ESCUELA POLITÉCNICA DEL EJÉRCITO

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ANÁLISIS, DISEÑO Y CONSTRUCCIÓN DEL SISTEMA DE CONTROL DE CALIFICACIONES Y SERVICIOS VIRTUALES PARA DOCENTES Y ESTUDIANTES DE ESTUDIOS PRESENCIALES DE LA ESPE A TRAVÉS DEL WEB

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS E INFORMÁTICA

POR:

DAVID MEZA G.

SANGOLQUÍ, Febrero del 2005

Índice de Contenidos

Listado de Figuras

RESUMEN

La Escuela Politécnica del Ejercito (ESPE) es una institución educativa con mucho reconocimiento y prestigio por su amplia trayectoria y nivel de educación que ofrece a sus alumnos. Tiene como una de sus metas estratégicas, ubicarse entre las mejores Universidades de Latinoamérica en el año 2007 mediante un proceso de mejora continua basada en talento humano y en la optimización de sus procesos a través del uso de herramientas tecnológicas de última generación.

Este proyecto de tesis, ha tenido como objetivo principal colaborar con la meta de la ESPE a través del análisis, diseño y construcción del sistema de control de calificaciones para docentes y estudiantes de estudios presenciales de la ESPE. La modalidad presencial la ESPE cuenta con 4 campus dispersos por el país:

- Sangolquí (Varias facultades)
- Latacunga (Varias facultades)
- Santo Domingo (IASA2)
- Quito (Idiomas)

El proyecto permite acceso a todos los docentes y alumnos de cualquiera de los cuatro campus, de cualquier facultad de modalidad presencial con una interfaz integrada y permitiendo el control de acceso a la aplicación (Autorización y Autenticación) de manera granular.

Los sistemas a ser implementados en esta tesis se realizarán utilizando la misma arquitectura y estándares definidos por la Dirección de Organización y

Sistemas, que es el departamento de Sistemas de la ESPE, para aplicaciones WEB, permitiendo de esta manera estandarizar las aplicaciones y haciendo más fácil su mantenimiento y control, además se requirió la reutilización y mejora de clases y componentes ya existentes.

En este contexto, la implantación de este sistema de comunicación abierta, accesible y de fácil manejo, puede constituirse en una herramienta poderosa que permita a los estudiantes, docentes y a la Universidad en sí, mejorar la calidad y la eficiencia en el desempeño de sus actividades.

I. GENERALIDADES

1.1 TEMA

Análisis, diseño y construcción del sistema de control de calificaciones y servicios virtuales para docentes y estudiantes de estudios presenciales de la ESPE a través del web

1.2 OBJETIVOS

1.2.1 Objetivo General

Realizar el análisis, diseño y construcción del sistema de control de calificaciones y servicios virtuales para docentes y estudiantes de estudios presenciales de la ESPE a través del web.

1.2.2 Objetivos Específicos

- Optimizar el proceso de registro y publicación de horarios, calificaciones, cartillas, récord académico y materias que toma el alumno en el período vigente.
- Permitir el registro en línea de información académica por parte de los docentes.
- Dinamizar el proceso de enseñanza aprendizaje mediante la creación de nuevos servicios de intercambio de información y material de investigación entre estudiantes y docentes.
- Integración del sistema de modalidad a distancia con el de modalidad presencial.
- Configuración e instalación de la aplicación para estudios presenciales.

1.3 JUSTIFICACIÓN E IMPORTANCIA DEL PROBLEMA

Al momento, la ESPE se encuentra organizada en 2 grandes modalidades de estudio: Presencial y Distancia, cada una de las cuales con una serie de características propias tanto en datos como en procedimientos, aparentemente independientes entre si, pero totalmente interrelacionadas por la existencia de alumnos que cursan materias en ambas modalidades.

A nivel informático los datos se encuentran organizados de forma similar a las modalidades, existiendo una base de datos por la modalidad presencial y otra base de datos por la Modalidad de Estudios a Distancia (MED). A nivel de la Red Lan, cada modalidad dispone de su aplicación informática. No así en el Internet, donde actualmente los servicios académicos que la ESPE ofrece a través del sitio web, han sido diseñados y tienen una mayor orientación al control académico de la MED, sin existir la integración con los procesos que se realizan en la modalidad presencial.

Esta situación ocasiona que exista: duplicación de funciones, desperdicio de recursos, prolongación de trámites, entre otros. Por estas razones surge este proyecto que pretende dar una solución a la problemática actual a través de la optimización de los servicios administrativos y académicos que la ESPE brinda a través de su portal.

1.4 ALCANCE

Este proyecto comprende el Análisis, Diseño, Desarrollo e Implementación, del control de calificaciones y servicios virtuales para docentes y estudiantes de la modalidad presencial con sus respectivas sedes.

Los siguientes servicios funcionarán en el portal institucional y permitirán:

- Servicios Académicos Estudiantes:
  - ✓ Actualización de datos personales
  - ✓ Consulta de las materias que el estudiante puede tomar para la matrícula (prematricula)
  - ✓ Consulta de materias registradas por el estudiante como prematrícula
  - ✓ Consulta de Notas del periodo vigente.
  - ✓ Consulta del récord académico de la diferentes sedes y facultades en la que el estudiante haya realizado sus estudios
  - ✓ Consulta de horarios de clase.
  - ✓ Servicios virtuales que le permitan bajarse archivos o trabajos enviados por los docentes.
  - ✓ Recepción y Envío de comentarios o sugerencias a estudiantes de la misma materia, docentes o coordinadores de área.

- Servicios Académicos Docentes:
  - ✓ Ingreso de calificaciones por materia y paralelo
  - ✓ Ingreso de calificaciones por alumno
  - ✓ Impresión de cartillas por materia y paralelo
  - ✓ Impresión de cartillas agrupados por materia
  - ✓ Ingreso de horarios de clases
  - ✓ Consulta de horarios de clases de docentes
  - ✓ Servicios virtuales: envío de archivos o trabajos a través del Internet a los estudiantes.
  - ✓ Recepción y envío de comentarios y sugerencias de los estudiantes o del resto de la comunidad.
  - ✓ Intercambio de material de investigación y consulta entre docentes.

- Mejoramiento del sitio de control académico en el portal

  - ✓ Integración en plataforma y programación entre el sistema de Internet para modalidad a distancia y el sistema de Internet para modalidad presencial.

- Configuración e instalación de los nuevos servicios de control académico de la ESPE, en el portal institucional.

II. MARCO TEÓRICO

2.1 INTRODUCCIÓN

La revolución tecnológica ha ingresado en todos los aspectos de la sociedad y como no podía ser de otra manera en el campo de la educación está presente y es una fuente inagotable de conocimiento disponible para todos. En este ámbito, la tecnología Internet juega un papel fundamental.

La creación de sistemas basados en el Internet que satisfagan las necesidades de la ESPE y que brinden facilidades de aprendizaje y enseñanza tanto a los estudiantes y a los profesores permitirá simplificar las labores cotidianas.

La ESPE, como todas las instituciones de gran tamaño y con gran cantidad de usuarios, requiere que los sistemas que se realizan para sus usuarios se basen en estándares y sean compatibles e integrados, para de esta manera reducir los tiempos de control y costos operativos.

Al desarrollar este proyecto se tratará de brindar nuevos servicios con todas las facilidades posibles, que la tecnología permite y que coadyuven a que nuestra universidad alcance día a día la excelencia, como uno de sus objetivos estratégicos.

2.2 Software Base

El Software Base de un sistema son los diferentes programas que se necesitan para que éste funcione. Para el caso específico de la aplicación a implementarse en este trabajo de tesis, el software base que es el estándar de Organización y Sistemas para aplicaciones WEB, y que se encuentra instalado y en funcionamiento, es el siguiente:

- Servidor de Aplicaciones Java - Jakarta Tomcat
- Motor de Base de Datos – Sybase

2.2.1 Servidor de Aplicaciones Java – Jakarta Tomcat

Jakarta Tomcat es uno de los servidores de aplicaciones java basados en los estándares y la plataforma J2EE definida por Sun Microsystems. Tomcat es desarrollado como parte del proyecto de código abierto Jakarta de la fundación de software Apache (www.apache.org) y es uno de los servidores de aplicaciones java más utilizados en el mundo, en especial porque es muy liviano, cumple con todos los estándares, es muy sencillo de instalar, tiene muy buena documentación y es gratuito.

Tomcat como cualquier servidor de aplicaciones ofrece básicamente los siguientes servicios:

- *Pooling*[1] y compartición de recursos como procesos, conexiones a la base de datos, o sesiones de red.
- Gestión de sesiones.
- Aislamiento de la lógica de negocio.
- Integridad transaccional.
- Seguridad de datos.
- Balanceo de carga y *fail over*[2] automático.

La aplicación desarrollada podrá ser ejecutada en cualquier servidor de aplicaciones J2EE que cumpla con la especificación de JSP 2.0 o superior, ya que los servidores de aplicaciones java tienen que seguir el estándar y proveer la funcionalidad especificada en este.

2.2.1.1 Plataforma para el desarrollo de Aplicaciones Empresariales Distribuidas J2EE.

J2EE es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales. J2EE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones.

Una aplicación distribuida, además de dar respuesta a las necesidades concretas para la que ha sido diseñada, debe manejar distribución de objetos, así

[1] Permite a una aplicación mantener listos para su utilización o reutlización, un número definido de recursos que son costosos de obtener o instanciar.
[2] Es la prevención contra fallos con sistemas redundantes o modos de recuperación.

como el almacenamiento y recuperación de objetos o persistencia (típicamente utilizando una Base de Datos), soporte para concurrencia y seguridad, soporte para transacciones y soporte para poder encontrar objetos o recursos distribuidos

Entre los principales servicios que presta la plataforma J2EE están los siguientes:

- Soporte de Distribución
- Persistencia
- Seguridad
- Transacciones
- Soporte para concurrencia
- Naming Service[3] y servicio de directorios

Las tecnologías que utiliza la especificación J2EE son las siguientes:

- **Tecnologías de Componentes.-** Son utilizadas para tomar la parte más importante de las aplicaciones – Las reglas del negocio.
- **Tecnologías de Servicios.-** Proveen componentes de la aplicación con soporte de servicios para funcionar eficientemente.
- **Tecnologías de Comunicación.-** Proveen mecanismos para la comunicación entre diferentes partes de la aplicación, siendo estos locales o remotos.

[3] Es un sistema de localización de recursos por medio de un nombre asignado en un repositorio.

La arquitectura de la plataforma J2EE se presenta en la figura 1.1, en ella se puede distinguir las diferentes API's para el desarrollo de diferentes aplicaciones que conforman la especificación.

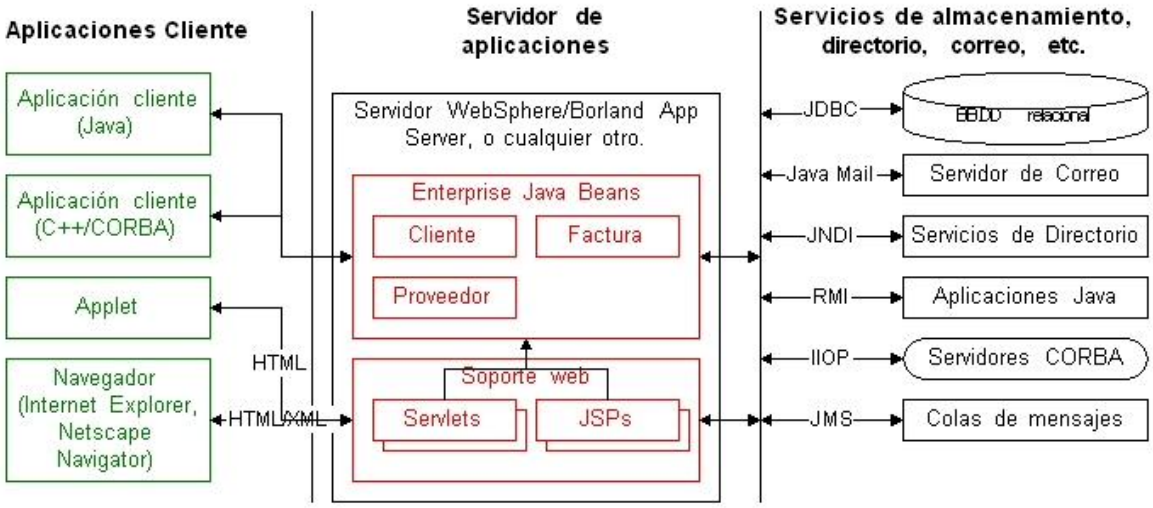


Figura 1.1: Arquitectura de la plataforma J2EE

Como se puede observar en la figura anterior, el servidor de aplicaciones está constituido principalmente por dos contenedores, el contenedor de Enterprise Java Beans (EJB Container) y el contenedor de componentes Web (Web Container), cabe aclarar que no todos los servidores J2EE tienen que cumplir estrictamente con la especificación, algunos de estos solo contienen uno de los contenedores, mientras que otros incorporan los dos.

