendo el software por ciclos se puedan detectar errores, una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes y utilizada para alcanzar un grado de certificación en el desarrollo del software.

En fin el uso de esta metodología asegura que se produzca desde las primeras fases de desarrollo del software, un producto de calidad que cumpla con las características de funcionalidad, usabilidad y fiabilidad.

2.3.2 Estructura organizativa de la arquitectura

Se quiere diseñar para el Sistema Gestión de Credenciales una arquitectura flexible, con un bajo acoplamiento entre sus componentes, organizada, que a la hora de cambiar o agregar algo en el sistema la tarea no se torne engorrosa, como pudiera ser el caso de cambiar la impresora, la base de datos, agregarle Web Services, etc.

La arquitectura de software es la organización de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución. Los estilos arquitectónicos son el nivel de abstracción mayor para estructurar el sistema, la elección del mismo está dada por el tipo de aplicación que se vaya a desarrollar.

A partir de los elementos mencionados anteriormente, la arquitectura propuesta para desarrollar el sistema es el estilo arquitectura en capas. Se propone implementar una arquitectura en capas porque esta es la que más se adapta al problema existente, en esta arquitectura a cada nivel se le confía una misión simple, lo que permite el diseño de una arquitectura escalable, es decir se puede ampliar con facilidad en caso de que las necesidades del sistema aumenten. El desarrollo se puede llevar a cabo en varios niveles y en caso de algún problema o cambio se ataca al nivel requerido sin tener que revisar entre código mezclado, reduce las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

Este estilo permite ganar en organización del sistema, además soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales, proporciona una amplia reutilización, facilita la corrección de errores y se obtiene al final una solución altamente flexible y reutilizable como la que se quiere para el Sistema Gestión de Credenciales.

En la actualidad muchos sistemas están basados en arquitecturas de dos capas, denominadas generalmente nivel de aplicación y nivel de la base de datos. Estas tienen algunos inconvenientes como es la recarga del nivel de aplicaciones, es por ello que existe una fuerte y avanzada tendencia a adoptar arquitecturas en más de dos capas. En este caso se propone 4 capas para tener todo separado, organizado y reducir las dependencias, las capas serían: presentación, lógica de negocio, acceso a datos y una capa de impresión para cuando haga falta cambiar algo referente a la impresión de credenciales.

2.3.3 Patrones de diseño a utilizar

La primera regla de los patrones de diseño coincide con la primera regla de la optimización: retrasar, del mismo modo que no es aconsejable optimizar prematuramente, no se deben utilizar patrones de diseño antes de tiempo. Es mejor implementar algo primero y asegurarse de que funciona, para luego utilizar el patrón de diseño para mejorar sus flaquezas. (4)

Los patrones permiten que algunos aspectos de la estructura del sistema puedan cambiar independientemente de otros aspectos. Facilitan la reusabilidad, extensibilidad y mantenimiento.

Un patrón es un esquema o microarquitectura que supone una solución a problemas (dominios de aplicación) semejantes. También conviene distinguir entre un patrón y una arquitectura global del sistema. Por decirlo en breve, es la misma distancia que hay entre el diseño de un componente (o módulo) y el análisis del sistema, es la diferencia que hay entre el aspecto micro y el macro, por ello, en ocasiones se denomina a los patrones como "microarquitecturas". En resumen, un patrón es el denominador común, una estructura común que tienen aplicaciones semejantes. (5)

En esta arquitectura propuesta se usará el patrón Decorador (Decorator) y el Data Access Object (DAO), seguidamente se explica el por qué de estos:

2.3.3.1 Patrón Decorador

Permite que cuando se tenga que agregar nuevas funcionalidades no se tenga que añadir más clases y subclases ya que se acabaría con una jungla de clases y subclases. Con este patrón se trata de evitar este efecto de herencia-jungla, ya que se asigna dinámicamente nuevas responsabilidades a un objeto. Es una alternativa más flexible a crear subclases para extender la funcionalidad de una clase. De esta forma se añaden atributos o comportamiento adicional a un objeto concreto, no a una clase.

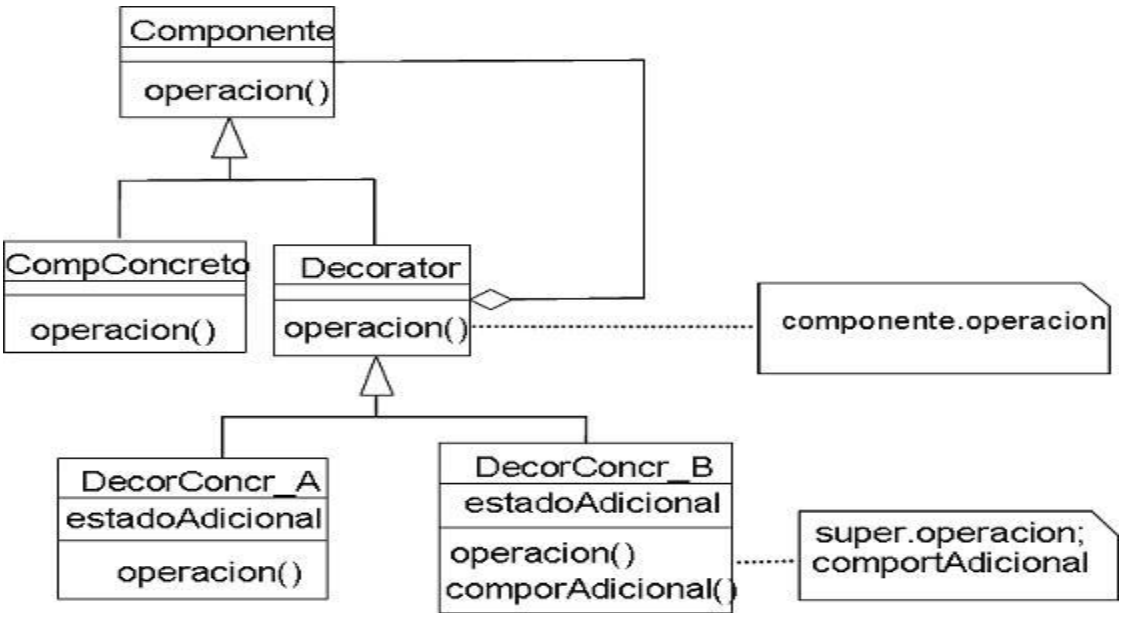

Figura 4 Estructura del Patrón Decorador (Decorator**)**

2.3.3.2 Patrón Data Access Object (DAO)

Suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. El DAO tiene un interfaz común, sea cual sea el modo y fuente de acceso a datos.

El problema que viene a resolver este patrón es el de contar con diversas fuentes de datos (base de datos, archivos, servicios externos, etc), de tal forma que se encapsula la forma de acceder a la fuente de datos.

Este patrón surge históricamente de la necesidad de gestionar una diversidad de fuentes de datos, aunque su uso se extiende al problema de encapsular no sólo la fuente de datos, sino además ocultar la forma de acceder a los datos. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento.

El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes, como el interface expuesto por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite adaptarse a diferentes esquemas de

almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos.

Este patrón es utilizado para:

- Atraer y encapsular los accesos.
- Gestionar las conexiones a la fuente de datos.
- Obtener los datos almacenados.

Ventajas:

- **Permite la transparencia**: Los objetos de negocio pueden utilizar la fuente de datos sin conocer los detalles específicos de su implementación. El acceso es transparente porque los detalles de la implementación se ocultan dentro del DAO.
- **Permite una migración más fácil**: Una capa de DAOs hace más fácil que una aplicación pueda migrar a una implementación de base de datos diferente. Los objetos de negocio no conocen la implementación de datos subyacente, la migración implica cambios sólo en la capa DAO.
- **Reduce la complejidad del código de los objetos de negocio**: Como los DAOs manejan todas las complejidades del acceso a los datos, se simplifica el código de los objetos de negocio y de otros clientes que utilizan los DAOs. Todo el código relacionado con la implementación está dentro del DAO y no en el objeto de negocio. Esto mejora la lectura del código y la productividad del desarrollo.
- **Centraliza todos los accesos a datos en una capa independiente**: Como todas las operaciones de acceso a los datos se ha delegado en los DAOs, esto se puede ver como una capa que aisla el resto de la aplicación de la implementación de acceso a los datos. Esta centralización hace que la aplicación sea más sencilla de mantener y de manejar.

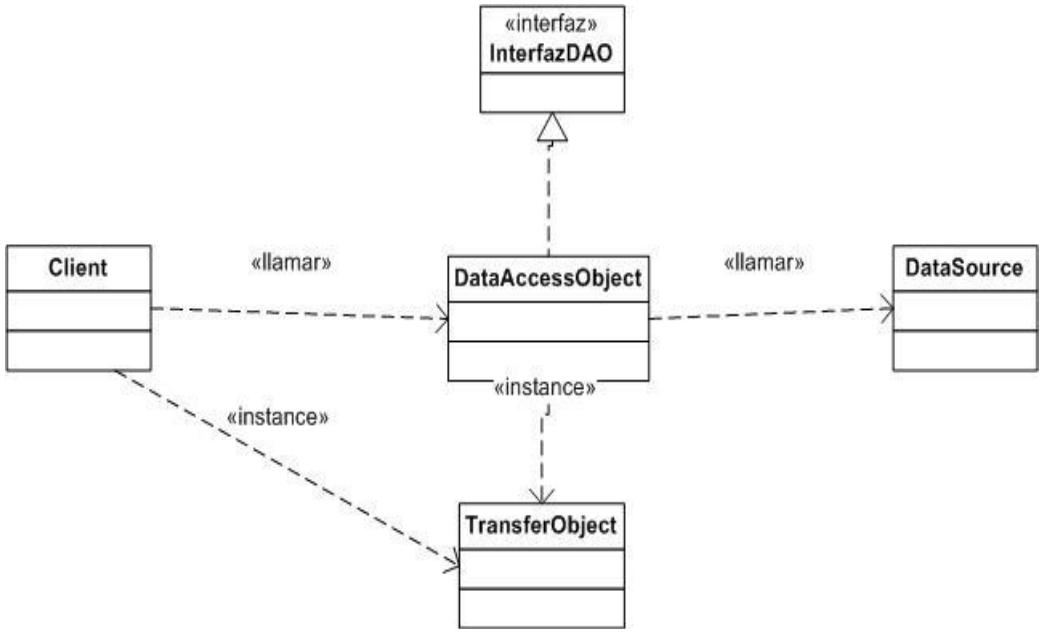

Figura 5 Estructura del Patrón DAO

2.3.4 Patrones de idiomas o estándares de código a utilizar

Estos estándares se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como estándares de codificación y proyecto, describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y predictibilidad.

- Los nombres de las clases comienzan con mayúscula. Si el nombre es compuesto de varias palabras, por lo general se escriben juntas y con el primer carácter en mayúscula.
  Ejemplo: class MyClase {…}
- El nombre de las variables y clases debe ser significativo, permitiendo tener una idea de su función con solo leerlo. Esto no siempre es posible, pero siempre es mejor dar un nombre parecido al objetivo que un nombre cualquiera.
- Las instancias de las clases, los objetos, comienzan con minúscula y en caso de estar compuesto de varias palabras, el resto comenzarán con mayúscula:
  Ejemplo: public String myCadena;

- Se deben emplear espacios o tabuladores para que se distingan con claridad los distintos bloques de sentencias. Es recomendable emplear espacios en blanco debido a las distintas interpretaciones que hacen los editores sobre las tabulaciones.
- Las líneas de código no deben ser muy extensas, por lo general inferiores a 80 caracteres. En caso de necesitar más, se bajan a la siguiente línea. Para así poder aprovechar un cierre de paréntesis o una coma, etc.
- No se puede emplear clases declaradas dentro de otra clase.
- El estilo de los comentarios debe ser como el estilo de comentarios para C (***/* */*** ó ***//***).
- Es muy recomendable documentar cada uno de los métodos y variables relevantes de la clase.
- Es obligatorio detallar el uso de cada constante.
- Las variables siempre se deben inicializar para evitar las excepciones.

2.3.5 ¿Lenguaje de programación a utilizar?

Java porque ha logrado un significativo avance en el mundo del software y por dos elementos claves que diferencian a este lenguaje de los otros desde un punto de vista tecnológico:

Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos, con una sintaxis fácilmente accesible, un entorno robusto y agradable y proporciona un conjunto de clases potentes y flexibles.

Este lenguaje es de gran rendimiento, sencillo con un gran nivel de seguridad, multiplataforma y contiene las herramientas necesarias para desarrollar cualquier tipo de aplicación. Hoy en día cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas.

Otro factor que lleva a la selección de este lenguaje es, la estrategia de hacer uso de tecnologías libres que lleva a cabo la UCI y porque así lo decidió la Dirección de Informatización por la cual se rige el proyecto Identificación y Control de Acceso.

2.3.6 ¿Qué IDE utilizar?

Es muy sugerido utilizar ambientes de desarrollo para obtener aplicaciones robustas ya que estos ambientes te ayudan a organizar, probar y escribir el código, aumentando la productividad y la calidad de la aplicación que se va a producir.

Hay varios ambientes de desarrollo para Java, pero los principales ambientes de desarrollos gratuitos son: Eclipse y NetBeans.

Se usará Eclipse porque NetBeans es demasiado lento en comparación con Eclipse, le falta velocidad y un editor de código un poco más fluido, Eclipse tiene un muy buen soporte de refactorización del cual NetBeans carece, las técnicas de refactorización básicas están presentes en ambos IDEs pero el Eclipse ha adelantado mucho más sus funciones de refactorización, además es mucho mejor que NetBeans en fluidez de la interfaz, integración con otros lenguajes de programación, mejor a la hora de compilar / ejecutar.

2.3.7 Plugins para Eclipse que se usarán

Un plugin es una aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal. En el caso particular de Eclipse no son más que un conjunto de clases que permiten hacerlo más extensible.

Los plugins que se usarán son los siguientes:

- **Visual Editor para Java**: Es un editor de código que brinda una ayuda para el diseño de aplicaciones que contienen interfaz gráfica de usuario (GUI). Permite editar el código fuente y el diseño simultáneamente además puede ser utilizado no sólo como una herramienta para generar código, sino como un editor para mostrar el efecto de las modificaciones de código fuente durante el desarrollo.
- **Subclipse**: Es un plugin para Eclipse, el cual permite conectarse a repositorios de Subversion y así poder obtener copias de los proyectos que se encuentran en el repositorio del mismo, hacer cambios y posteriormente mandarlos. Permite además

hacer operaciones sobre el repositorio directamente, operaciones de sincronización, actualización, también permite el uso de bloqueos a recursos para que otros usuarios no puedan modificarlo.

- **Hibernate Tools**: Es un conjunto de herramientas para trabajar con el framework Hibernate. Presenta asistentes como generador de archivos de configuración rápidamente, configuración de la consola, entre otros. La perspectiva de consola de Hibernate permite configurar conexiones a bases de datos, provee visualización de clases y sus relaciones, admite además ejecutar preguntas en formato del lenguaje de preguntas de Hibernate interactivamente contra la base de datos y mostrar los resultados. Este plugin tiene un editor de mapeos que es un editor para los archivos de mapeos XML de Hibernate, soportando auto completamiento y sintaxis resaltada. Soporta además completamiento semántico para nombres de clases, propiedades, tablas y columnas. La Ingeniería inversa es su característica más importante, permitiendo generar las clases del modelo de dominio, los archivos de mapeos de Hibernate y documentación en formato HTML, a partir de una base de datos.

2.3.8 ¿Qué Gestor de Base de Datos usar?

El Sistema Gestión de Credenciales maneja grandes cantidades de datos debido al considerable número de credenciales que se imprimen diariamente. Todos estos datos deben poder ser guardados en una base de datos.

Se usará como servidor de base de datos PostgreSQL que es el que más se adapta a las necesidades del sistema ya que este soporta grandes volúmenes de datos que es lo que se necesita en este sistema. Posee una gran escalabilidad, es capaz de ajustarse a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta, se dice que ha llegado a soportar el triple de carga de lo que soporta MySQL que no es viable para su uso con grandes bases de datos, ya que no implementa una buena escalabilidad.

PostgreSQL Implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en las que MySQL no podría ya que carece de soporte para transacciones, rollback's y subconsultas.

PostgreSQL tiene la capacidad de comprobar la integridad referencial, MySQL no maneja la integridad referencial, lo que hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para programadores que provienen de otros gestores que sí que poseen esta característica.

Los dos gestores son magníficos, simplemente se trata de escoger el más conveniente en cada caso. En este caso el más conveniente es PostgreSQL que permite almacenar grandes volúmenes de datos.

2.3.9 ¿Qué herramienta de modelado utilizar?

Se usará Visual Paradigm y no Rose porque este presenta ventajas que Rose no posee, tales como:

- Permiten incorporar los dibujos de Visio en cualquier diagrama de UML a diferencia de otras herramientas de modelado que apoyan sólo diagramas de UML normales como es el caso de Rose.
- Con el diseño del diagrama automático en Visual Paradigm, se puede arreglar los diagramas desarreglados con unos clics del ratón. Se puede escoger el diseño en dependencia de la naturaleza del diagrama.
- Permite la importación y exportación de ficheros XML.
- Permite editar figuras.
- Permite la ingeniería inversa de bases de datos, desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- Es una herramienta de modelado que apoya las tarjetas de CRC.
- Permite invertir código fuente de programa, binario y archivos ejecutables en modelos al instante.

- No importa qué sistema operativo se está usando, se puede ejecutar en muchas plataformas como Windows, Linux, etc.

2.3.10 ¿Qué controlador de versiones usar?

Se usará el controlador de versiones Subversion que es unos de los controladores más utilizados en proyectos de software libre, este se diseñó para remplazar a uno muy popular como lo era el CVS que posee muchos errores.

Algunos ejemplos que demuestran que Subversion es más eficiente:

- **Transparencia al eliminar y cambiar nombres de archivos**: En CVS requiere intervención manual en el depósito para lograrlo, Subversion contempla esta deficiencia y la corrige con éxito.
- **Copias diferenciales de archivos binarios**: Subversion es capaz de mantener un control diferencial sobre cualquier archivo binario del depósito, reduciendo el consumo de espacio, esto contrastado con CVS que requiere archivar copias completas de un archivo binario cada vez que éste cambia.
- **Copias ligeras sobre ramificaciones**: La generación de ramificaciones en CVS además de ser un proceso involucrado implica la generación de una copia nueva en el depósito, mecanismo que hace crecer exponencialmente el tamaño del depósito, Subversion independientemente del número de ramificaciones creadas mantiene un árbol diferencial de cambios, minimizando así el espacio consumido en el depósito.
- En Subversion se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos).
- Subversion maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).

2.3.11 Framework de desarrollo que se utilizará

Se usará Hibernate que es un framework de software libre, permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la programación orientada a objetos que es mucho más fácil. No necesita ningún entorno especial para ejecutarse, provee un lenguaje de consultas orientado a objetos que facilita la ejecución de estas, crea una capa de abstracción que permite migrar de gestor de base de datos sin cambiar una sola línea de código, facilita el desarrollo de la lógica de acceso a datos en las aplicaciones. Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente.

2.3.12 Versiones de las herramientas que se usarán

2.3.12.1 Visual Paradigm UML 6.0

Esta versión brinda las siguientes ventajas:

- **Permite enriquecer la documentación del modelo con imágenes**: Además de apoyar la documentación de texto enriquecido para los modelos y esquemas, puede visualizar su concepto mediante la incorporación de imágenes en la documentación.
- **Permite guardar o cargar plantilla para la documentación de modelo**: Se puede usar plantillas para la documentación previa a la reutilización, permite definir la documentación, estructura para ahorrar tiempo y esfuerzo.
- **Apoya el modelado visual con los procedimientos almacenados**: Puede crear y editar procedimientos almacenados fácilmente en un entorno intuitivo.
- Permite generar código Java que ayudan a manipular los registros de bases de datos relacionales en una base orientada a objetos, además permite generar código fuente para muchos lenguajes de programación.

2.3.12.2 Postgre 8.2

Se utilizará como Sistema Gestor de Bases de Datos la versión Postgre 8.2 que presenta las siguientes ventajas:

- Mejora el rendimiento en alrededor de un 20%. Las mejoras incluyen, ordenamientos en memoria y en disco más rápido, mejor escalabilidad en sistemas multi-procesador, mejor optimización de consultas sobre datos particionados, cargas masivas más rápidas y outer joins considerablemente acelerados.
- Permite a los administradores crear fácilmente una copia para recuperación inmediata (failover) de su cluster de bases de datos.
- Permite la construcción de índices que puede ocurrir mientras las aplicaciones escriben en las tablas de la base de datos, permitiendo el afinamiento del rendimiento sin afectar la disponibilidad.
- Se puede definir un valor de retorno para los insert/update/delete, en los updates se puede utilizar múltiples columnas con una lista de valores. Ejemplo: update agenda set (teléfono, nombre) = (‘120202’,’horaciod’) where idagenda=1.

2.3.12.3 Subversion 1.4.6

Como Controlador de versiones se usará Subversion 1.4.6, este presenta las siguientes ventajas:

- Esta versión presenta una herramienta llamada SVNsync, Esta proporciona la capacidad de replicar la historia de un repositorio a otro, el origen y el destino de los depósitos puede ser local, remoto, o cualquier combinación de estos.
- Alto rendimiento en la copia de trabajos enormes, el formato de copia que presenta esta versión permite que los clientes busquen rápidamente una copia de trabajo, detectar las modificaciones de archivos y tratar con archivos de gran tamaño.
- Presenta mejoras en el manejo de archivos binarios.