Con un servidor de aplicaciones Java, se obtiene grandes beneficios, muchos de estos son producto del lenguaje como: independencia de plataforma, seguridad, escalabilidad, entre las principales.

2.2.2 Motor de Base de Datos Sybase

El motor de Base de Datos Sybase es parte del servidor empresarial Sybase Adaptive desarrollado por la empresa Sybase que fue fundada en 1984 y que es una plataforma para el manejo de datos que provee servicios de tiempo real para aplicaciones con gran volumen de transacciones.

La mayoría de aplicaciones de la ESPE de Intranet e Internet están desarrolladas para trabajar con esta Base de Datos.

2.3 Arquitectura de la aplicación

El estándar 1471 de IEEE define la arquitectura de software como la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el entorno, y los principios que guían su diseño y evolución.

La arquitectura estándar de la Dirección de Organización y Sistemas para aplicaciones WEB es una arquitectura de n capas, basada en Frameworks y mejores prácticas de la industria para aplicaciones WEB y consta de lo siguiente:

- Cliente liviano (Browser)
- Capa de presentación
- Capa de Negocio
- Capa de Persistencia de Datos
- Capa de Datos

Además se tiene un módulo de Autorización y Autenticación al que deberán acoplarse todos los sistemas desarrollados.

2.3.1 Framework

Un framework es una abstracción sistemática de la solución a un problema en un contexto y que puede ser reutilizable y de construcción basada en ciclos de experiencia; es decir, provee la solución de una manera consistente a un problema especifico cuyo objetivo es facilitar la creación de este tipo de aplicaciones al desarrollador.

Framework se los define como diseños reusables de todo o una parte del sistema de software descrito por un conjunto de clases abstractas y la manera cómo las instancias de esas clases colaboran entre sí.

Un Framework bien estructurado reduce los tiempos y costos de desarrollo considerando el alcance de la aplicación, ya que permite reutilizar tanto el diseño como el código. No requieren sucesiva incorporación de nueva tecnología ya que pueden ser implementados con lenguajes orientados a objetos existentes.

El desarrollo con Frameworks es tan aceptado debido a su fortaleza al permitir la reutilización de diseño y de código. Los Frameworks facilitan la generación de aplicaciones que se encuentran directamente relacionados con un dominio específico.

Con la aparición de los frameworks se hizo mucho más sencilla la generación de código, puesto que se abstrae la solución al problema específico y se crea una solución sistemática y estándar.

2.3.2 Esquema de autorización y Autenticación

Para que un sistema sea seguro debe cumplir con los requerimientos de Confiabilidad, Disponibilidad e Integridad. Además debe considerar conceptos como: autentificación, autorización y acceso, AAA por sus significados en inglés (Authentication, Authorization and Accounting)

Confiabilidad

Confiabilidad es la parte que asegura que la información sólo puede ser vista y utilizada por las partes que están autorizadas. La comunicación entre partes confiables se establece una vez que se ha realizado el proceso de autentificación.

Disponibilidad

Disponibilidad se refiere a que la información está en todo momento a través de un medio seguro para los usuarios que la requieran, siempre y cuando tengan los recursos y accesos necesarios. Se tendrá acceso a la información siempre y cuando el que la requiera, tenga previa autentificación los permisos y acreditaciones.

Autentificación

La autentificación es simplemente asegurarse que los usuarios son en realidad quien dicen ser. Cuando se utilizan recursos o se envían mensajes en una red privada amplia, la autentificación es lo más importante.

Autorización

Cada uno de los usuarios podrá acceder solamente a los recursos que le fueron permitidos. Ningún usuario puede hacer uso de aplicaciones que no le han sido asignadas y tampoco se pueden auto asignar derechos, recursos y tareas.

Control de Acceso

Cada usuario tiene un registro de los accesos, de manera que se pueda monitorear la información a la que se ingresa y las aplicaciones que son empleadas.

Integridad

Integridad es saber que la información enviada no ha sido alterada, modificada o eliminada en el transcurso del camino.

Una característica de seguridad que se logra con los conceptos anteriores es el No Repudio en la información. El receptor de información debe estar completamente seguro de que los datos recibidos son los auténticos enviados.

2.3.3 Capa de Presentación

La capa de presentación está definida como la capa de la aplicación donde se construye o genera la interfaz grafica que podrá ver el usuario.

En esta capa se resuelven cuestiones como:

- Navegabilidad del sistema, mapa de navegación.
- Formateo de los datos de salida: Resolución del formato más adecuado para la presentación de resultados. Está relacionado directamente con la internacionalización de la aplicación.
- Internacionalización: Los textos, etiquetas, y datos en general a presentar se obtendrán de uno u otro fichero de recursos en base al idioma preferido del navegador del usuario. En base a esta condición se ven afectadas las representaciones numéricas, las validaciones sobre los datos de entrada (coma decimal o punto decimal) y otros aspectos relativos al idioma del usuario remoto.
- Validación de los datos de entrada, en cuanto a formatos, longitudes máximas, entre otros.
- Interfaz gráfica con el usuario.

2.3.3.1 Model, View, Controler

La arquitectura MVC (Model/View/Controller) fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales son que el

Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. (figura 1.2)

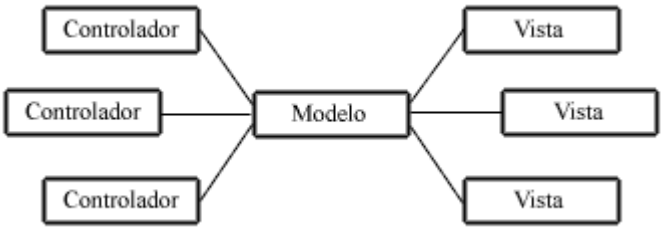


Figura 1.2: Modelo de Arquitectura

Este modelo de arquitectura presenta varias ventajas:

* Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado

* Hay un API muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.

* La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas.

2.3.4 Capa de Negocio

En esta capa es donde se deben implementar todas aquellas reglas obtenidas a partir del análisis funcional del proyecto. Así mismo, debe ser completamente independiente de cualquiera de los aspectos relacionados con la presentación de la misma. De esta forma, la misma capa de negocio debe poder ser empleada para una aplicación web común, una aplicación WAP, o una standalone.

Por otro lado, la capa de negocio ha de ser también completamente independiente de los mecanismos de persistencia empleados en la capa de acceso a datos. Cuando la capa de negocio requiera recuperar o persistir entidades o cualquier conjunto de información, lo hará siempre apoyándose en los servicios que ofrezca la capa de acceso a datos para ello.
De esta forma, la sustitución del motor de persistencia no afecta lo más mínimo a esta parte del sistema.

Las responsabilidades que conviene abordar en esta capa son:

- Implementación de los procesos de negocio identificados en el análisis del proyecto. Los procesos de negocio implementados en esta capa son totalmente independientes de cualquier aspecto relativo a la presentación de los mismos. De esta forma, si el usuario requiere la modificación de un aspecto de presentación de información se limitaría en la aplicación a una modificación de la capa de presentación.

- Control de acceso a los servicios de negocio. Dado que una misma aplicación puede contar con más de una capa de presentación al mismo tiempo, es aconsejable que la responsable última de ejecutar tareas sobre el control de acceso a los servicios del sistema no sea la capa de presentación, sino la de negocio. Los implementadores de la capa de negocio ni pueden ni deben confiar en que las futuras implementaciones de nuevas capas de presentación gestionen adecuadamente el acceso a los servicios de negocio. De esta forma, en cada invocación a cualquier método restringido de negocio se deberá comprobar por medio del sistema de autentificación adecuado, los derechos del usuario actual a realizar tal operación.
- Invocación a la capa de persistencia. Los procesos de negocio son los que determinan que, como y cuando se debe persistir en el repositorio de información. Los servicios ofertados por la interfaz de la capa de acceso a datos son invocados desde la capa de negocio en base a los requerimientos de los procesos en ella implementados.

2.3.5 Capa de Persistencia

Una capa de persistencia es un framework de Mapeo Objeto-Relacional lo cual permite abstraer la estructura de la BD a una estructura de objetos con lo cual el programador puede trabajar directamente con objetos evitando el manejo de SQL especifico.

La capa de persistencia es un framework aplicado específicamente al dominio de la abstracción del acceso a datos. El Mapeo Objeto-Relacional de la capa de persistencia es la especificación de cómo el objeto abstrae la tabla en la BD relacional, además de las relaciones entre tablas como relaciones entre objetos.

Antes del manejo de capas de persistencia, se tenía manejos dispares para accesos a la BD y de ejecución de sentencias SQL de acceso a datos. En implementaciones que no utilizan una capa de persistencia se debe considerar la creación de una conexión a la BD o el uso de un pool de conexiones. Una vez obtenida la conexión se crea una sentencia SQL, se ejecuta y obtiene los resultados iterando el ResultSet (Resultado de la consulta). Si se quería abstraer el acceso a la Base de datos se crea un manejador o Gestor que permite abstraer la lógica de acceso a la BD.

El utilizar una capa de persistencia presenta varias ventajas:

- Modelo de programación natural, el programador accede a los datos en forma de objetos en el mismo lenguaje con el que se encuentra trabajado, abstrayendo totalmente el lugar donde se encuentren almacenados o la manera de accederlos
- Minimiza la cantidad de líneas de código, evita la repetición o duplicación de código para acceso de datos.
- Oportunidades para el manejo de caches, se puede implementar lógica de manejo de caches en la capa de persistencia.

- Mapeo estructural más robusto cuando el modelo objeto-tabla cambia, existe la facilidad de que los cambios se aíslen en un solo punto.
- Permite delegar a la BD las actividades que realiza mejor y obtener todos los beneficios del trabajo con objetos para el programador.

Los sistemas de Base de Datos relacionales son adecuados para:

- Trabajar con grandes cantidades de datos: búsquedas, ordenamiento.
- Trabajar con grupos de datos: filtrar, juntar, agregar.
- Compartir: concurrencia, múltiples aplicaciones.
- Integridad: restricciones, insolación de transacciones.

Los modelos orientados a objetos permiten modelar de mejor manera objetos del mundo real, permitiendo tener relaciones entre objetos como el cerebro las entendería como un objeto teniendo características y conteniendo otro grupo de objetos.