- Presenta más de 40 nuevas soluciones a fallos (ejemplo, si ocurre algún problema posee un sistema de recuperación automático).
- Presenta unos interruptores y opciones de líneas de comandos para el cliente que lo hacen más fácil y cómodo que las otras versiones.
- Esta versión puede leer los depósitos creados por las versiones anteriores.
- Se puede utilizar en cualquier lenguaje de programación y para cualquier tipo de proyecto.
- Es un superconjunto de todas las versiones anteriores y se considera la actual mejor puesta en libertad, cualquier cosa en las versiones anteriores también está en esta.
- Presenta además otras ventajas importantes, contiene las soluciones a fallos que no están presente en cualquier versión anterior.

2.3.12.4 Hibernate 3.2

Como framework de desarrollo se usará Hibernate 3.2 el mismo tiene las siguientes ventajas:

- Presenta una curva de aprendizaje muy corta ya que es un framework simple.
- Los resultados con Hibernate 3.2 siempre están entre los mejores en cualquier comparativa en cuanto a rendimiento.
- Permite la generación de código mediante plugins para Eclipse y otros IDES.
- Los Archivos de Mapeo (XML) en esta versión son opcionales, en las versiones anteriores son obligatorios.
- Soporta el mapeo de relaciones complejas de objetos.
- Soporte de claves simples y compuestas.
- Mapeo de superclases y relaciones de herencia de clases.
- Actualizaciones y eliminación de datos en cascada.

En cuanto a configuración de acceso a base de datos:

- Puede usar su propio archivo de configuración o hacer uso de la configuración de otro framework.

- Puede trabajar sin un servidor de aplicaciones, tanto en desarrollos de aplicaciones livianas, como en desarrollos de aplicaciones complejas.
- Soporta mediante extensiones de Hibernate y mediante el uso de otros frameworks como Spring el manejo de seguridad, manejo de transacciones e inyecciones de dependencias.

Esta versión es completamente aceptada por la industria, Existe un gran número de desarrolladores Java en el mundo entero que lo apoyan.

2.3.12.5 Eclipse 3.2 (Callisto)

Se usará Eclipse 3.2 porque para la última versión (Eclipse 3.3) a pesar de que es más rápido que Callisto, que integra más proyectos (versiones de 21 proyectos) y es un conjunto de las versiones anteriores, hay plugins que no son compatibles con esta, como es el caso del Visual Editor, que es para facilitar el trabajo con aplicaciones que contienen interfaz gráfica.

Eclipse 3.2 es un completo entorno de desarrollo para los programadores que trabajan en Java, permite programar, desarrollar y compilar aplicaciones en Java, sitios Web, programas en C++, etc. Proporciona un ciclo de desarrollo más transparente y predecible. Integra 10 proyectos al mismo tiempo, aunque no es una unificación de los proyectos.

Con esta versión se pueden utilizar todas las herramientas que un buen IDE ofrece. Todo esto además de su agradable interfaz hacen de Eclipse 3.2 un programa confiable y muy utilizado por los programadores.

Como plugins para esta versión se usará Subclipse 1.2.4 actual versión disponible para eclipse 3.2 que permite conectarse al controlador de versiones Subversion que es el que se usará. Hibernarte Tool 3.2 que es también compatible con Eclipse 3.2 y está relacionado con el framework Hibernate, con este plugins se puede implementar toda la capa de persistencia de una aplicación en cuestión de minutos y el Visual Editor 1.2.3 que se integra elegantemente con esta versión de Eclipse y facilita el trabajo con aplicaciones que contienen interfaz gráfica de usuario.

2.3.13 Seguridad del Sistema

Este sistema será usado por una persona, es decir, no existirá roles, ni varios usuarios, sólo una persona se encargará de trabajar con el sistema que es la especialista de acreditación. La aplicación que se propone es de escritorio, o sea no acceden personas a ella por la red, para acceder al sistema será a través de usuario y dominio y para asegurar la transmisión o almacenamiento seguro de información con los dispositivos de almacenamientos se usará la encriptación de los datos con el propósito de transformar la información, donde el contenido de la misma lucirá como un conjunto de caracteres extraños, sin ningún sentido o lógica de lectura.

Existen dos tipos de cifrados de mensajes el cifrado simétrico y el cifrado asimétrico, el cifrado simétrico necesita siempre utilizar nuevas claves para cada información que se quiera enviar, con este no se puede firmar mensajes por lo que el emisor no conoce el origen de los datos, tiene graves problemas con la distribución de las claves, con estos cifrados se corre el riesgo de que la información trasmitida pueda ser interceptada por otra persona y como la clave para descifrar el mensaje es la misma con el que se cifra y es conocida por muchas personas, la información puede ser desencriptada.

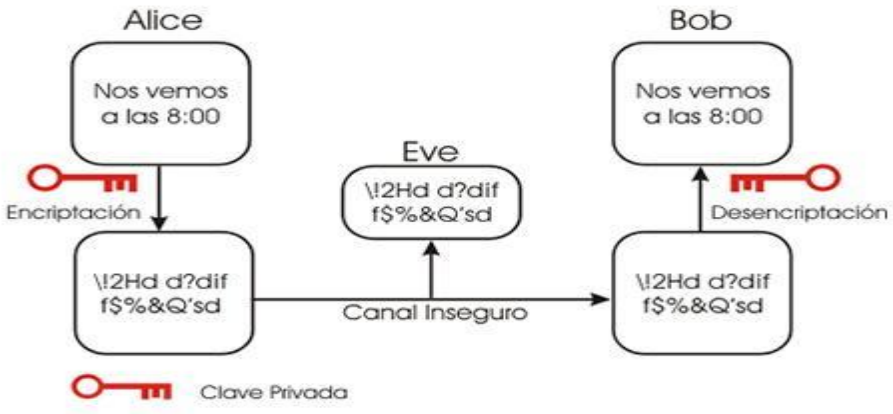

Figura 6 Cifrado simétrico

Se usará el cifrado asimétrico que resuelve el problema del intercambio de claves ya que este usa dos claves, una pública y otra privada. Las dos claves pertenecen a quien se le envía el mensaje, una clave es pública y la puede conocer cualquiera, la otra clave es privada y el propietario debe guardarla para que nadie tenga acceso a ella ya que en esta radica la seguridad del sistema. El remitente usa la clave pública del destinatario para cifrar el mensaje, y una vez cifrado, sólo la clave privada del destinatario podrá descifrar este mensaje.

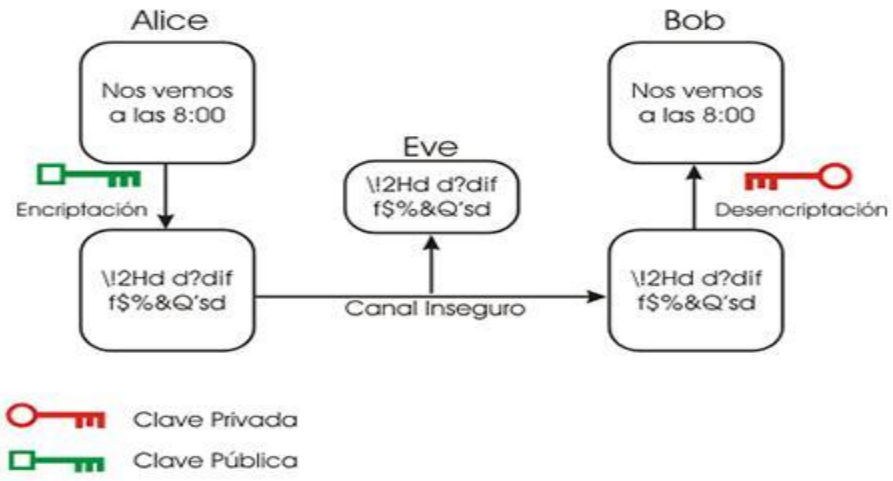

Figura 7 Cifrado asimétrico

Dentro de los cifrados asimétricos, los algoritmos más utilizados son:

ECC (Elliptical Curve Cryptography)
RSA (Rivest, Shamir, Adleman)

El ECC (Elliptical Curve Cryptography) es un algoritmo que se utiliza muy poco, aunque es equivalente en fortaleza al RSA pero requiere menos potencia de cálculo en sus operaciones, es lento y no posee una gran seguridad por lo que se corre riesgo de poder ser descifrado el mensaje, por tal motivo se usará el RSA que es hoy en día el algoritmo de mayor uso en la encriptación asimétrica, es el de mayor seguridad de los algoritmos asimétricos y es el más rápido de ellos, se puede usar para el manejo de firmas, con las que se puede comprobar quien es el emisor de los datos.

Los mensajes enviados usando el algoritmo RSA se representan mediante números y su funcionamiento se basa en el producto de dos números primos grandes (mayores que10^{100}) elegidos al azar para conformar la clave de descifrado, Emplea expresiones exponenciales en aritmética modular. Basado en la exponenciación modular de exponente y módulo fijos, el sistema RSA crea sus claves de la siguiente forma:

Se buscan dos números primos lo suficientemente grandes: **p** y **q** (de entre 100 y 300 dígitos).
Se obtienen los números **n = p * q y Ø = (p-1) * (q-1)**
Se busca un número de tal forma que no tenga múltiplos comunes con **Ø**.
Se calcula **d = e-1mod Ø**, con **mod =** resto de la división de números enteros. Con estos números obtenidos, **n** es la clave pública y **d** es la clave privada. Los números **p**, **q** y **Ø** se destruyen. También se hace público el número **e**, necesario para alimentar el algoritmo. El cálculo de estas claves se realiza en secreto en la máquina en la que se va a guardar la clave privada.

En cuanto a la longitud de las claves, el sistema RSA permite longitudes variables, pero se utilizarán claves de no menos de 1024 bits ya que se han roto claves de hasta 512 bits, aunque se necesitaron más de 5 meses y casi 300 ordenadores trabajando juntos para hacerlo.

La seguridad de este algoritmo radica en que no hay maneras rápidas conocidas de factorizar un número grande en sus factores primos utilizando computadoras tradicionales, se basa en el uso de dos claves diferentes, claves que poseen una propiedad fundamental: una clave puede desencriptar lo que la otra ha encriptado, las claves pública y privada tienen características matemáticas especiales, de tal forma que se generan siempre a la vez, por parejas, estando cada una de ellas ligada intrínsecamente a la otra, en ser una función computacionalmente segura, ya que si bien realizar la exponenciación modular es fácil, su operación inversa, la extracción de raíces de módulo **Ø** no es factible a menos que se conozca la factorización de la clave privada del sistema, y con este cifrado se garantiza:

Integridad: Es decir que el mensaje no sea alterado.

Confidencialidad: Garantiza que solo quienes posean la llave puedan descifrar el mensaje.

Autenticidad: Permite comprobar que el emisor es quien dice ser y no otro.

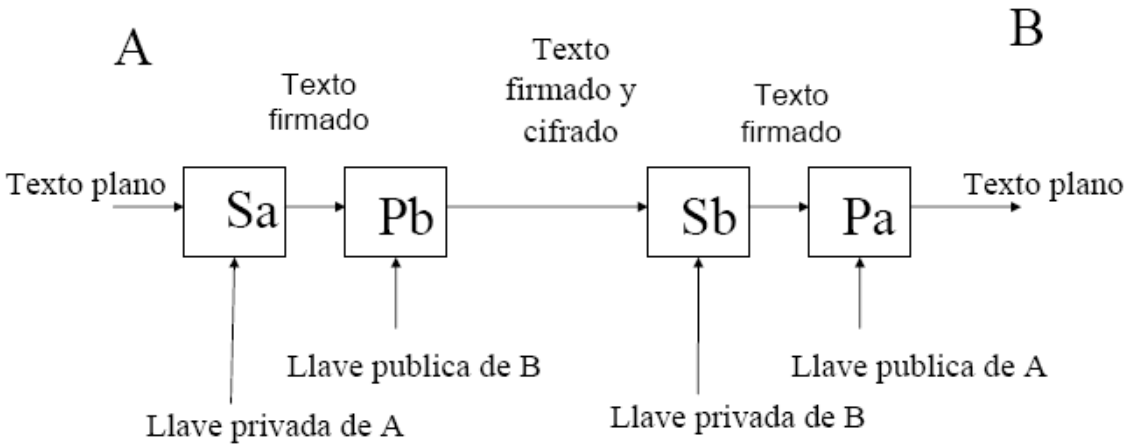

Figura 8 Firmado y cifrado de un mensaje

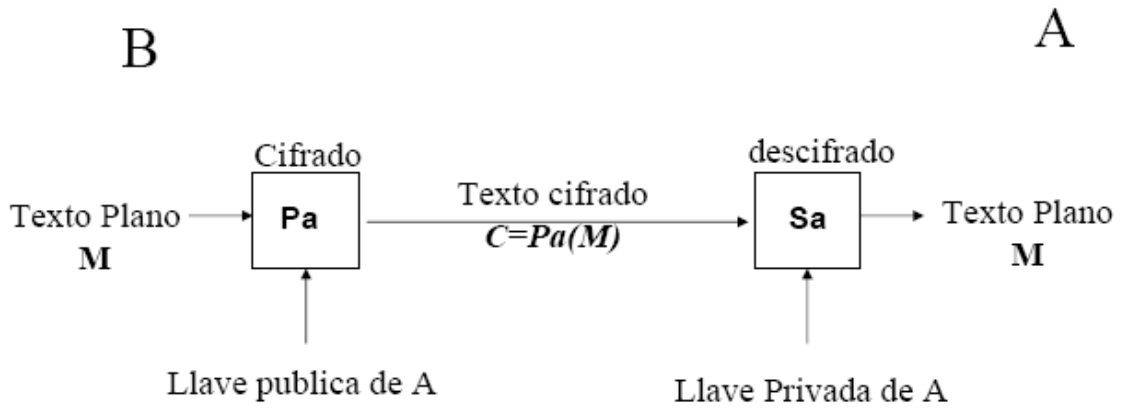

Figura 9 Cifrado de un mensaje

2.4 Conclusiones

En este capítulo se propone la arquitectura del Sistema Gestión de Credenciales basándose en las características que necesita el mismo y en las orientaciones de la Dirección de Informatización. También se proponen las herramientas y patrones que podrán ser empleadas en la implementación de este sistema.

.

CAPÍTULO 3 DESCRIPCIÓN DE LA ARQUITECTURA

3.1 Introducción

La arquitectura de un sistema está guiada, en gran medida, por los requerimientos que debe cubrir el sistema y normalmente se toma el subconjunto arquitectónicamente más importante de dichos requerimientos para definirla.

En este capítulo se especifican los requerimientos no funcionales, casos de uso, clases y diagramas que son arquitectónicamente significativos para el sistema.

3.2 Metas y restricciones arquitectónicas

A continuación se plantean las metas y restricciones con las que se deberá cumplir para la correcta puesta en funcionamiento del Sistema Gestión de Credenciales propuesto y que además son significativas para la arquitectura.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. A continuación se hace mención de los requerimientos no funcionales arquitectónicamente más significativos para el desarrollo del sistema en cuestión.

Apariencia o interfaz externa:

Tendrá una interfaz sencilla y fácil de usar, permitiendo que el tiempo de aprendizaje con el sistema sea pequeño, teniendo en cuenta que el sistema será utilizado por personas que no deberán tener grandes conocimientos de computación.

Requerimientos de Hardware:

280Mb de espacio en el disco duro o más.
Procesador 133 MHz o superior.
128 MB de memoria RAM o más.

Requerimientos de Software:

PostgreSQL.
Java.

Requerimientos de Seguridad:

Identificar al usuario antes de que pueda realizar cualquier acción sobre la configuración del sistema.
Garantizar que la información sea vista únicamente por quien tiene derecho a verla.
Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
Verificación sobre acciones irreversibles (eliminaciones).
Legales: Debe cumplir con el plan de seguridad y protección del centro.

Requerimientos de Usabilidad:

El sistema podrá ser utilizado por personas que posean conocimientos básicos de computación.

Requerimientos de Soporte:

Se requiere un servidor de bases de datos con las siguientes características:
Soporte para grandes volúmenes de datos y velocidad de procesamiento.

Requerimientos de Rendimiento:

El sistema operará con grandes volúmenes de información, ya que este almacena todos los datos de cada una de las credenciales a imprimir, por tanto, se hacen necesarios tiempos de respuesta cortos, al igual que la velocidad de procesamiento de la información, con vistas a que no existan grandes demoras para imprimir.

Requerimientos de Portabilidad:

Facilidad para adaptarlo a diferentes ambientes.

Requerimientos de Redes

Tipo de cable a utilizar UTP categoría 5, con trasmisiones de datos de hasta 100 Mbps.
Debe estar protegida contra fallos de corriente y conectividad.
La conexión entre la computadora de la especialista de acreditación y la base de datos será por TCP/IP.
La conexión entre la computadora de la especialista de acreditación y los Web Services serán a través de SOAP.

3.3 Descripción de la Arquitectura

3.3.1 Vistas arquitectónicas

En RUP, la arquitectura se compone de 4+1 vistas fundamentalmente: Vista de Casos de Uso, Vista Lógica, Vista de Procesos, Vista de Implementación y Vista de Despliegue. Estas vistas son la parte esencial de lo que se obtuvo en los flujos de Requerimientos, Análisis y Diseño e Implementación, que son las etapas que más tributan a la arquitectura. La arquitectura se representa a través de 4+1 vistas, 4 de estas vistas están regidas por una rectora: la Vista de Casos de Uso. Esto responde, a una de las características de RUP: Guiado por Casos de Usos.

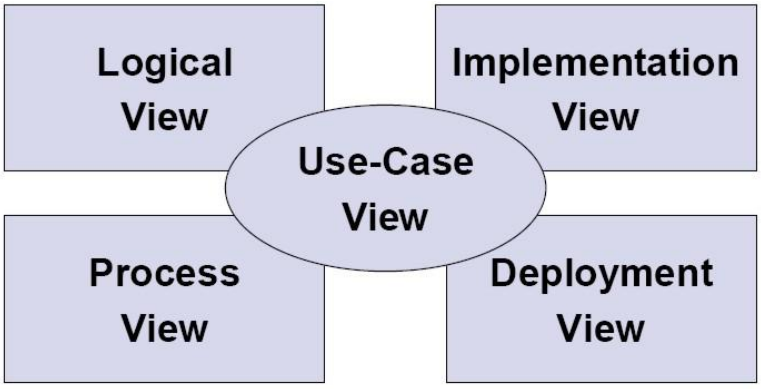

Figura 10 Vistas de la arquitectura propuesta por RUP.

3.3.1.1 Vista de Casos de Uso

La vista de casos de uso representa un subconjunto del artefacto Modelo de Casos de Uso y lista los casos de uso más significativos, con las funcionalidades centrales del sistema.

Los Casos de Uso arquitectónicamente significativos, son aquellos que describen funcionalidades imprescindibles para el sistema, y que a través de estos se valida la arquitectura propuesta para el mismo.

En esta vista se representa el diagrama de casos de uso con los casos de uso arquitectónicamente significativos para el sistema y una descripción de cada uno.

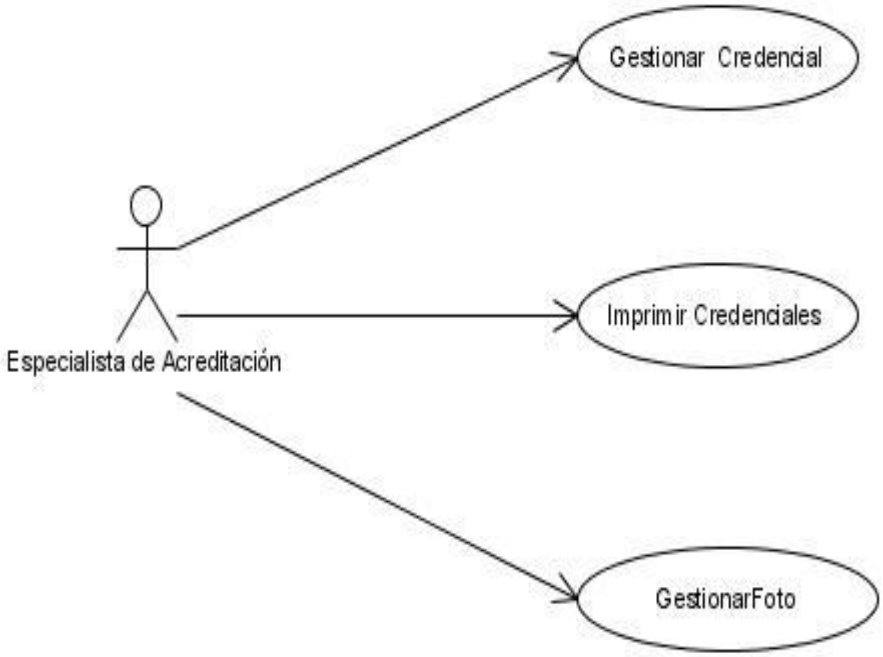

Figura 11 Casos de Uso arquitectónicamente más significativos para el sistema

Descripción de los casos de uso del sistema arquitectónicamente más significativos

Tabla 2.1 Descripción textual del Caso de Uso Gestionar Credencial

Caso de Uso:	Gestionar Credencial

Actores:	Especialista de Acreditación.
Resumen:	El caso de uso comienza cuando el Especialista de Acreditación selecciona la opción gestionar credencial. En esta sección podrá crear una nueva credencial, editarla, eliminarla, visualizarla o añadir credenciales a un grupo.
Precondiciones:	
Referencias	RF 3, RF 3.1, RF 3.2, RF 3.3, RF 3.4, RF 3.5
Prioridad	

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El Especialista de Acreditación selecciona la opción de Gestionar Credencial.	2. El sistema muestra las opciones de crear credencial, editarlas, actualizarlas, eliminarlas, visualizarlas o añadirlas a un grupo.
3.El Especialista de Acreditación selecciona una de las opciones: - Crear credencial. - Editar credencial. - Eliminar credencial. - Visualizar credencial. - Añadir credencial a un grupo.	
Sección “Crear Credencial”	
Acción del Actor	**Respuesta del Sistema**
4. El actor selecciona la opción Crear Credencial.	5. El sistema muestra una ventana con un buscador y los botones Buscar, Crear y

	Cancelar.
6. El actor introduce los patrones de la búsqueda (Solapín, Usuario, Cargo, Área, Nombre, Grupo) y oprime el botón Buscar. En caso contrario ver FA 1.	7. El sistema busca las personas que cumplen las condiciones dadas. 8. El sistema muestra un listado con los campos Nombre, CI, Solapín, Tipo de Persona.
9. El actor selecciona la o las personas a las que quiere crearle la credencial y hace clic en el botón Crear.	10. Genera un número de Solapín y un código de barras por cada credencial a crear. 11. Guarda el número de Solapín y el código de barras de cada credencial a crear y el carnet de identidad de la persona a la que pertenece la credencial. 12. Pone el estado de la credencial a Pendiente.
Sección "Editar Credencial"	
Acción del Actor	**Respuesta del Sistema**
13. El actor selecciona la opción Editar Credencial.	14. El sistema muestra una ventana que contiene un listado de todas las credenciales activas con los campos, Nombre, CI, Solapín, Tipo de Persona, y los botones Editar y Cancelar
15. El actor hace doble clic en el campo de la credencial que desea editar.	16. Si el campo es editable se permite la escritura en el mismo. En caso contrario FA 2.