2.3.5.1 JDBC

JDBC (Java Database Connectivity) es un API de Java para ejecutar sentencias SQL. Está formado por un conjunto de clases e interfaces programadas con el propio Java. (figura 1.3)

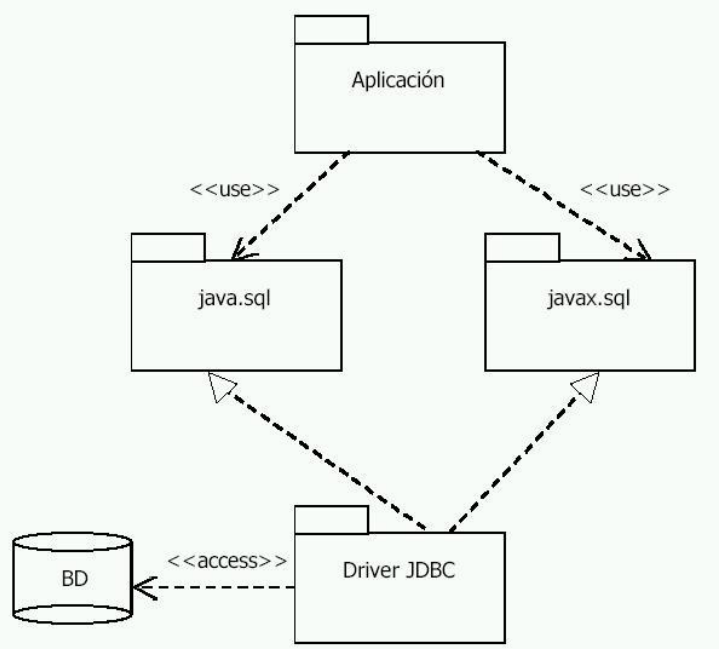


Figura 1.3: Esquema de conexión con Driver JDBC

JDBC consta de dos partes:

1. El paquete java.sql, que contiene las clases e interfaces que permiten acceder a la funcionalidad básica de JDBC. Forma parte de la edición estándar de J2SE.
2. El paquete javax.sql, que forma parte de J2EE, en el que se incluye funcionalidad avanzada del API.

Permite interactuar con bases de datos, de forma transparente al tipo de la misma. Es decir, es una forma única de programar el acceso a bases de datos desde Java, independiente del tipo de la base de datos. JDBC realiza llamadas directas a SQL.

Un driver JDBC es una capa de software que traduce las llamadas de JDBC a los APIs específicos del vendedor.

Existen cuatro categorías de drivers que soportan la conectividad JDBC, como se muestra en la siguiente figura:

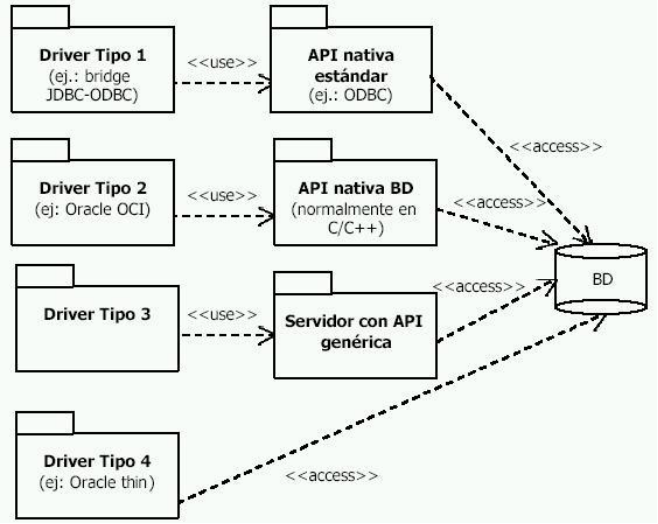


Figura 1.4: Esquema de tipos de drivers

Tipo 1: Puente JDBC-ODBC

Un driver tipo puente JDBC-ODBC delega todo el trabajo sobre un driver ODBC, que es quien realmente se comunica con la BD. El puente JDBC-ODBC permite la conexión desde Java a BD que no provean un driver JDBC.

La aplicación llama al gestor de driver JDBC, quien a su vez llama al driver JDBC (puente JDBC-DBC), quien llama al dirver ODBC, que es el que finalmente llama a la base de datos. El puente forma parte del jdk de SUN y esta en el paquete sun.jdbc.odbc.

Tipo 2: Parcialmente nativo; parte Java, parte driver nativo

Utiliza Java para realizar llamadas a una API nativa, sita en el cliente y dependiente de cada proveedor de BBDD, esta a su vez proporciona la conectividad con la BD. Por lo general son más rápidos que los de tipo 1.

Tipo 3: Servidor intermedio

Utilizan un servidor intermedio que proporciona conectividad a varios clientes Java con distintos SGBD. La aplicación Java conecta con este servidor intermedio mediante los protocolos de red suministrados por JDK, y es el servidor intermedio el que lo traduce a protocolos específicos del SGBD. No requiere software en la parte cliente.

Tipo 4: Java puro

Convierten las llamadas JDBC en llamadas directas utilizando protocolos específicos del SGBD. No necesitan instalar librerías adicionales, ni software intermedio.

2.3.5.2 Pool de Conexiones

Es una aplicación servidora con carga alta, como por ejemplo un contenedor de páginas JSP/Servlets, un contenedor de EJBs.

El Pool de conexiones fue creado como respuesta a la masiva solicitud que se puede realizar a la BD por minuto, así como a la formación de cuello de botella por estas peticiones.

A continuación se muestra un ejemplo de implementación.

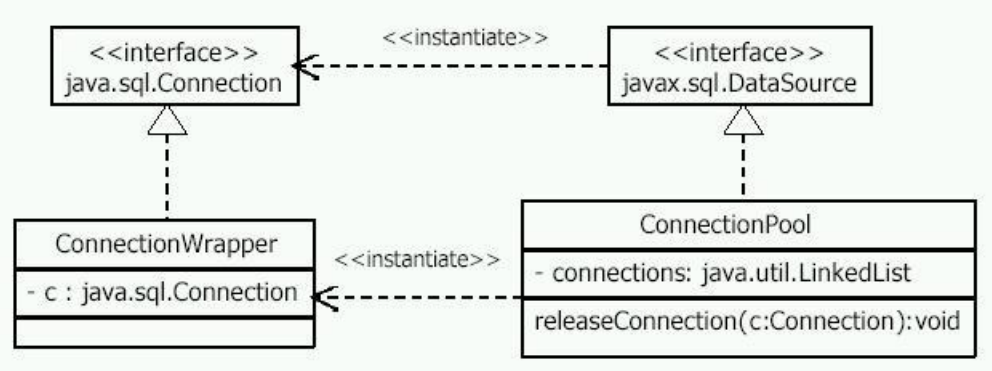


Figura 1.5: Pool de Conexiones

2.4 Metodología Orientada a objetos

Una apropiada metodología Orientada a Objetos debe tener los siguientes componentes: un ciclo de vida que permita adaptarse a las reglas de negocio y factibilidades tecnológicas; un conjunto completo de modelos y conceptos que son internamente consistentes; una colección de reglas y guías de desarrollo; una notación; un conjunto de técnicas para pruebas; un grupo de métricas apropiadas, estándares y estrategias de pruebas; identificación de reglas organizacionales, de reglas de negocios y programación; guía de manejo de proyectos y control de calidad; así como consejos sobre reutilización.

Una metodología de ingeniería del software es un proceso para producir software de una manera organizada, usando convenciones y técnicas de notación predefinidas. Desde que la comunidad de programación orientada a objetos tuvo la noción de incorporar el pensamiento de que los objetos son entidades coherentes con identidad estado y conducta, estos objetos pueden ser

organizados por sus similitudes y sus diferencias, puestas en uso en herencia y polimorfismo, las metodologías orientadas a objetos incorporan estos conceptos para definir sus reglas, normas, procedimientos, guías y notaciones para alcanzar un producto de calidad que satisfaga las necesidades del cliente

El análisis y diseño estructurado en los diagramas casi no proveen o proveen muy poco de mecanismos que hagan uso de componentes reusables. Mediante el mecanismo de herencia, las metodologías orientadas a objetos, promueven la reusabilidad, así los desarrolladores de software crean nuevas subclases que heredan atributos y métodos (funciones) de superclases previamente desarrollados.

Desde que se incorporó el pensamiento de que los objetos son entidades coherentes con identidad estado y conducta, estos objetos pueden ser organizados por sus similitudes y sus diferencias, puestas en uso en herencia y polimorfismo, las metodologías orientadas a objetos incorporan estos conceptos para definir sus reglas, normas, procedimientos, guías y notaciones para alcanzar un producto de calidad que satisfaga las necesidades del cliente.

Una metodología de desarrollo de software Orientada a Objetos consta de los siguientes elementos:

- Un ciclo de vida que permita adaptarse a las reglas de negocio y factibilidades tecnológicas
- Conjunto completo de modelos y conceptos internamente consistentes
- Colección de reglas y guías de desarrollo

- Notación
- Técnicas para pruebas
- Métricas apropiadas
- Estándares y estrategias de pruebas
- Identificación de reglas organizacionales, de reglas de negocios y programación
- Guía de manejo de proyectos y control de calidad

2.4.1 Tipos de Metodologías Orientadas a Objetos

En cuanto a las metodologías Orientadas a Objetos, hay un gran número de métodos orientados a objetos actualmente. Algunos de las metodologías más conocidas y estudiadas hasta antes del UML son:

- Object-Oriented Design (OOD), Booch.
- Object Modeling Technique (OMT), Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Object Oriented Systems Analysis (OOSA), Shaler y Mellor.
- Responsibility Driven Design (RDD), Wirfs-Brock, entre otros.

Actualmente las metodologías más importantes de análisis y diseño de sistemas han confluido en lo que se es el UML, bajo el respaldo del Object Management Group.

Se analizarán tres de las metodologías más utilizadas en la actualidad al momento de desarrollar una aplicación de software Orientado a Objetos, esta son: OMT, OOD, RUP.

2.4.1.1 Object Modeling Technique (OMT):

La metodología OMT (Object Modeling Technique) fue desarrollada por James Rumbaugh, esta metodología pone énfasis en la importancia del modelo y uso del mismo para lograr una abstracción, en el cual el análisis está enfocado en el mundo real para un nivel de diseño, también pone detalles particulares para modelado de recursos de la computadora.

Esta metodología puede ser aplicada en varios aspectos de implementación incluyendo archivos, base de datos relacionales, base de datos orientados a objetos. OMT está construido alrededor de descripciones de estructura de datos, constantes, sistemas para procesos de transacciones.

OMT pone énfasis en especificaciones declarativas de la información, para capturar los requerimientos, especificaciones imperativas para poder descender prematuramente en el diseño, declaraciones que permiten optimizar los estados, además provee un soporte declarativo para una directa implementación de DBMS[4].

[4] Data Base Manager System

Los puntos más importantes para esta metodología son los siguientes:

- Poner énfasis en el análisis y no en el desarrollo.
- Poner énfasis en los datos más que en las funciones, lo que proporciona estabilidad al proceso de desarrollo.
- Utilizar una notación común en todas las fases a través de tres modelos que capturan los aspectos estáticos, dinámicos y funcionales que combinados proveen una descripción completa del software.

La Metodología OMT divide el proceso de desarrollo en tres partes aisladas: análisis, diseño e implantación.

- *Análisis:* Su objetivo es desarrollar un modelo de lo que va a hacer el sistema. El modelo se expresa en términos de objetos y de relaciones entre ellos, flujo dinámico de control y las transformaciones funcionales.
- *Diseño:* Es la estrategia de alto nivel para resolver el problema y cómo construir una solución. Se define la arquitectura del sistema y se toman las decisiones estratégicas.
- *Implementación* En esta fase se convierte finalmente el diseño de objetos en código.

A su vez, cada una de estas fases se divide en subtareas, como son: modelos de objetos, dinámico y funcional; análisis y del sistema, y objetos del sistema.

- ***Modelo de Objetos**:* En esta primera parte del análisis se forma una primera imagen del modelo de clases del sistema con sus atributos y las

relaciones entre ellas, usando para ello un diagrama entidad relación modificado en el que además de las clases y sus relaciones se pueden representar también los métodos.