17. El actor entra el nuevo valor. 18. El actor oprime el botón Editar. En caso contrario ver FA 1.	19. Se almacena el nuevo valor de dicho campo.
Sección "Eliminar Credencial"	
Acción del Actor	**Respuesta del Sistema**
20. El actor selecciona la opción eliminar credenciales.	21. El sistema muestra una interfaz con un listado de todas las credenciales activas, con los campos Nombre, CI, Solapín, Tipo de Persona y los botones Eliminar y Cancelar.
22. El actor selecciona la o las credenciales que desea eliminar. 23. El actor oprime Eliminar En caso contrario ver FA 1 25. El actor selecciona OK En caso contrario ver FA 3	24. Se muestra un mensaje de advertencia preguntándole al usuario si está seguro de borrar las credenciales seleccionadas. 26. Se les cambia el estado para Devuelta en la Base de Datos a las credenciales eliminadas. 27. Se muestra una notificación de que las credenciales se han eliminado correctamente.
Sección "Visualizar Credencial"	
Acción del Actor	**Respuesta del Sistema**
28. El actor selecciona la opción visualizar credenciales.	29. El sistema muestra una interfaz con un listado de todas las credenciales activas, con los campos Nombre, CI, Solapín, Tipo de Persona, un área para mostrar las credenciales, las opciones siguiente (>>) y anterior (<<) y el botón Cancelar.

30. El actor selecciona una o más credenciales del listado. 31. El actor selecciona la opción Visualizar. En caso contrario ver FA1.	32. El sistema busca la plantilla correspondiente a la primera credencial seleccionada y los datos de dicha credencial. 33. El sistema muestra en el área de las credenciales la primera de las credenciales seleccionadas por el actor con todos los datos llenos.
34. El actor oprime siguiente (>>) para visualizar la siguiente credencial.	35. El sistema busca la plantilla correspondiente a la siguiente credencial seleccionada y los datos de dicha credencial. 36. El sistema muestra en el área de las credenciales la siguiente de las credenciales con todos los datos llenos.
37. El actor oprime anterior (<<) para visualizar la credencial anterior.	38. El sistema busca la plantilla correspondiente a la credencial anteriormente seleccionada y los datos de dicha credencial. 39. El sistema muestra en el área de las credenciales la anterior de las credenciales con todos los datos llenos.
Sección "Añadir Credenciales a un grupo"	
Acción del Actor	**Respuesta del Sistema**
40. El actor selecciona la opción añadir credenciales.	41. El sistema muestra una ventana que contiene un listado de todas las credenciales, un listado de todos los grupos existentes y los botones Añadir y Cancelar.

42. EL actor selecciona las credenciales que desea añadir y el grupo al cual serán añadidas y oprime el botón Añadir. En caso contrario ver FA 1	43. El sistema adiciona todas las credenciales seleccionadas al grupo seleccionado. 44. El sistema muestra un mensaje de "Credenciales Adicionadas".
Flujos Alternos	
FA1	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona la opción cancelar.	2. Se termina el caso de uso.
FA2	
Acción del Actor	**Respuesta del Sistema**
	1. Muestra un mensaje de Campo no editable. Vuelve al paso 14 del flujo normal de eventos.
FA3	
Acción del Actor	**Respuesta del Sistema**
1. El actor oprime Cancelar	2. El sistema vuelve al paso 20 del flujo normal.
Prototipo de Interfaz	
Poscondiciones	

Tabla 2.2 Descripción textual del Caso de Uso Imprimir Credenciales

Caso de Uso:	Imprimir Credenciales
Actores:	Especialista de Acreditación

Resumen:	El caso de uso se inicia cuando el especialista de acreditación presiona la opción imprimir credencial. El caso de uso termina cuando el Especialista de Acreditación logra imprimir las credenciales.
Precondiciones:	
Referencias	RF 9
Prioridad	

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El Especialista de Acreditación selecciona el submenú Imprimir Credenciales.	2. El sistema muestra la interfaz que contiene una tabla con las credenciales, del grupo seleccionado y los botones. Eliminar, Cargar Hoja, Imprimir, Cerrar.
3. El actor selecciona las credenciales que desea imprimir.FA1. 4. El actor oprime el botón Cargar Hoja.	5. El sistema conforma las hojas con sus respectivas credenciales.
6. El actor oprime el botón imprimir. En caso contrario ver FA2.	7. Se imprimen las hojas con las credenciales.
Prototipo de Interfaz	
Flujos Alternos	
FA1	
Acción del Actor	**Respuesta del Sistema**
1. Si el actor desea, oprime el botón Eliminar para suprimir las credenciales que no sean necesarias.	2. El sistema elimina las credenciales seleccionadas.
FA2	
Acción del Actor	**Respuesta del Sistema**

1. El actor oprime el botón Cerrar.	2. Se termina el caso de uso.
Prototipo de Interfaz	
Poscondiciones	La(s) credencial (es) queda(n) impresa(s).

Tabla 2.3 Descripción textual del Caso de Uso Gestionar Foto

Caso de Uso:	Gestionar Foto
Actores:	El Especialista de Acreditación
Resumen:	El caso de uso comienza cuando el Especialista de Acreditación selecciona la opción Gestionar Foto. En esta sección podrá cargar foto de una localización física a la Base de Datos.
Precondiciones:	
Referencias	RF 6
Prioridad	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El Especialista de Acreditación selecciona la opción de Gestionar Fotos.	2. El sistema muestra la interfaz que contiene un campo carnet de identidad, un área para mostrar el nombre, categoría de la persona, un área para mostrar la foto de la persona, y los botones Buscar, Examinar (deshabilitado), Guardar y Cancelar.
3. El actor introduce el carné de identidad y oprime el botón Buscar.	4. El sistema verifica que la persona no tenga foto asociada. En caso contrario FA1(LA PERSONA

	POSEE UNA FOTO) 5. El sistema muestra los datos de la persona. 6. El sistema habilita el botón Examinar.
7. El actor oprime el botón Examinar y busca la foto.	8. El sistema muestra en el área imagen la foto seleccionada.
9. El actor oprime el botón Guardar. En caso contrario FA 2(CANCELAR)	10. El sistema guarda la foto seleccionada y el carné de identidad de la persona.
Prototipo de Interfaz	
Flujos Alternos	
FA1	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la foto en el campo imagen.
FA2	
Acción del Actor	**Respuesta del Sistema**
1. El actor oprime el botón Cancelar.	2. Se termina el caso de uso.
Prototipo de Interfaz	
Poscondiciones	La foto queda guardada.

3.3.1.2 Vista Lógica

Esta vista representa un subconjunto del artefacto Modelo de Diseño, representando los elementos de diseño más importantes para la arquitectura del sistema, se describen las clases más importantes, su organización en paquetes y subsistemas.

Distribución de la aplicación por capas:

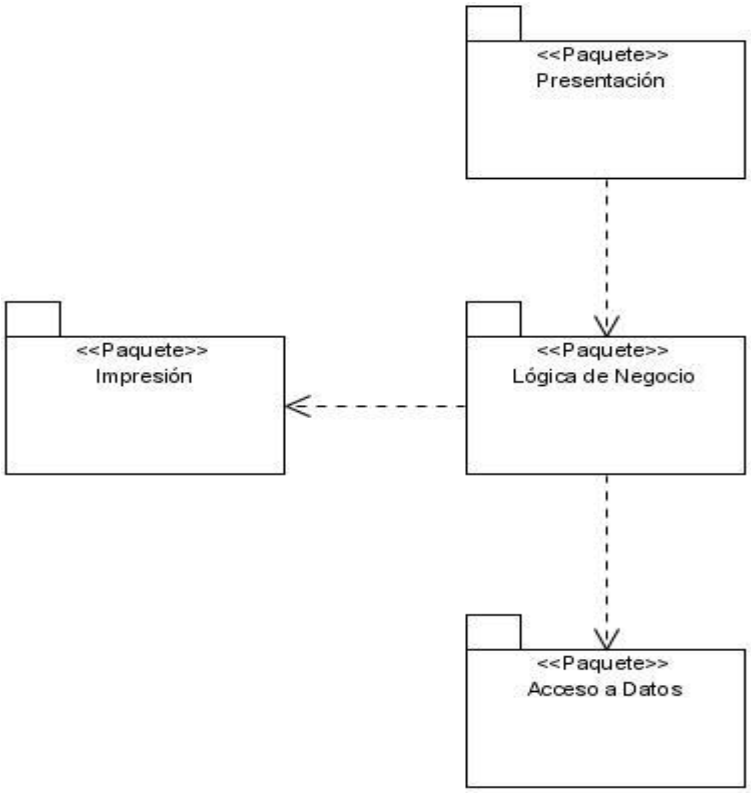

Figura 12 Estructura en capas

Nombre	Estereotipo	Descripción
Presentación	Paquete	Conjunto de clases que componen las interfaz del sistema.
Lógica de Negocio	Paquete	Conjunto de clases encargadas de gestionar todos los procesos de la lógica de negocio.
Impresión	Paquete	Conjunto de clases encargadas de gestionar los detalles de las credenciales a imprimir.
Acceso a Datos	Paquete	Conjunto de clases que controlan el tratamiento de los datos.

Tabla 2.4 Descripción de los paquetes

En el paquete de **Presentación** se tienen las clases interfaces:

Frm_FormaPrincipal

Frm_CrearCredencial

Frm_EditarCredencial

Frm_EliminarCredencial

Frm_VisualizarCredencial

Frm_AñadirCredencialesGrupo

Frm_ImprimirCredenciales

Frm_GestionarFoto

En el paquete **Lógica de Negocio** se tienen las clases controladoras:

GestionarCredencialNEG.

GestionarFotoNEG

ImprimirCredencialesNEG

En el paquete **Impresión** se tiene la clase:

GestionarDetallesImpresiónNEG

La estructura del paquete Acceso a Datos es la siguiente:

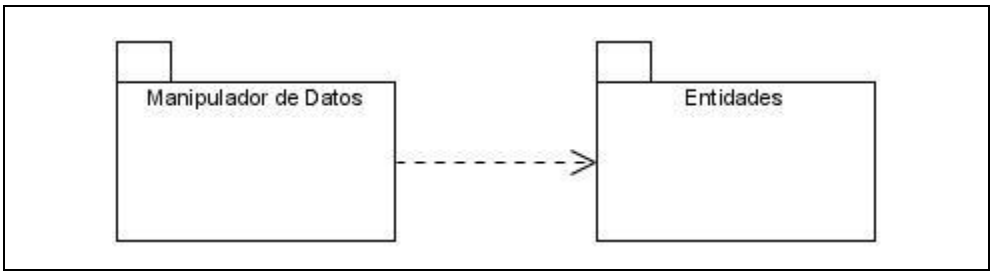

Figura 13 Capa de acceso a datos

Manipulador de Datos

En este paquete están definidas las clases que manipulan el acceso a los datos:

GestionarCredencialesDAO

GestionarFotoDAO
ImprimirCredencialesDAO

Entidades

Conjunto de clases persistentes del proyecto:

CE Estudiante
CE Trabajador
CE Credencial

3.3.1.3 Vista de Implementación

En esta vista se muestra como se implementan los componentes físicos del sistema agrupándolos en capas y subsistemas de implementación, cada capa agrupa a su vez un conjunto de componentes que son los que van a dar solución al problema.

Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

Descripción Global

En esta descripción global se muestra de forma general cómo están distribuidos los paquetes de componentes por capas.

El sistema se divide en 4 capas, cada capa va a contener un paquete de componentes. Cada paquete agrupa a su vez un conjunto de componentes que son los que van a dar solución al problema.

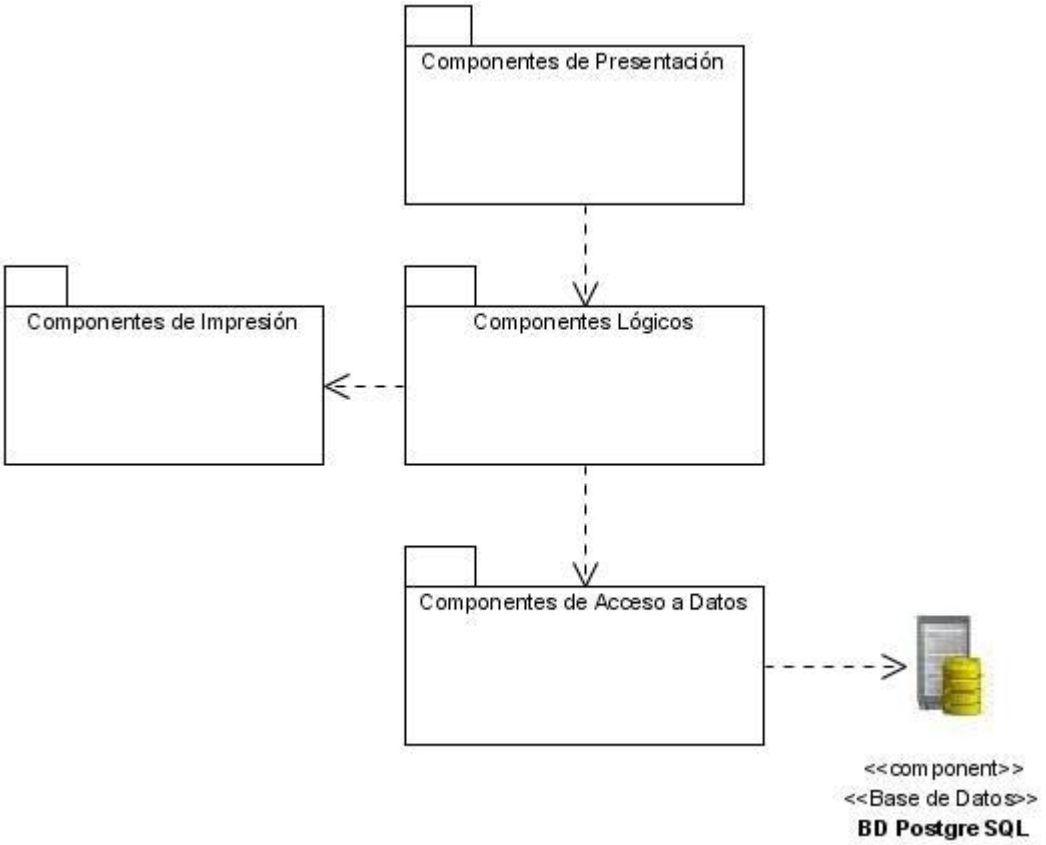

Figura 14 Paquetes de componentes por capas

Capas

En esta apartado se muestra cómo se implementan los componentes físicos del sistema agrupándolos en capas.

Capa de Presentación:

Los componentes que se implementarán para interactuar con el usuario del sistema están agrupados en la capa de presentación.

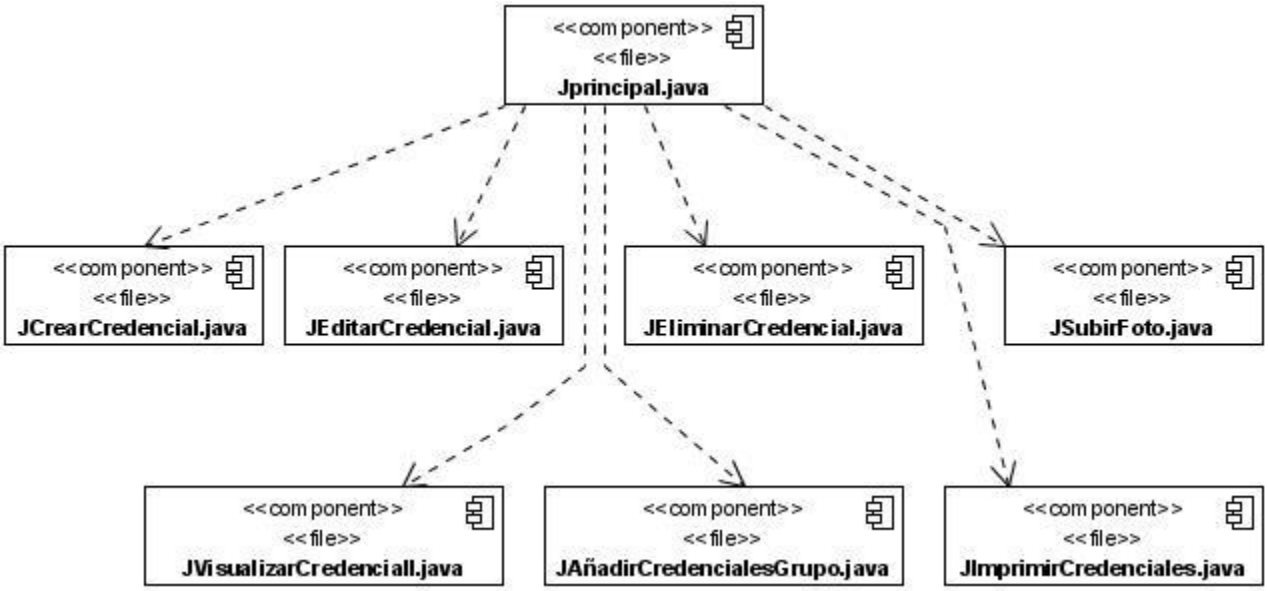

Figura 15 Componentes distribuidos en la capa de presentación.

Componentes que están agrupados en el paquete **Componentes de Presentación**:

Estos componentes son los que permiten interactuar al usuario con el sistema a través de interfaces que serán sencillas y amigables para que cualquier persona con conocimientos básicos pueda trabajar con la aplicación.

Capa Lógica.

Los componentes que intervienen en la lógica de la solución del problema están agrupados en esta capa lógica.

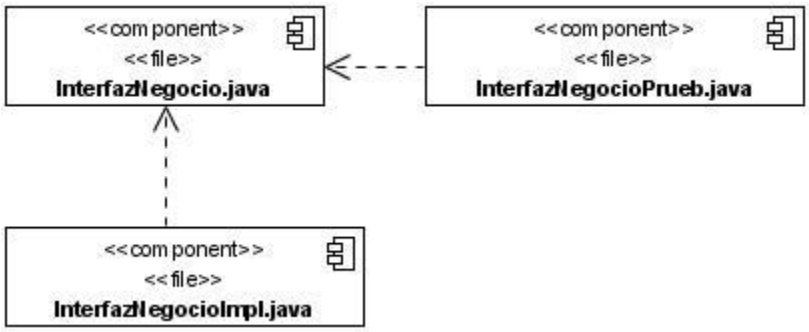

Figura 16 Componentes distribuidos en la capa lógica.

Componentes que están agrupados en el paquete **Componentes Lógicos**:

Componente **InterfazNegocio.java:** Es la interfaz que expone a la capa de presentación los métodos para las operaciones de negocio.

Componente **InterfazNegocioImpl.java:** Es el componente que implementa la interfaz de negocio, desarrollando los métodos que contendrán la lógica de negocio.

Componente **InterfazNegocioPrueb.java:** Es el componente que prueba todos los métodos del componente **InterfazNegocioImpl**, con la utilización de valores de pruebas en casos extremos, es la responsable de asegurar que la lógica del negocio implementada en el componente **InterfazNegocioImpl** responda a los requerimientos que debe cumplir.

Capa de Impresión:

Esta capa contendrá las clases que proveen al usuario de la aplicación de las funcionalidades para ajustar los detalles de las credenciales a imprimir.

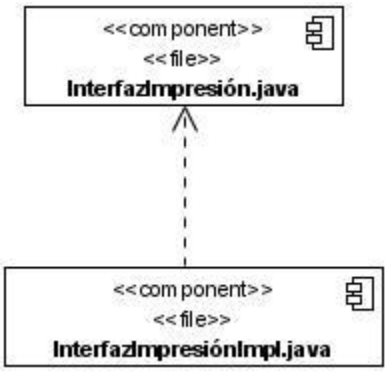

Figura 17 Componentes distribuidos en la capa de impresión.

Componentes que están agrupados en el paquete **Componentes de Impresión**:

Componente **InterfazImpresión.java:** A través de este componente se comunica la capa impresión con el negocio.

Componente **InterfazImpresiónImpl.java:** Es el componente que implementa la interfaz de impresión, desarrollando los métodos que permitirán ajustar los detalles de las credenciales a imprimir.

Capa de Acceso a Datos

Los componentes que interactúan con los dispositivos de almacenamientos, es decir los que se encargan del acceso de los datos están agrupados en la capa de acceso a datos.