- ***Modelo Dinámico****:* El modelo dinámico usa un grafo para representar el comportamiento dinámico de cada clase, es decir, el comportamiento de estas ante cada evento que se produce en el sistema. Un evento desencadenará en un cambio de estado en la clase que se traducirá en una modificación de los atributos o relaciones de ésta.
- ***Modelo Funcional:*** Muestra que es lo que el sistema ha de hacer mediante un diagrama de flujo de datos, sin entrar en la secuencia temporal en la que los procesos se ejecutan. El modelo funcional puede revelar nuevos objetos y métodos que se pueden incorporar en los dos modelos anteriores. Por eso se dice que el método OMT es iterativo.
- ***Diseño del Sistema***: Se centra en la parte física del sistema como la descomposición de éste en subsistemas, el tipo de entorno en el que se va a ejecutar, el manejo de recursos y el almacenamiento de datos.
- ***Diseño de los objetos:*** Determina que operaciones van a realizar los métodos y profundiza incluso los algoritmos que va a usar. Se escogen los distintos tipos de representación de datos y se subdivide en módulos que pasarán a formar parte de la implementación.

2.4.1.2 Object Oriented Design - Grady Booch

La metodología Booch (OOD), es una metodología de propósito general en la que se parte de que cada etapa no es un proceso aislado si no que ha de

interactuar con sus siguientes y precedentes en una especie de bucle del que se sale cuando se esté satisfecho con el modelo conseguido. En un principio se tienen una serie de objetos y clases que forman el sistema, a continuación se construye el modelo de interfaz y se examinan las relaciones entre las clases lo que, a su vez, genera la adición de nuevos interfaces que generarán nuevas relaciones iterándose hasta llegar al estado de refinamiento deseado. El método Booch proporciona un conjunto de herramientas gráficas y notaciones que ayudan a representar visualmente los modelos definidos en las fases de análisis y diseño. Algunas de ellas son:

Diagramas de clase: Es una variación de los diagramas de entidad relación en los que se añaden nuevos tipos de relaciones como la herencia, instanciación y uso. Además permite agrupar las clases y relaciones en categorías para diagramas demasiado complejos.

Diagramas de objetos: En este tipo de gráfico de muestran los objetos y sus relaciones de forma dinámica mostrando la forma en la que los objetos se pasan mensajes entre ellos. Así mismo, en esto diagramas es posible representar la visibilidad de los objetos siendo ésta la que determina que objetos se pueden comunicar con otros.

Diagramas temporales: Muestran la secuencia temporal de creación y destrucción de objetos. Suelen ir acompañados de pseudocódigo en el que se explica el flujo de mensajes de control entre los objetos del sistema.

Diagramas de transición de estados: Permiten definir como las instancias de las clases pasan de un estado a otro a causa de ciertos eventos y que acciones se desencadenan de esos cambios de estado.

Diagramas de módulo y proceso: En Booch, en la fase de implementación, es posible representar mediante estos gráficos la parte física del sistema, es decir, podemos mostrar como se van a almacenar internamente las clases y objetos, relaciones entre módulos en tiempo de compilación, procesos, dispositivos y las comunicaciones entre ellos.

2.4.1.3 Rational Unified Process (RUP)

Rational Unified Process o Proceso Unificado de Desarrollo es el resultado de tres décadas de desarrollo y uso práctico. Es un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Mas que un simple proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de dominios.
Rational Unified Process (RUP), es la metodología estándar de la industria para la construcción completa del ciclo de ingeniería de software, tanto para sistemas tradicionales como para sistemas web.

Esta metodología permite mayor productividad en equipo y la realización de mejores prácticas de software a través de plantillas y herramientas que guían en todas las actividades de desarrollo crítico del software. RUP unifica las disciplinas en lo que a desarrollo de software se refiere, incluyendo modelado de negocio,

manejo de requerimientos, componentes de desarrollo, ingeniería de datos, manejo y configuración de cambios, y pruebas, cubriendo todo el ciclo de vida de los proyectos basado en la construcción de componentes y maximizando el uso del UML (Unified Modeling Language). El software en construcción está formado por componentes interconectados a través de interfaces.

Los puntos principales en los que se basa RUP son los siguientes:

Casos de Uso:

Los casos de uso representan los requisitos funcionales de la aplicación a ser desarrollada; en otras palabras, qué es lo que debe hacer el sistema.

Arquitectura del producto:

El concepto de arquitectura de software incluye los aspectos estáticos y dinámicos más significativos del sistema. Hay que tomar en cuenta que tanto la arquitectura como los casos de uso deben ser generados en paralelo, pues los casos de uso deben encajar en la arquitectura, así como la arquitectura debe permitir que los casos de uso se lleven a cabo.

Ciclo de vida Iterativo Incremental:

El ciclo de vida Iterativo Incremental, consiste en dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencias a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. Para una efectividad máxima, las

iteraciones deben estar controladas; esto es, deben seleccionarse y ejecutarse de una forma planificada.

El RUP se sostiene en los tres puntos básicos anteriores. Para hacer que funcionen, se necesitan un proceso polifacético, que tenga en cuenta ciclos, fases, flujos de trabajo, gestión del riesgo, control de calidad, gestión del proyecto y control de la configuración. El RUP ha establecido un Framework que integra todas esas diferentes facetas.

2.4.2 Lenguaje de Modelado Unificado (UML)

El Lenguaje de Modelado Unificado (UML) es la sucesión de una serie de métodos de análisis y diseños orientados a objetos que aparecen a fines de los 80's y principios de los 90s. Directamente unifica los métodos de Booch, Rumbaugh (OMT), y Jacobson.

UML es llamado un lenguaje de modelado, no un método. Los métodos consisten de ambos de un lenguaje de modelado y de un proceso. El lenguaje de modelado es la notación (principalmente gráfica) que usan los métodos para expresar un diseño. El proceso indica los pasos que se deben seguir para llegar a un diseño.

La estandarización de un lenguaje de modelado es invaluable, ya que es la parte principal de comunicación. Si se quiere discutir un diseño con alguien más,

ambos deben conocer el lenguaje de modelado y no así el proceso que se siguió para obtenerlo.

Una de las metas principales de UML es avanzar en el estado de la industria proporcionando herramientas de interoperabilidad para el modelado visual de objetos. Sin embargo para lograr un intercambio exitoso de modelos de información entre herramientas, se requirió definirle una semántica y una notación.

UML es un Lenguaje de Modelado Unificado basado en una notación gráfica la cual permite: especificar, construir, visualizar y documentar los objetos de un sistema programado. Este lenguaje es el resultado de la unificación de los métodos de modelado orientados a objetos de Booch, Rumbaugh (OMT: Object Modeling Technique) y Jacobson (OOSE: Object-Oriented Sotfware Engineering).

El UML modela los sistemas mediante el uso de objetos que forman parte de él así como, las relaciones estáticas o dinámicas que existen entre ellos, puede ser utilizado por cualquier metodología de análisis y diseño orientada por objetos para expresar los diseños.

La notación puede ser presentada desde 3 diferentes perspectivas:

1. La que proporcionan los elementos que la constituyen. Los elementos se clasifican en estructurales, de comportamiento, de agrupamiento y de documentación.
2. La que introducen las relaciones entre los elementos de la notación.

3. Los diagramas en los que se organizan los elementos.

Existen nueve tipos de diagramas con los que modelar las distintas vistas de una aplicación:

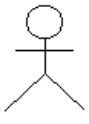
1. Diagramas de casos de uso.
2. Diagramas de clases.
3. Diagramas de objetos.
4. Diagramas de secuencia.
5. Diagramas de colaboración.
6. Diagramas de estado.
7. Diagramas de actividad.
8. Diagramas de componentes.
9. Diagramas de implantación.

Este último suele ser el preferido para introducir la notación, ya que permite presentar los elementos en un orden semejante al de su utilización durante el proceso de desarrollo.

Diagramas de casos de uso

- *Definición*.- Un diagrama de Casos de Uso muestra las distintas operaciones que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones).
- *Caso de Uso.-* Se representa en el diagrama por una elipse, denota un requerimiento solucionado por el sistema. Cada caso de uso es una operación

completa desarrollada por los actores y por el sistema en un diálogo. El conjunto de casos de uso representa la totalidad de operaciones desarrolladas por el sistema. Va acompañado de un nombre significativo

- *Actor.-* Es un usuario (en el sentido amplio de la palabra: ser humano, o aplicación) del sistema, que necesita o usa algunos de los casos de uso. Se representa mediante un  actor, acompañado de un nombre significativo, si es necesario.
- *Relaciones en un diagrama de casos de uso.-* Entre los elementos de un diagrama de Casos de uso se pueden presentar tres tipos de relaciones, representadas por líneas dirigidas entre ellos (del elemento dependiente al independiente):

    - Comunica (communicates). Relación entre un actor y un caso de uso, denota la participación del actor en un caso de uso determinado. En el diagrama anterior todas las líneas que salen del actor denotan este tipo de relación. Es la relación por defecto.

    - Usa (uses). Relación entre dos casos de uso, denota la inclusión del comportamiento de un escenario en otro.
    - Extiende (extends). Relación entre dos casos de uso, se utiliza cuando un caso de uso es una especialización de otro

Diagrama de Clases

- *Definición.* Un diagrama de estructura estática muestra el conjunto de clases y objetos importantes que conforman un sistema, junto con las relaciones

existentes entre los mismos. Estos diagramas reciben su denominación por presentar de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases. Esta viene expresada por sus relaciones con el resto de clases que se han identificado en el modelo que se ha desarrollado

- *Clase:* Representada por un rectángulo (por lo general con tres divisiones internas), son los elementos fundamentales del diagrama. Una clase describe un conjunto de objetos con características y comportamiento idéntico. Los tres compartimientos estándares del rectángulo se utilizan (de arriba a abajo) para identificar la clase normalmente con el nombre de la clase, mostrar la estructura del estado de los objetos que representan mediante declaración de sus atributos, y del interfaz de comunicación de dichos objetos a través de la declaración de los métodos que podrán ser utilizados en el paso de mensajes.
- *Atributo:* Identifica las características propias del estado de los objetos de cada clase. Los nombres empleados hacen referencia por lo general a tipos simples, ya que los atributos de tipos compuestos se suelen representar mediante asociaciones de composición con otras clases. La sintaxis de un atributo es:

visibility name : type-expression = initial-value { property-string }

Donde *visibility* es uno de los siguientes:

+ *public visibility*

*protected visibility*

- *private visibility*

type-expression es el tipo del atributo con nombre *name*.

El atributo puede especificarse dotado de un valor inicial y de un conjunto de propiedades.

- *Operación* El conjunto de operaciones (métodos) describen el comportamiento de los objetos de una clase. La sintaxis de una operación en UML es:

  visibility name (parameter-list) : return-type-expression { property-string }

  Cada uno de los parámetros en *parameter-list* se denota igual que un atributo. Los demás elementos son los mismos encontrados en la notación de un atributo.

- *Asociación (rol, multiplicidad)* Una asociación en general es una línea que une dos o más símbolos. Pueden tener varios tipos de representaciones, que definen su semántica y características. Los tipos de asociaciones entre clases presentes en un diagrama estático son:

  - Asociación binaria
  - Asociación n-aria
  - Composición
  - Generalización
  - Refinamiento

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

- Rol: Identificado como un nombres al los finales de la línea, describe la semántica de la relación en el sentido indicado.
- Multiplicidad: Describe la cardinalidad de la relación.

- *Asociación binaria* Se identifica como una línea sólida que une dos clases. Representa una relación de algún tipo entre las dos clases, no muy fuerte (es decir, no se exige dependencia existencial, ni encapsulamiento).
- *Composición* Es una asociación fuerte, que implica tres cosas:
    - Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
    - Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene
    - Los objetos contenidos no son compartidos; esto es, no forman parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación.

Existe también una relación de composición menos fuerte (no se exige dependencia existencial y permite partición) que es denotada por una un rombo sin rellenar en uno de los extremos.

- *Generalización* La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.
- *Clase paramétrica* Una clase paramétrica representa el concepto de clase genérica en los conceptos básicos Orientados a Objetos o de *template* en C++. Se dibuja como una clase acompañada de un rectángulo en la esquina superior derecha, con los parámetros del caso.