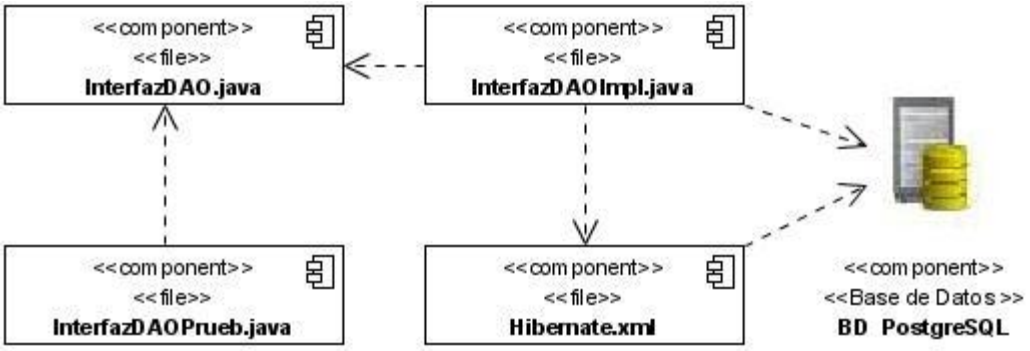

Figura 18 Componentes distribuidos en la capa de acceso a datos.

Componente **InterfazDAO.java:** Es la interfaz que expone a la capa de servicios de negocio los métodos del DAO.

Componente **InterfazDAOImpl.java:** Este componente es el que implementa la interfaz, es el que permite dar funcionalidad real a los métodos de la InterfazDAO.

Componente **InterfazDAOPrueb.java:** Es el componente que prueba todos los métodos de la clase InterfazDAOImpl, con la utilización de valores de pruebas en casos extremos, es la responsable de asegurar que los métodos del objeto de acceso a datos están funcionando correctamente, para utilizar el DAO desde la capa de servicios de negocio.

Componente **Hibernate.xml:** Facilita el mapeo de los atributos de la base de datos y el modelo de objetos de la aplicación y permite manipular los datos operando sobre objetos, con todas las características de la POO.

Relación entre las capas

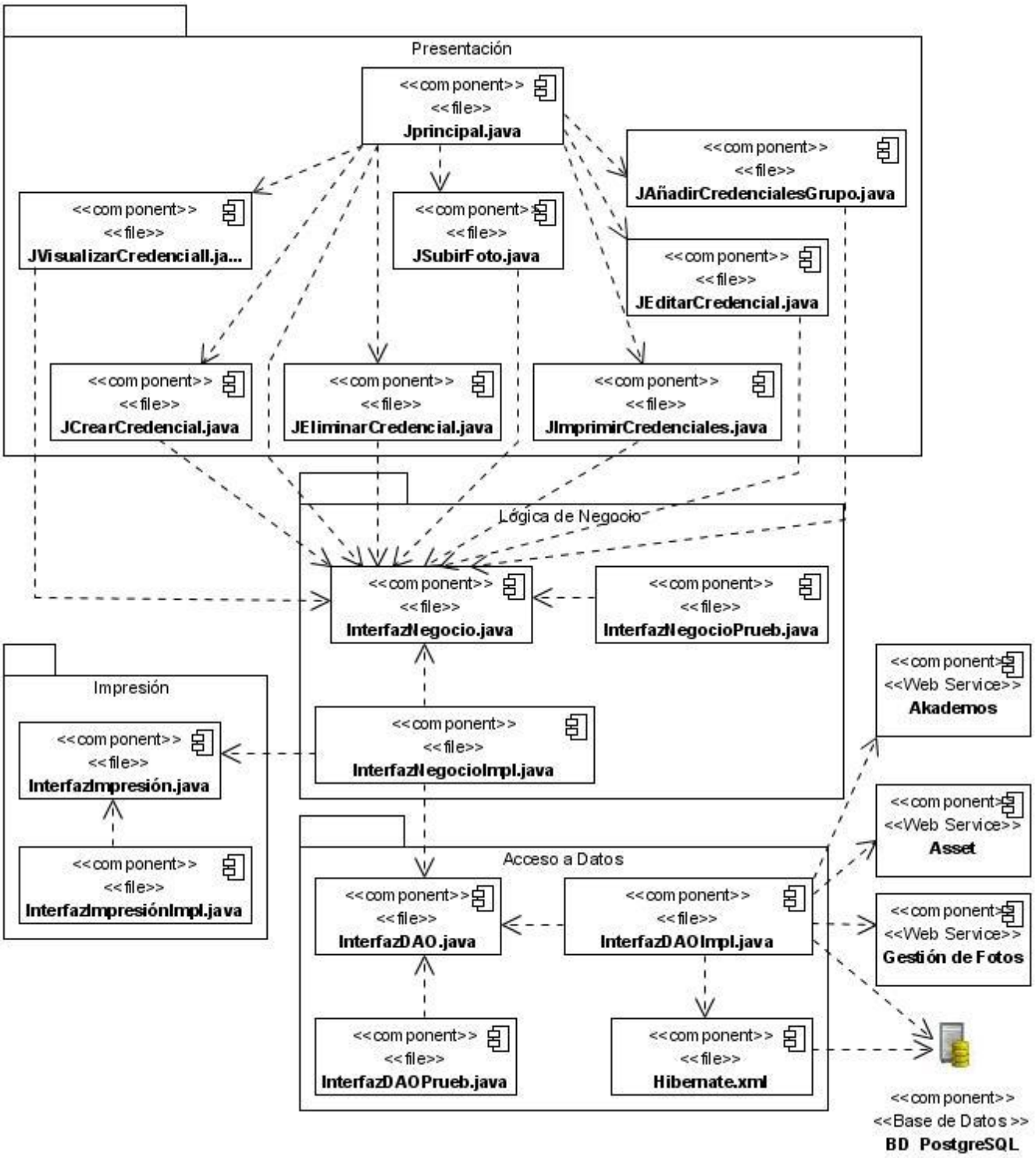

Figura 19 Relación de los componentes entre las capas.

Los componentes de las capas en las que se ha dividido el sistema para la mejor organización y nivelación del trabajo estarán muy relacionados. Los componentes de una capa solamente podrán usar los componentes de la capa inmediata inferior, pero no ocurrirá lo contrario. Los componentes de una capa solamente brindarán servicios a los componentes de la capa inmediata superior.

3.3.1.4 Vista de Despliegue

Mediante esta vista se obtiene una base para la comprensión de la distribución física de un sistema a través de nodos, un nodo es un recurso en ejecución como puede ser un computador, un dispositivo, un procesador o memoria, estos se identifican utilizando estereotipos.

El sistema de gestión de credenciales es una aplicación que está situada en la computadora de la especialista de acreditación, que se conectará a una base de datos a través de TCP/IP para guardar y obtener informaciones importantes que el sistema maneja, además se utilizarán los Servicios Web Akademos, Asset y Gestión de Fotos, los cuales estarán conectados con la computadora de la especialista de acreditación mediante el protocolo SOAP con el objetivo de gestionar informaciones que hacen falta en el proceso de gestión e impresión de credenciales y en la computadora donde esta la aplicación estará conectada una impresora mediante un puerto serie para realizar la impresión de las credenciales PVC.

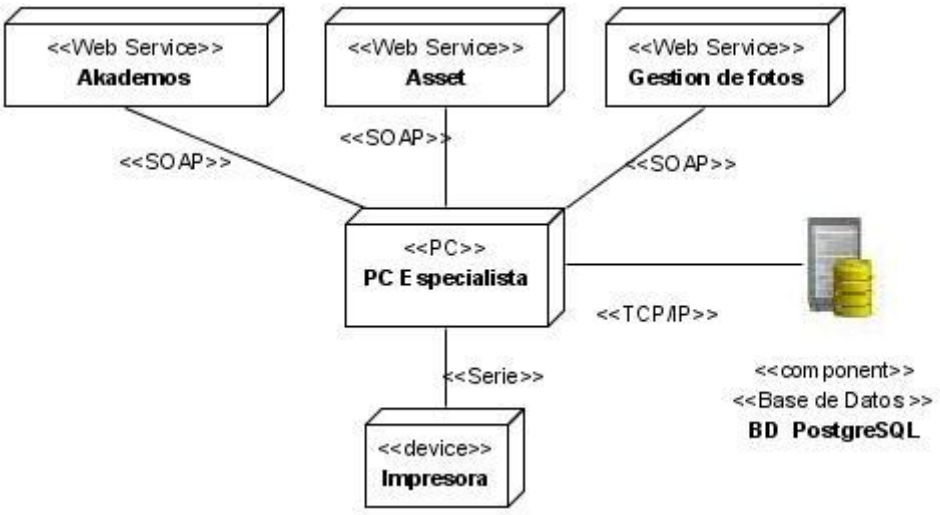

Figura 20 Diagrama de Despliegue.

Nodo **PC Especialista:** En el se encuentra la aplicación de gestión de credenciales y a través de él se gestiona toda la información relevante para la acreditación del personal de la Universidad.

Nodo **Web Service Akademos:** Brinda las funcionalidades de gestionar los datos de los estudiantes.

Nodo **Web Service Asset:** Brinda las funcionalidades de gestionar los datos de los trabajadores de la UCI.

Nodo **Web Service Gestión de fotos:** Brinda las funcionalidades de gestionar fotos del personal de la UCI.

Nodo **BD PostgreSQL:** En él se encuentra la base de datos que permite almacenar información importante que el Sistema Gestión de Credenciales debe ser capaz de guardar.

Dispositivo **Impresora**: Permite imprimir las credenciales gestionadas por el sistema.

3.3.1.5 Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

En este caso la solución propuesta no presenta procesos concurrentes por tanto no se hace representación de esta vista.

3.4 Conclusiones

En este capítulo se desarrollaron los modelos, que según RUP, son más importantes para la descripción de la arquitectura. Finalmente quedó establecida la arquitectura del Sistema de Gestión de Credenciales.

CONCLUSIONES

El desarrollo de la arquitectura de software es una de las etapas fundamentales en la construcción del mismo, ya que es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para construir la mejor propuesta de solución que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema. Es de trascendental importancia que todo sistema de software esté amparado por una arquitectura sólida que provea el entendimiento del mismo, que sea capaz de organizar el proceso, fomentar la reutilización y hacer progresar el sistema.

En el presente trabajo se realizó un estudio acerca de varios estilos arquitectónicos que existen en la actualidad, para emplear las mejores técnicas de diseño arquitectónico. Se propuso una arquitectura de cuatro capas que facilitará el desarrollo paralelo del sistema, su mantenimiento y el soporte a la aplicación.

Esta aplicación puede llegar a convertirse en una importante herramienta para llevar el control de las credenciales que existen en nuestra Universidad, pudiendo gestionar este proceso con eficacia y rapidez como se hace necesario en la actualidad debido al gran número de personas que residen en ella y a las que entran cada curso.

El sistema a desarrollar estará provisto de una interfaz amigable, sencilla e intuitiva, utilizando modernas técnicas de programación orientada a objetos.

Con la realización de este trabajo queda una documentación sobre el proceso de gestión de credenciales, lo cual permitirá su estudio para futuros cambios o cuando sea necesario darle mantenimiento y soporte a la aplicación.