- *Paquete* Un paquete es una forma de agrupar clases (u otros elementos en otro tipo de diagramas) en modelos grandes. Pueden tener asociaciones de dependencia o de generalización entre ellos.
- *Dependencia* Denota una relación semántica entre dos elementos (clases o paquetes) del modelo. Indica que cambiar el elemento independiente puede requerir cambios en los dependientes. Se muestra como una línea punteada direccional, indicando el sentido de la dependencia. Puede tener asociados ciertos estereotipos que den una explicación del tipo de dependencia presentada.
- *Nota* Es un comentario dentro de un diagrama. Puede estar relacionado con uno o más elementos en el diagrama mediante líneas punteadas. Suele representar aclaraciones al diagrama o restricciones sobre los elementos relacionados. Se representa mediante un rectángulo con su borde superior derecho doblado.

Diagrama de Secuencia

- *Definición* Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, como también muestra el uso de los mensajes de las clases diseñadas en el contexto de una operación.
- *Línea de vida de un objeto* Un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos. El rectángulo de

encabezado contiene el nombre del objeto y el de su clase, en un formato nombreObjeto: nombreClase.

- *Activación* Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto.
- *Mensaje* El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.
- *Tiempos de transición* En un ambiente de objetos concurrentes o de demoras en la recepción de mensajes, es útil agregar nombres a los tiempos de salida y llegada de mensaje
- *Condiciones caminos alternativos de ejecución. Concurrencia* En algunos casos sencillos pueden expresarse en un diagrama de secuencia alternativas de ejecución. Estas alternativas pueden representar condiciones en la ejecución o diferentes hilos de ejecución (threads).
- *Destrucción de un objeto* Se representa como una X al final de la línea de ejecución del objeto.
- *Métodos recursivos* Es un rectángulo un poco salido de la activación principal y con líneas de llamado de mensajes, que indican la entrada y salida de la recursividad.

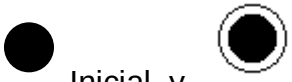
Diagrama de Colaboración

Un diagrama de colaboración es una forma de representar interacción entre objetos, alterna al diagrama de secuencia. A diferencia de los diagramas de secuencia, pueden mostrar el contexto de la operación (cuáles objetos son atributos, cuáles temporales) y ciclos en la ejecución.

Diagrama de Estados

Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro

- *Estado* Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto esta esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente). Se marcan también los estados iniciales y finales mediante los símbolos:

 Inicial y Final

- *Eventos* Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una de varias cosas:

- Condición que toma el valor de verdadero o falso
- Recepción de una señal de otro objeto en el modelo
- Recepción de un mensaje
- Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular

El nombre de un evento tiene alcance dentro del paquete en el cual está definido, no es local a la clase que lo nombre.

- *Envío de mensajes* Además de mostrar una transición de estados por medio de eventos, puede representarse el momento en el cual se envían mensajes a otros objetos. Esto se realiza mediante una línea punteada dirigida al diagrama de estados del objeto receptor del mensaje
- *Transición simple* Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas. Se representa como una línea sólida entre dos estados, que puede venir acompañada de un texto con el siguiente formato:

  event-signature [í guard-condition] / action-expression ^ send-clause

  event-signature es la descripción del evento que da a lugar la transición, guard-condition son las condiciones adicionales al evento necesarias para que la transición ocurra,

  action-expression es un mensaje al objeto o a otro objeto que se ejecuta como resultado de la transición y el cambio de estado y

  send-clause son acciones adicionales que se ejecutan con el cambio de estado, por ejemplo, el envío de eventos a otros paquetes o clases

- *Transición interna* Es una transición que permanece en el mismo estado, en vez de involucrar dos estados distintos. Representa un evento que no causa cambio de estado. Se denota como una cadena adicional en el compartimiento de acciones del estado.
- *Subestados* Un estado puede descomponerse en subestados, con transiciones entre ellos y conexiones al nivel superior. Las conexiones se ven al nivel inferior como estados de inicio o fin, los cuales se suponen conectados a las entradas y salidas del nivel inmediatamente superior
- *Transición compleja* Una transición compleja relaciona tres o más estados en una transición de múltiples fuentes y/o múltiples destinos. Representa la subdivisión en threads del control del objeto o una sincronización. Se representa como una línea vertical del cual salen o entran varias líneas de transición de estado
- *Transición a estados anidados* Una transición de hacia un estado complejo (descrito mediante estados anidados) significa la entrada al estado inicial del subdiagrama. Las transiciones que salen del estado complejo se entienden como transiciones desde cada uno de los subestados hacia afuera (a cualquier nivel de profundidad).

Diagrama de Actividad

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados son estados de acción (identifican que acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la acción ejecutada en el estado anterior. Puede dar detalle a un caso de

uso, un objeto o un mensaje en un objeto. Sirven para representar transiciones internas, sin hacer mucho énfasis en transiciones o eventos externos.

- *Estado de acción* Representa un estado con acción interna, con por lo menos una transición que identifica la culminación de la acción (por medio de un evento implícito). No deben tener transiciones internas ni transiciones basadas en eventos (Si este es el caso, se lo debe representar en un diagrama de estados). Permite modelar un paso dentro del algoritmo. Se representan por un rectángulo con bordes redondeados.
- *Transiciones* Las flechas entre estados representan transiciones con evento implícito. Pueden tener una condición en el caso de decisiones.
- *Decisiones* Se representa mediante una transición múltiple que sale de un estado, donde cada camino tiene una etiqueta distinta. Se representa mediante un diamante al cual llega la transición del estado inicial y del cual salen las múltiples transiciones de los estados finales.

Diagrama de Componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean éstos componentes fuentes, binarios o ejecutables.

Los componentes de software tienen tipo, que indica si son útiles en tiempo de compilación, enlace o ejecución. Se consideran en este tipo de diagramas solo tipos de componentes. Instancias específicas se encuentran en el diagrama de ejecución.

Se representa como un grafo de componentes de software unidos por medio de relaciones de dependencia (generalmente de compilación). Puede mostrar también contenencia de entre componentes software e interfaces soportadas.

Diagrama de Implantación

Un diagrama de implantación muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software, procesos y objetos que se ejecutan en ellos. Instancias de los componentes de software representan manifestaciones en tiempo de ejecución del código. Componentes que solo sean utilizados en tiempo de compilación deben mostrarse en el diagrama de componentes.

Un diagrama de implantación es un grafo de nodos conectados por asociaciones de comunicación. Un nodo puede contener instancias de componentes de software, objetos, procesos (un caso particular de un objeto). Las instancias de componentes de software pueden estar unidos por relaciones de dependencia, posiblemente a interfaces.

- *Nodos* Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Pueden representarse instancias o tipos de nodos. Se representa como un cubo 3D en los diagramas de implementación.

- *Componentes* Un componente representa una unidad de código (fuente, binario o ejecutable) que permite mostrar las dependencias en tiempo de compilación y ejecución. Las instancias de componentes de software muestran unidades de software en tiempo de ejecución y generalmente ayudan a identificar sus dependencias y su localización en nodos. Pueden mostrar también que interfaces implementan y qué objetos contienen. Su representación es un rectángulo atravesado por una elipse y dos rectángulos más pequeños.

2.4.3 Metodología XP (eXtreme Programming)

Esta práctica se presenta como una metodología de trabajo, y no tanto como una metodología de orientación a objetos.

Su filosofía se basa en satisfacer las necesidades del cliente, y la reevaluación de los requerimientos en períodos relativamente cortos.

2.4.3.1 Características de XP

Las características esenciales que se pueden mencionar de esta metodología son:

- Comunicación: Los programadores están en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que después de

concretar los requisitos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.

- Simplicidad: Codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se quieren ampliar resulta imposible hacerlo y se tienen que desechar y partir de cero.
- Realimentación (Feedback): Mediante la realimentación se ofrece al cliente la posibilidad de conseguir un sistema apto a sus necesidades ya que se le va mostrando el proyecto a tiempo para poder ser cambiado y poder retroceder a una fase anterior para rediseñarlo a su gusto.
- Coraje: Se debe tener coraje o valentía para cumplir los tres puntos anteriores; Hay que tener valor para comunicarse con el cliente y enfatizar algunos puntos, a pesar de que esto pueda dar sensación de ignorancia por parte del programador, hay que tener coraje para mantener un diseño simple y no optar por el camino más fácil y por último hay que tener valor y confiar en que la realimentación sea efectiva.

2.4.3.2 Fases y Reglas de XP

Existen de manera general 4 fases de XP, y estas son planificación, diseño, codificación y pruebas.

1ª Fase: Planificación del proyecto

Historias de usuario. El primer paso de cualquier proyecto que siga la metodología X.P es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso pero con algunas diferencias: Constan de 3 ó 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia.

Release planning: Después de tener ya definidas las historias de usuario es necesario crear un plan de publicaciones, en inglés "Release plan" (Planificación de publicaciones), donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones.

Un "Release plan" es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que serán implementadas y las historias que serán implementadas en cada versión del programa. Después de un "Release plan" tienen que estar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las historias que se deben desarrollar en cada versión), el tiempo que tardarán en desarrollarse y publicarse las versiones del programa, el

número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado.

Iteraciones. Todo proyecto que siga la metodología X.P se ha de dividir en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada iteración los clientes deben seleccionar las historias de usuario definidas en el "Release planning" que serán implementadas. También se seleccionan las historias de usuario que no pasaron el test de aceptación que se realizó al terminar la iteración anterior. Estas historias de usuario son divididas en tareas de entre 1 y 3 días de duración que se asignarán a los programadores.

Velocidad del proyecto. La velocidad del proyecto es una medida que representa la rapidez con la que se desarrolla el proyecto; estimarla es muy sencillo: basta con contar el número de historias de usuario que se pueden implementar en una iteración; de esta forma, se sabrá el cupo de historias que se pueden desarrollar en las distintas iteraciones. Usando la velocidad del proyecto controlaremos que todas las tareas se puedan desarrollar en el tiempo del que dispone la iteración. Es conveniente reevaluar esta medida cada 3 ó 4 iteraciones y si se aprecia que no es adecuada hay que negociar con el cliente un nuevo "Release Plan".

Programación en pareja. La metodología X.P aconseja la programación en parejas pues incrementa la productividad y la calidad del software desarrollado. El trabajo en pareja involucra a dos programadores trabajando en el mismo equipo; mientras uno codifica haciendo hincapié en la calidad de la función o método que

está implementando, el otro analiza si ese método o función es adecuado y está bien diseñado. De esta forma se consigue un código y diseño con gran calidad.

Reuniones diarias. Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo tiene que tener voz y voto.

2ª Fase: Diseño

Diseños simples: La metodología X.P sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e implementable que a la larga costará menos tiempo y esfuerzo desarrollar.

Glosarios de términos: Usar glosarios de términos y una correcta especificación de los nombres de métodos y clases ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reusabilidad del código.

Riesgos: Si surgen problemas potenciales durante el diseño, X.P sugiere utilizar una pareja de desarrolladores para que investiguen y reduzcan al máximo el riesgo que supone ese problema.

Funcionalidad extra: Nunca se debe añadir funcionalidad extra al programa aunque se piense que en un futuro será utilizada. Sólo el 10% de la misma es utilizada, lo que implica que el desarrollo de funcionalidad extra es un desperdicio de tiempo y recursos.

Refactorizar. Refactorizar es mejorar y modificar la estructura y codificación de códigos ya creados sin alterar su funcionalidad. Refactorizar supone revisar de nuevo estos códigos para procurar optimizar su funcionamiento. Es muy común reusar códigos ya creados que contienen funcionalidades que no serán usadas y diseños obsoletos. Esto es un error porque puede generar códigos completamente inestables y muy mal diseñados; por este motivo, es necesario refactorizar cuando se va a utilizar código ya creado.

Tarjetas C.R.C. El uso de las tarjetas C.R.C (Class, Responsabilities and Collaboration) permiten al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación procedural clásica.

3ª Fase: Codificación

El cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de X.P. A la hora de codificar una historia de usuario su presencia es aún más necesaria. Los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada.

La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

Crear test que prueben el funcionamiento de los distintos códigos implementados ayudará a desarrollar dicho código. Crear estos test antes, ayuda a saber qué es exactamente lo que tiene que hacer el código a implementar y que una vez implementado pasará dichos test sin problemas ya que dicho código ha sido diseñado para ese fin. Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crearán primero los test para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco se conseguirá un desarrollo que cumpla todos los requisitos especificados.

X.P opta por la programación en pareja ya que permite un código más eficiente y con una gran calidad.

X.P sugiere un modelo de trabajo usando repositorios de código dónde las parejas de programadores publican cada pocas horas los códigos implementados y corregidos junto a los test que deben pasar. De esta forma el resto de programadores que necesiten códigos ajenos trabajarán siempre con las últimas versiones. Para mantener un código consistente, publicar un código en un repositorio es una acción exclusiva para cada pareja de programadores.

X.P también propone un modelo de desarrollo colectivo en el que todos los programadores están implicados en todas las tareas; cualquiera puede modificar

o ampliar una clase o método de otro programador si es necesario y subirla al repositorio de código. El permitir al resto de los programadores modificar códigos que no son suyos no supone ningún riesgo ya que para que un código pueda ser publicado en el repositorio tiene que pasar los test de funcionamiento definidos para el mismo.

La optimización del código siempre se debe dejar para el final. Hay que hacer que funcione y que sea correcto, más tarde se puede optimizar.

X.P afirma que la mayoría de los proyectos que necesiten más tiempo extra que el planificado para ser finalizados no podrán ser terminados a tiempo se haga lo que se haga, aunque se añadan más desarrolladores y se incrementen los recursos. La solución que plantea X.P es realizar un nuevo "Release plan" para concretar los nuevos tiempos de publicación y de velocidad del proyecto.

4ª Fase: Pruebas

Uno de los pilares de la metodología X.P es el uso de test para comprobar el funcionamiento de los códigos que se vayan implementando. El uso de los test en X.P es el siguiente:

- Se deben crear las aplicaciones que realizarán los test con un entorno de desarrollo específico para test.
- Hay que someter a tests las distintas clases del sistema omitiendo los métodos más triviales.
- Se deben crear los test que pasarán los códigos antes de implementarlos.

Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará. Hay que crear los test abstrayéndose del futuro código, de esta forma asegurar la independencia del test respecto al código que evalúa.

Los distintos test se deben subir al repositorio de código acompañados del código que verifican. Ningún código puede ser publicado en el repositorio sin que haya pasado su test de funcionamiento, de esta forma, se asegura el uso colectivo del código.

El uso de los test es adecuado para observar la refactorización. Los test permiten verificar que un cambio en la estructura de un código no tiene porqué cambiar su funcionamiento.

Test de aceptación. Los test mencionados anteriormente sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario. Para asegurar el funcionamiento final de una determinada historia de usuario se deben crear "Test de aceptación"; estos test son creados y usados por los clientes para comprobar que las distintas historias de usuario cumplen su cometido.

III. Análisis

3.1 Análisis de la situación actual

La modalidad de estudios presencial, dispone de un sistema para control académico desarrollado para la Intranet como cliente-servidor en PowerBuilder, pero debido al crecimiento institucional y al avance tecnológico se hace necesario implementar varios de estos procesos para que funcionen en el Internet. Además, en el sitio web de la ESPE se encuentran funcionando algunos servicios como son: consultas de notas de los diferentes parciales, consultas de record académico, consultas de horarios de estudios al momento de matricularse, entre otros.

Estos servicios existentes se han realizado en diferentes lenguajes como PHP, LotusNotes, ASP y Java, sin tener ninguna relación entre estos sistemas exceptuando la fuente de datos y teniendo en cada uno diferente forma o procedimiento para autorización y autenticación, o no teniendo implementado ninguno de estos sistemas. El que estén en diferentes lenguajes hace muy difícil su estandarización, integración y modificaciones.

En la modalidad presencial se cuenta con 4 campus dispersos por el país:

- Sangolquí (Varias facultades)
- Latacunga (Varias facultades)
- Santo Domingo (IASA2)

- Quito (Idiomas y Héroes del Cenepa)

Cada uno de estos campus se maneja con estructuras de base de datos iguales, pero instancias de Base de Datos diferentes. Es decir se cuenta con 5 bases de datos diferentes que no tienen conexión alguna, pero que tienen la mismas tablas y campos, también datos diferentes para cada campus.

Por esta razón, cualquier sistema que se desarrolle, debe tener la capacidad para manejar todas estas diferentes bases de datos o fuentes de datos, o solo se puede acceder a uno de ellos con la instalación de la aplicación. Esto ha impedido el poder compartir servicios o aplicaciones entre los diferentes campus, limitando el desarrollo de servicios igualitario y marginando a los campus más apartados.

Esta situación ocasiona que exista: duplicación de funciones, desperdicio de recursos, prolongación de trámites, entre otros.

En el modelo de datos actual existe cierta redundancia de campos y datos en las diferentes tablas, causado por las modificaciones y nuevos desarrollos que han existido en los sistemas realizados en PowerBuilder; pero se trabajará con el mismo modelo de datos como parte de los requerimientos de este sistema ya que estos seguirán en uso y no se desea causar ninguna disrupción.

Se ha proyectado la implementación del modelo de autorización y autenticación que debe ser utilizado como parte de este proyecto, por lo cual se

participará en el análisis y diseño del mismo, puesto que se prevee integrar las modalidades de Educación a Distancia y Presencial, con sus diferentes campus.

La autorización y autenticación se la realizará por medio de la cédula del usuario, en el caso de la autorización además por medio de la clave, y en el caso de la autenticación será en base al rol que seleccione el usuario.

Los usuarios podrán tener diferentes roles, ya que un usuario puede ser docente y además ser o haber sido estudiante, al seleccionar el rol se presentará su menú de acceso e internamente en la aplicación se configurará la fuente de datos que se va ha utilizar.

3.2 Recolección de la Información

La información recolectada fue provista en su totalidad del personal que labora en el departamento de Organización y Sistemas, ellos proporcionaron todos los modelos de datos, datos de pruebas, explicación de tablas y explicación de los procesos de negocio. Además proporcionaron los estándares de arquitectura e información sobre versiones de los aplicativos en los que debe correr el sistema.

También se tomó el estándar de diseño del sitio web de la ESPE (www.espe.edu.ec). Y se trabajó de manera similar en formularios y navegación al sistema de servicios ya realizado en la MED.

3.3 Definición de servicios virtuales

Dentro de los servicios que se brindará para docentes y estudiantes de la modalidad presencial con sus respectivas sedes, y que forman parte del control de calificaciones y servicios virtuales, están los siguientes:

- ***Servicios Académicos Estudiantes:*** estos servicios están disponibles para los estudiantes que han pertenecido en algún momento a alguna de las facultades de la modalidad presencial de la ESPE:

    - Actualización de datos personales: (Datos Académicos) se desplegaran los datos personales del estudiante y se le permitirá actualizarlos.
    - Consulta de las materias que el estudiante puede tomar para la matrícula (prematricula): (Matrícula Presencial) se presentaran las materias en las que el estudiante se puede matricular por facultad.
    - Consulta de materias matriculadas con sus respectivos docentes en el período vigente: (Impresión de comprobante) se le permitirá al estudiante imprimir su comprobante de matricula por facultad.
    - Consulta de Notas del período vigente: (Notas Presencial) Se desplegarán las notas del período activo de las materias en las cuáles el estudiante se encuentre matriculado.
    - Consulta del récord académico de las diferentes sedes y facultades en las que el estudiante haya realizado sus estudios: (Record Académico) Se desplegará el Record Académico del estudiante por facultad,

aunque éste no se encuentre matriculado en un período activo, además se presentará la opción de imprimir este record de acuerdo con los estándares de Organización y Sistemas.

- Pago de Matrícula por TODO1: permitirá conexión entre el sistema de la ESPE y el sistema del Botón de pagos de Todo1 para que el estudiante pueda realizar el pago de su matrícula en línea.

- ***Servicios Académicos Docentes:*** estos servicios estarán disponibles para los docentes que tengan asignado este rol para alguna de las facultades de la ESPE en modalidad presencial:

  - Ingreso de calificaciones por materia y paralelo: (Administración de Notas) el docente podrá ingresar las notas de sus alumnos en una facultad, materia y paralelo específicos.

    Hay que tomar en cuenta dos particularidades:

    1. En modalidad presencial alumnos de las diferentes facultades asisten de manera conjunto a clases en los primeros niveles en lo que se denomina Ciencias Básicas por lo que es necesario realizar un ingreso especial de notas para ciencias básicas en donde además se presente a que facultad pertenece el alumno.
    2. Además en lo que son las materias humanísticas (como son Pintura, Escultura, Literatura, etc.), también asisten estudiantes de las diferentes facultades de modalidad presencial, y por lo tanto es necesario hacer un ingreso especial de notas donde se presente la facultad a la que pertenece el alumno.

- Ingreso de calificaciones por estudiante e Impresión de cartillas por materia y paralelo (Ver Cartilla de Notas): el docente podrá seleccionar la faculta, materia y paralelo para la que desea imprimir la cartilla y se presentará la opción de imprimir esta cartilla de acuerdo con los estándares de Organización y Sistemas.

  Hay que tomar en cuenta que existirán las opciones de impresión de cartillas, también, para las materias de Ciencias Básicas y de las materias Humanísticas, agrupadas con todos sus estudiantes sin importar la facultad de la que provengan.

3.4 Definición de la Metodología

Dentro de la ingeniería del software una metodología es un proceso que ayuda a producir software de una manera organizada, usando convenciones y técnicas de notación predefinidas, para transformar los requerimientos del usuario en un sistema de software. Existen varias metodologías para el desarrollo de Software, el presente trabajo se enfocará en las Metodologías Orientadas a Objetos, y como notación el Lenguaje de Modelado Unificado UML.

La metodología utilizada se fundamenta en una perspectiva Iterativa - Incremental basada en priorización de Riesgos, que consiste en identificar los riesgos del proyecto, priorizarlos y en base a esto planificar iteraciones (romper el proyecto en mini proyectos completos). Con cada iteración añadirá mayor valor, claridad y funcionalidad al proyecto. La priorización de riesgos permite descubrir

desde el inicio los problemas más complicados del sistema y atacarlos al inicio, para así reducir la posibilidad de que el desarrollo se complique posteriormente.

3.5 Definición de la Arquitectura de la Aplicación

La arquitectura definida por Organización y Sistemas para aplicaciones web, es la siguiente:

Todas las aplicaciones deben integrarse al módulo de autorización y autenticación, el cual provee las pantallas de inicio, y un sitio seguro, además de proveer la selección de perfil para el usuario.

La capa de presentación esta realizada con JSP modelo 1 es decir, que no tiene un manejo especifico para modelo MVC por medio de un framework, sino que se trabaja con paginas JSP que se encargan del flujo de la aplicación (Controlador) y de la llamada a la capa de negocio (Modelo), es decir no hay una clara separación entre estas dos responsabilidades.

La capa de negocio esta representada como clases Java que generalmente no tienen estado, y que se los llama Manejadores y en donde se encapsulan las reglas de negocio y las llamadas a la Capa de persistencia.

La capa de persistencia esta representada por 2 elementos:

1. Data Transfer Object (DTO) es un contenedor de datos con un mapeo a objeto de los campos de la tabla, es decir un registro de la

BD y que no contiene nada de lógica de negocio, y sus únicos métodos son getters y setters.

2. Gestor es una implementación de una clase abstracta que se encarga del manejo de la conexión y de las operaciones de persistencia y recuperación, haciendo transparente para los gestores que la implementan estos manejos.