RECOMENDACIONES

En el presente trabajo se propuso una arquitectura para el Sistema de Gestión de Credenciales de la UCI, pero pudieran surgir nuevas ideas que puedan servir para mejorar mucho más el mismo, por lo que se recomienda:

1. Continuar desarrollando el sistema, adecuándolo a las nuevas necesidades que aparezcan en el transcurso del tiempo debido a la importancia que este tiene, ya que de este sistema dependen muchas actividades en la UCI.

2. Utilizar el sistema en otros centros o instituciones convirtiéndolo así en una fuente de ingresos.

3. Realizar una evaluación de los costes de la tecnología utilizada, ya que aunque la misma en su totalidad es libre, es necesario evaluar los costes del hardware que la soporta.

4. Continuar con el refinamiento de la arquitectura propuesta.

5. Aplicar los métodos de evaluación de la arquitectura para identificar las posibles debilidades.

6. Que sea aplicada esta arquitectura en el proyecto Identificación y Control de Acceso.

BIBLIOGRAFÍA CONSULTADA

1. **Sun.** Java en castellano. [En línea] 1999-2007. http://www.programacion.com/java/tutorial/patrones2/8#patrones28_implem_data.

2. **Sarwat, Rames.** La nueva oleada de algoritmos criptográficos. [En línea] http://www.revista-ays.com/DocsNum10/PersEmpresarial/sarwat.pdf.

3. **Pecos, Daniel.** PostGreSQL vs. MySQL. [En línea] http://www.netpecos.org/docs/mysql_postgres/index.html.

4. **Navarro, Juanjo.** más que código. [En línea] 9 de diciembre de 2003. http://www.juanjonavarro.com/masquecodigo/2003/12/09/netbeans-frente-a-eclipse.

5. **Mason, Mike.** Osmosis Latina. [En línea] 20 de octubre de 2005. http://www.osmosislatina.com/subversion/basico.htm.

6. **M., Jose F. Torres.** Sistema criptográfico de llave publica RSA. *Análisis y diseño de algoritmos.* [En línea] http://www.ing.ula.ve/~ibc/ayda/c26rsa.pdf.

7. **Gracia, Joaquin.** IngenieroSoftware. [En línea] 7 de Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/uml.php.

8. **Española, Oficina.** WORD WIDE WEB consortium. [En línea] p de febrero de 2008. http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb.

9. **Andersen, Max R.** HIBERNATE. [En línea] 2008. http://www.hibernate.org/255.html.

10. wikipedia. [En línea] 29 de mayo de 2008. http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado.

11. WEBSERVICES.ORG. [En línea] 2001 - 2008. http://webservices.org/.

12. Visual Paradigm. [En línea] 2008. http://www.visual-paradigm.com/.

13. Tutorial de PostgreSQL. [En línea] http://es.tldp.org/Postgresql-es/web/navegable/tutorial/tutorial.html.

14. TextosCientíficos. [En línea] http://www.textoscientificos.com/redes/redes-virtuales/tuneles/encriptacion.

15. Slideshare. [En línea] 2008. http://www.slideshare.net/vivi_jocadi/rational-rose/.

16. SESDI. [En línea] 2007. http://www.sesdi.com/cb/cgcardfive.htm.

17. PostgreSQL. [En línea] 2008. http://www.postgresql.org/.

18. Patrones de diseño. [En línea] http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-12.pdf.
19. OsmosisLatina. [En línea] 20 de Octubre de 2005. http://www.osmosislatina.com/subversion/basico.htm.
20. NetBeans. [En línea] http://www.netbeans.org/index_es.html.
21. MBCEStores. [En línea] mayo de 2008. http://www.mbcestore.com.mx/evolis/emedia.htm.
22. Lenguaje de Programación. [En línea] 2003. http://www.lenguajes-de-programacion.com/.
23. Herramientas WEB para la enseñanza de PROTOCOLOS DE COMUNICACIÓN. [En línea] http://neo.lcc.uma.es/evirtual/cdd/tutorial/presentacion/rsa.html.
24. GEk (n.). [En línea] 27 de enero de 2007. http://mundogeek.net/archivos/2007/01/27/hibernate/.
25. FREE DOWNLOAD MANAGE. [En línea] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/.
26. epistemoWIKIA. [En línea] 2008. http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=Eclipse_SDK.
27. desarrolloweb. [En línea] http://www.desarrolloWeb.com/articulos/1557.php.
28. CVS - Concurrent Versions System. [En línea] 3 de diciembre de 2006. http://www.nongnu.org/cvs/
29. Visual Paradigm. [En línea] 12 de mayo de 2007. http://www.visual-paradigm.com/news/vpsuite30sp2/vpuml60sp2.jsp.
30. Viva Linux. [En línea] 6 de Diciembre de 2006. http://www.vivalinux.com.ar/soft/postgresql-8.2.html.
31. CodigoPHP.com. [En línea] 15 de octubre de 2006. http://www.codigophp.com/2006/10/15/postgresql-82/ .
32. Buayacorp. [En línea] 5 de diciembre de 2006. http://www.buayacorp.com/archivos/postgresql-82/.
33. Tigris.org Open Sourse Software Engineering Tools. [En línea] http://subversion.tigris.org/.
34. Todo Programas. [En línea] enero de 2008. http://www.todoprogramas.com/programa/eclipsesdk.

35. Mancomun . [En línea] 2008. http://www.mancomun.org/novas/observacion-tecnoloxica/lanzado-o-novo-eclipse-3.2-3.html.

36. Tigris.org. [En línea] http://subclipse.tigris.org/.

37. Herrera, Cristhian. AdictosalTrabajo. [En línea] 16 de agosto de 2007. http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateVSEJB3.

.

BIBLIOGRAFÍA CITADA

1. **Kruchten, Philippe.** Wikipedia. [En línea] mayo de 2008. http://es.wikipedia.org/wiki/Arquitectura_software.

2. **Sanchez, María A. Mendoza.** Informatizate. [En línea] 7 de junio de 2004. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

3. Estilos y Patrones. [En línea] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/EstilosyPatrones.htm.

4. Patrones de diseño. [En línea] 23 de octubre de 2003. http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-12.pdf .

5. **Lago, Ramiro.** Patrones de diseño software. [En línea] Abril de 2007. http://www.proactiva-calidad.com/java/patrones/index.html.

GLOSARIO DE TÉRMINOS

Lenguaje de Modelado Unificado (UML): Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

Metodología: En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

Rational Unified Process (RUP): Es un proceso de desarrollo de software.

Proceso de desarrollo de software: Es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

API's: Conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Credencial: Documento que sirve para que a un empleado se le reconozca como perteneciente a su centro de trabajo y se reconozca su plaza.

Acreditación: Proceso de creación de credenciales a una entidad dada.

PVC: (PolyVinyl Chloride, Cloruro de polivinilo): Material plástico altamente flexible.

TCP/IP: (Transfer Control Protocol/ Internet Protocol, Protocolo de Control de Transmisión/ Protocolo de Internet) Protocolo de transmisión de datos en Internet, ampliamente utilizado, permite conectar computadoras con diferentes sistemas operativos.

Servicios Web: Un servicio Web es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, a través de la Web. Los Servicios Web pueden ser utilizados por distintas aplicaciones de software, desarrollados en diferentes lenguajes de programación y ejecutados sobre cualquier plataforma, para intercambiar datos en redes de ordenadores como Internet.

HTTP (Hyper Text Transfer Protocol): En español, protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la Web. El hipertexto es el contenido de las páginas Web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido. También sirve el protocolo para enviar información adicional en ambos sentidos, como formularios con campos de texto.

Repositorio: Un repositorio, depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente base de datos o archivos informáticos.

Protocolo: Conjunto de normas y procedimientos útiles para la transmisión de datos, conocido por el emisor y el receptor.

GUI (en inglés **G**raphical **U**ser **I**nterface): Interfaz Gráfica de Usuario es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

XML (sigla en inglés de e**X**tensible **M**arkup **L**anguage): Lenguaje de marcado extensible es un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Bajo acoplamiento: Dependencia mínima entre componentes del sistema.

Interfaz: Un límite a través del cual dos entidades independientes se encuentran y se relacionan o comunican la una con la otra.

Tarjetas CRC: Son una metodología para el diseño de software orientado a objetos creada por Kent Beck y Ward Cunningham. Como una extensión informal a UML, la técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis guiados por la responsabilidad. Las clases se examinan, se filtran y se refinan basándose en sus

responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades.

SOAP (Simple Object Access Protocol): Protocolo de Acceso Simple a Objetos, es un protocolo estándar define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

VCL (Visual Component Library): Biblioteca de componentes creada por Borland para su producto Delphi y que, posteriormente, también ha usado en C++ Builder. Con esos componentes se simplifica el desarrollo visual de aplicaciones para Windows.

ODBC (Open DataBase Connectivity): Conectividad Abierta de Bases de Datos.

ADO (ActiveX Data Objects): Conjunto de objetos para el acceso a datos desarrollado por Microsoft, que acompañan a su servidor IIS. Mediante ADO se pueden escribir secuencias de comandos que conecten con bases de datos compatibles con ODBC.

dbExpress: Sistema de acceso a datos, los controladores DBExpress son veloces, simples de configurar e instalar.

Ajax (Asynchronous JavaScript and XML): Utilización conjunta de varias tecnologías (XHTML, CSS, DOM, XML, XSLT y JavaScript) para conseguir que las aplicaciones puedan intercambiar datos con el servidor de manera asíncrona, sin que la interacción con el usuario se vea nunca interrumpida. El mecanismo fundamental de Ajax es el Intercambio de datos asíncrono mediante XMLHttpRequest. Con este objeto, JavaScript puede realizar peticiones al servidor remoto y recibir respuestas HTTP sin la recarga de la página, por lo que el usuario no aprecia interrupción alguna.

JSP (Páginas de Servidor Java): Es una tecnología orientada a crear páginas web con programación en Java.

Release: Se refiere a un producto final, preparado para lanzarse como versión definitiva a menos que aparezcan errores que lo impidan. En esta fase el producto implementa todas las

funciones del diseño y se encuentra libre de cualquier error que suponga un punto muerto en el desarrollo.

Sobrecarga de operadores: La sobrecarga de operadores permite redefinir ciertos operadores, como "+" y "-", para usarlos con las clases que hemos definido. Se llama sobrecarga de operadores porque se reutiliza el mismo operador con un número de usos diferentes, y el compilador decide cómo usar ese operador dependiendo sobre qué opera. La sobrecarga de operadores sólo se puede utilizar con clases, no se pueden redefinir los operadores para los tipos simples predefinidos.

Plugin: Pieza de software, generalmente pequeña, que añade funcionalidades a un software mayor. El usuario, partiendo de una versión reducida del software, añade aquellas capacidades nuevas que necesita, eliminando así la necesidad de tener que instalar un software con todas las posibilidades.

UTP (Unshielded Twisted Pair): Par Trenzado no Apantallado.