Esto hace de la capa de persistencia un framework, que permite realizar la implementación de persistencia por tabla en la BD, haciendo el manejo granular de datos y permitiendo un mapeo objeto-relacional adecuado. En la implementación del gestor el programador solo debe preocuparse del mapeo entre campos al guardar y recuperar.

El DTO permite pasar información entre las diferentes capas de la aplicación, es decir, por ejemplo, el Gestor de Notas nos permite recuperar registros de notas dada la cédula de un estudiante, estos registros se los pasa a la capa de negocio donde por medio de otros gestores se recupera información de periodo, docente u otra que se necesite y todos estos datos se pasan a la capa de presentación para ser mostrados al usuario.

3.6 Diagramas de Casos de Uso

3.6.1 Definición de actores

Los siguientes son los actores del sistema:

- Estudiantes de modalidad presencial
- Profesores de modalidad presencial
- Módulo de Autorización y Autenticación
- Sistema de pago en línea de Todo1 (Banco del Pichincha)

3.6.2 Autorizar y Autenticar

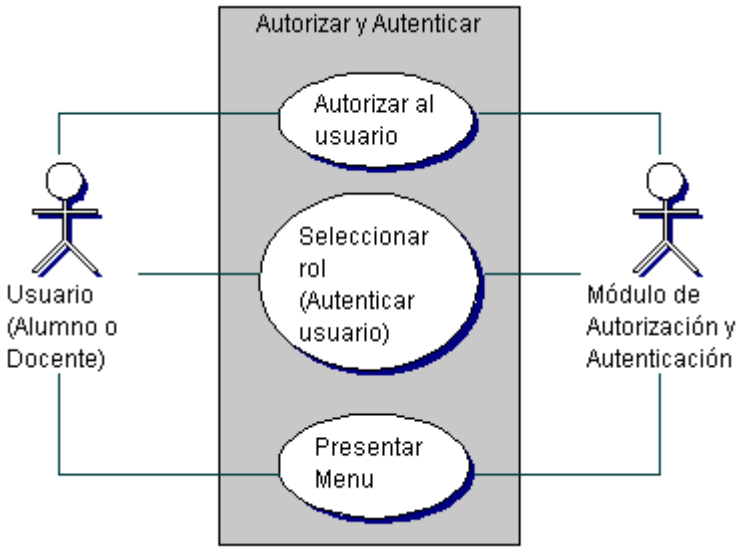


Figura 3.1: Caso de Uso para Autorizar y Autenticar

Descripción:

1. El usuario, sea este docente o estudiante, ingresa al sistema colocando su cédula y clave (Autorización).
2. Se presenta los roles que ha sido asignados al usuario, y el usuario selecciona uno de estos roles (Autenticación).
3. Se presente el menú con las opciones que tiene disponible el usuario para este rol.

Variación: Falla en Autorización

2a. El usuario no pudo ser autorizado y no puede ingresar al sistema, esto puede ser causado por error en el ingreso de usuario y clave o porque el usuario no existe como usuario del sistema.

3.6.3 Actualización de Datos Académicos

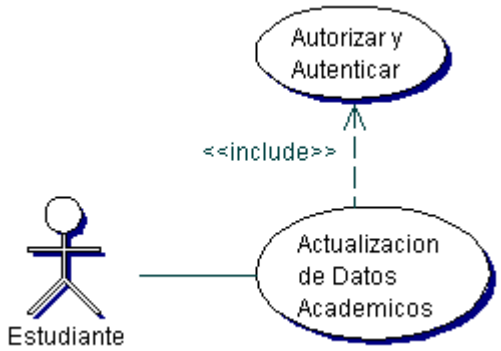


Figura 3.2: Caso de Uso para Actualización de Datos Académicos

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El estudiante puede actualizar sus datos como son nombres, edad, correo electrónico y cambiar su clave, esto ayuda a mantener actualizado el sistema.

3.6.4 Consultar Record Académico

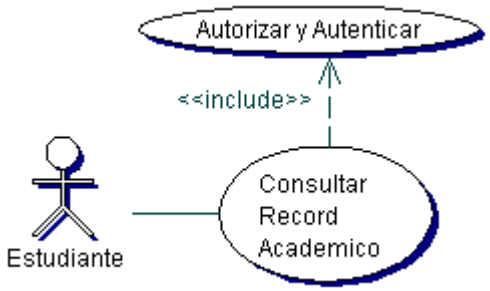


Figura 3.3: Caso de Uso para Consultar Record Académico

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El estudiante selecciona la facultad de la que desea obtener el record académico.
2. Se visualiza el record académico para los periodos pasados del estudiante, de la facultad seleccionada.
3. Se permite seleccionar la opción de impresión del record y lleva a una pantalla con formato amigable a impresión.

Variación: No existen facultades en el listado

1a. El estudiante esta en su primer periodo en la ESPE y por lo tanto no tiene record académico en ninguna facultad.

3.6.5 Consultar Notas Presencial

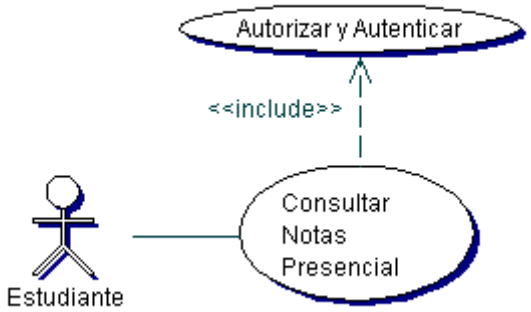


Figura 3.4: Caso de Uso para Consultar Notas Presencial

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El estudiante selecciona la facultad de la que desea obtener sus notas para el período actual.
2. Se visualizan las notas del período actual para las materias de la facultad seleccionada, en las que se encuentra inscrito el estudiante.
3. Se permite seleccionar la opción de impresión de las notas y lleva a una pantalla con formato amigable a impresión.

Variación: No está matriculado en ninguna facultad

1a. El estudiante no se encuentra matriculado en ninguna facultad en la ESPE en el período actual y por lo tanto no tiene notas.

3.6.6 Realizar Matricula presencial

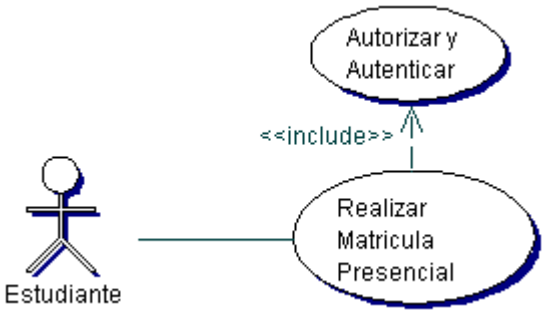


Figura 3.5: Caso de Uso para Realizar Matricula presencial

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El estudiante selecciona la facultad en la que desea matricularse.
2. Se visualizan las materias en las que el estudiante puede matricularse en esta facultad, las materias se presentan con su horario y paralelo.
3. Una vez seleccionadas las materias en que desea matricularse el estudiante se presenta un detalle de las materias seleccionadas.
4. Al aceptar la matriculación se presenta una pantalla con el resumen de la matricula para ser impreso (Similar al comprobante de pago).

Variación: No está registrado en ninguna facultad

1a. El estudiante no se encuentra registrado en ninguna facultad en la ESPE como estudiante y por lo tanto no puede matricularse.

3.6.7 Pagar Matricula presencial

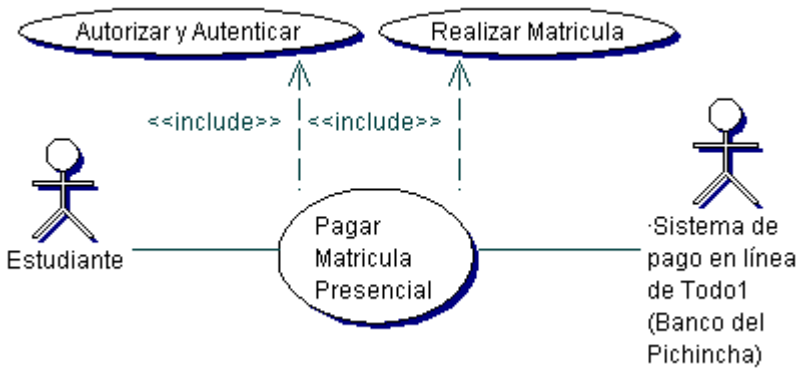


Figura 3.6: Caso de Uso para Pagar Matricula presencial

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.
2. Haber realizado la matricula presencial en la facultad que se desea pagar.

Descripción:

1. El estudiante selecciona la facultad de la que desea pagar la matricula.
2. Se presenta su comprobante de pago.
3. El sistema redirecciona al estudiante al sistema de pago en línea de Todo 1 (Banco del Pichincha), en donde el estudiante realiza el pago del valor del comprobante de pago.
4. Una vez finalizado el proceso de pago en Todo 1, este sistema redirecciona a una pagina del sistema donde se presenta un recibo de pago que puede ser impreso por el alumno.

Variación: No está registrada la matricula en ninguna facultad

1a. El estudiante no ha registrado su matrícula en ninguna facultad en la ESPE como estudiante y por lo tanto no puede pagar la matrícula.

3.6.8 Imprimir Comprobante de Pago

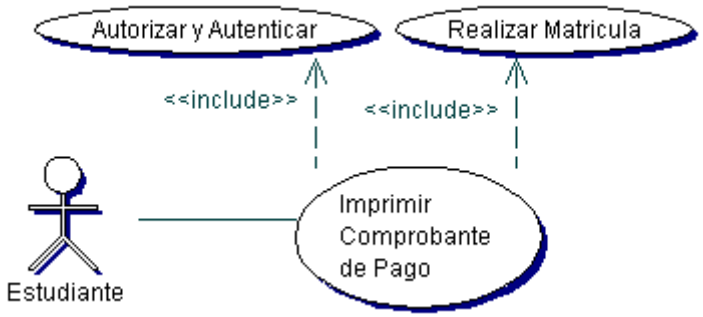


Figura 3.7: Caso de Uso para Imprimir Comprobante de Pago

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.
2. Haber realizado la matricula presencial en la facultad que se desea pagar.

Descripción:

1. El estudiante selecciona la facultad en la que desea imprimir el comprobante de pago de la matricula.
2. Se presenta el comprobante de pago para la matricula registrada en la facultad seleccionada.

3. Se permite seleccionar la opción de impresión del comprobante y lleva a una pantalla con formato amigable a impresión.

Variación: No esta registrada la matricula en ninguna facultad

1a. El estudiante no ha registrado su matricula en ninguna facultad en la ESPE como estudiante y por lo tanto no puede pagar la matricula.

3.6.9 Administración de Notas (Presencial)

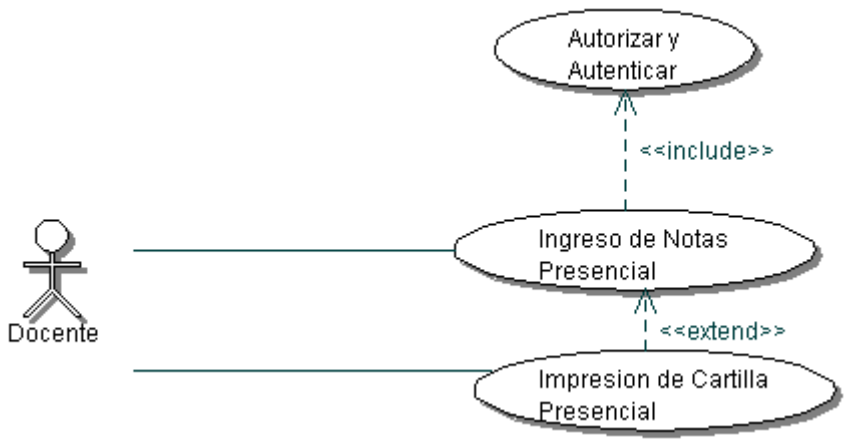


Figura 3.8: Caso de Uso para Administración de Notas presencial

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El docente selecciona la facultad, materia y paralelo del cual desea administrar las notas.

2. Se presenta el listado de alumnos matriculados en esa facultad, materia y paralelo, en este punto se presentarán tantos campos de notas como períodos se hayan insertado para esta materia y esta facultad. Se encontrarán activos para el ingreso de notas por parte del docente solo los períodos que estén vigentes para el ingreso.
3. Existirá un botón para almacenar a la BD las notas ingresadas, en este punto se debe validar que las notas sean correctas en formato y rango.

Variación: No está registrada ninguna materia para el docente

1a. El secretario académico de cada facultad debe ingresar las materias y paralelos para los docentes, en este caso el docente no tendrá materias que seleccionar para el ingreso de notas.

Variación: desea imprimir cartilla

2b. El docente selecciona la opción de impresión de la cartilla, se presenta una lista de los estudiantes con sus notas que no son editables, para la facultad, materia y paralelo seleccionados.

3b. Se permite seleccionar la opción de impresión de la cartilla y lleva a una pantalla con formato amigable a impresión.

Variación: almacenamiento automático de las notas ingresadas por tiempo

3c. Cuando ha pasado un periodo determinado sin que el docente envíe las notas a ser almacenadas en la BD, el sistema automáticamente envía estas notas para ser almacenadas.

3.6.10 Administración de Notas (Humanísticas)

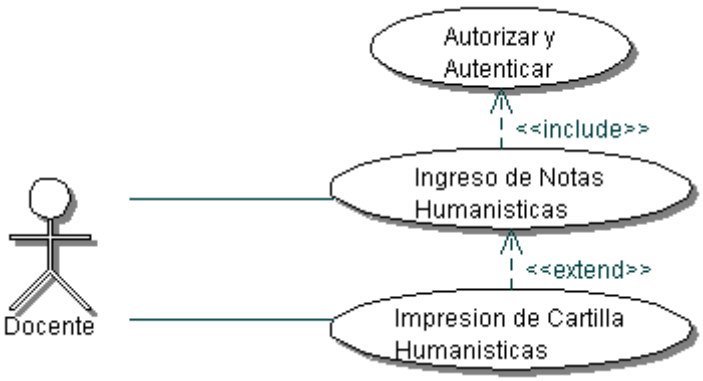


Figura 3.9: Caso de Uso para Administración de Notas Humanísticas

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El docente selecciona la materia humanística y paralelo del cual desea administrar las notas.
2. Se presenta el listado de alumnos matriculados en esa materia humanística y paralelo, en este punto se presentan tantos campos de notas como períodos se hayan insertado para esta materia en la facultad predeterminada para las materias humanísticas. Se muestran activos para el ingreso de notas por parte del docente solo los periodos que estén vigentes para el ingreso. Se presenta la facultad a la que

pertenece el estudiante, ya que en las materias humanísticas se encontrarán matriculados estudiantes de diferentes facultades.

3. Existe un botón para almacenar a la BD las notas ingresadas, en este punto se debe validar que las notas sean correctas en formato y rango.

Variación: No está registrada ninguna materia para el docente

1a. El secretario académico en la facultad predeterminada para las materias humanísticas, debe ingresar las materias y paralelos para los docentes, en este caso al docente no tendrá materias que seleccionar para el ingreso de notas.

Variación: desea imprimir cartilla

2b. El docente selecciona la opción de impresión de la cartilla, se presenta un lista de los estudiantes con sus notas que no son editables, para la materia humanística y paralelo seleccionados.

3b. Se permite seleccionar la opción de impresión de la cartilla y lleva a una pantalla con formato amigable a impresión.

Variación: almacenamiento automático de las notas ingresadas por tiempo

3c. Cuando ha pasado un periodo determinado sin que el docente envíe las notas a ser almacenadas en la BD, el sistema automáticamente envía estas notas para ser almacenadas.

3.6.11 Administración de Notas (Ciencias Básicas)

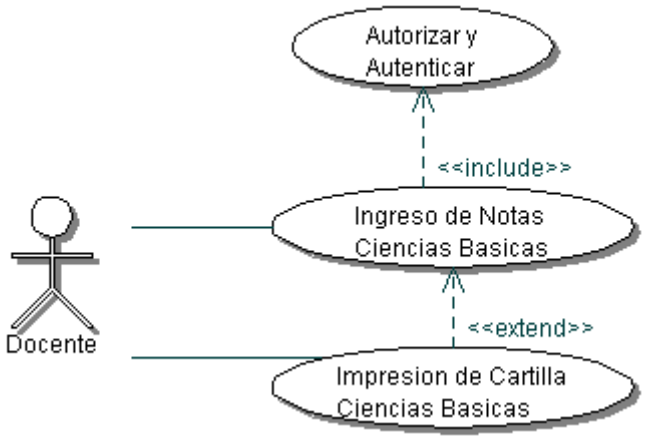


Figura 3.10: Caso de Uso para Administración de Notas Ciencias Básicas

Pre-condición:

1. Pasar por la Autorización y Autenticación del sistema.

Descripción:

1. El docente selecciona la materia de ciencias básicas y paralelo del cual desea administrar las notas, aquí además se presentan las facultades para las que se dicta esta materia.
2. Se presenta el listado de alumnos matriculados en esa materia de ciencias básicas y paralelo, en este punto se presentan tantos campos de notas como períodos se hayan insertado para esta materia en la facultad seleccionada de Ciencias Básicas. Se muestran activos para el ingreso de notas por parte del docente, solo los períodos que estén vigentes para el ingreso. Se debe presentar la facultad a la que

pertenece el estudiante, ya que en las materias ciencias básicas se encuentran matriculados estudiantes de diferentes facultades.

3. Existe un botón para almacenar a la BD las notas ingresadas, en este punto se debe validar que las notas sean correctas en formato y rango.

Variación: No está registrada ninguna materia para el docente

1a. El secretario académico en la facultad seleccionada de Ciencias Básicas debe ingresar las materias y paralelos para los docentes, en este caso al docente no tendrá materias que seleccionar para el ingreso de notas.

Variación: desea imprimir cartilla

2b. El docente selecciona la opción de impresión de la cartilla, se presenta un lista de los estudiantes con sus notas que no son editables, para la materia de ciencias básicas y paralelo seleccionados.

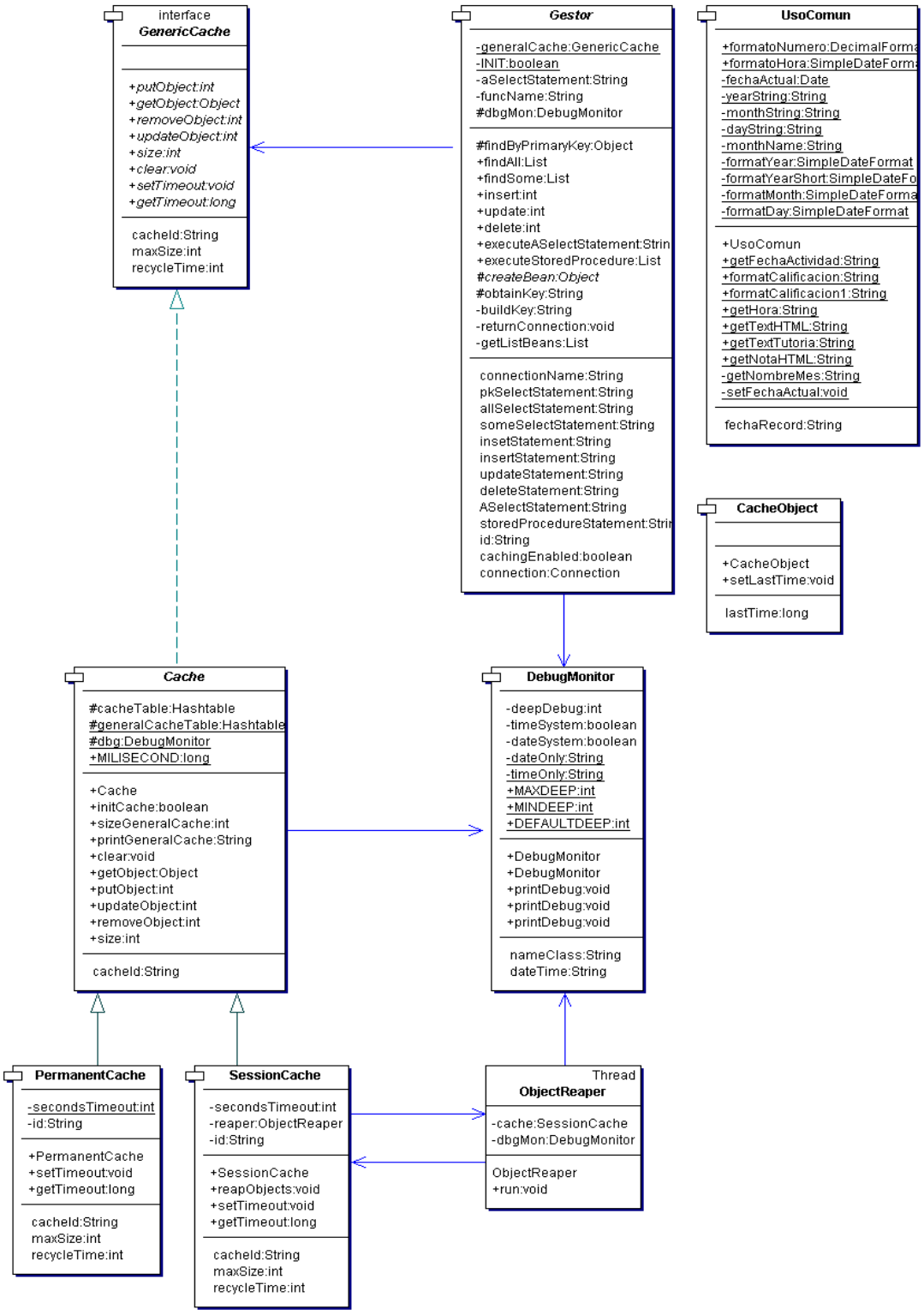
3b. Se permite seleccionar la opción de impresión de la cartilla y lleva a una pantalla con formato amigable a impresión.

Variación: almacenamiento automático de las notas ingresadas por tiempo

3c. Cuando ha pasado un período determinado sin que el docente envíe las notas a ser almacenadas en la BD, el sistema automáticamente envía estas notas para ser almacenadas.

IV. DISEÑO

4.1 Diagramas de Clases



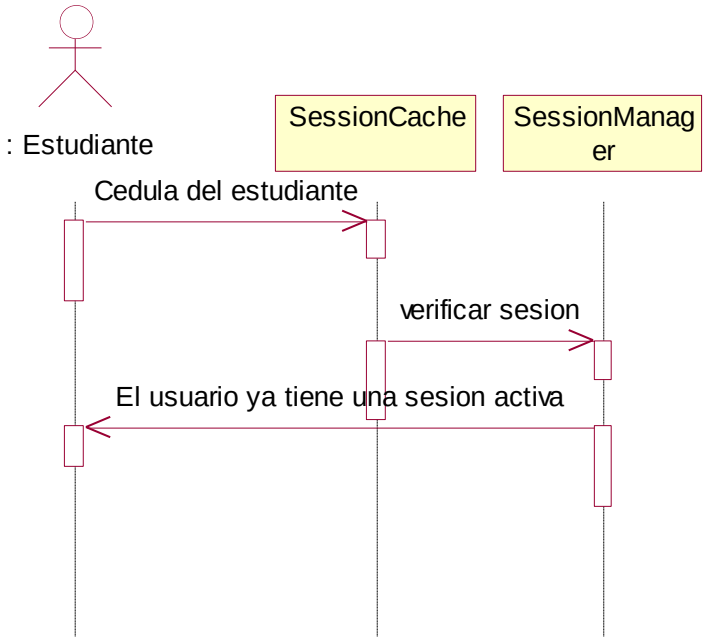
4.2 Diagramas de Interacción

4.2.1 Diagramas de Secuencia

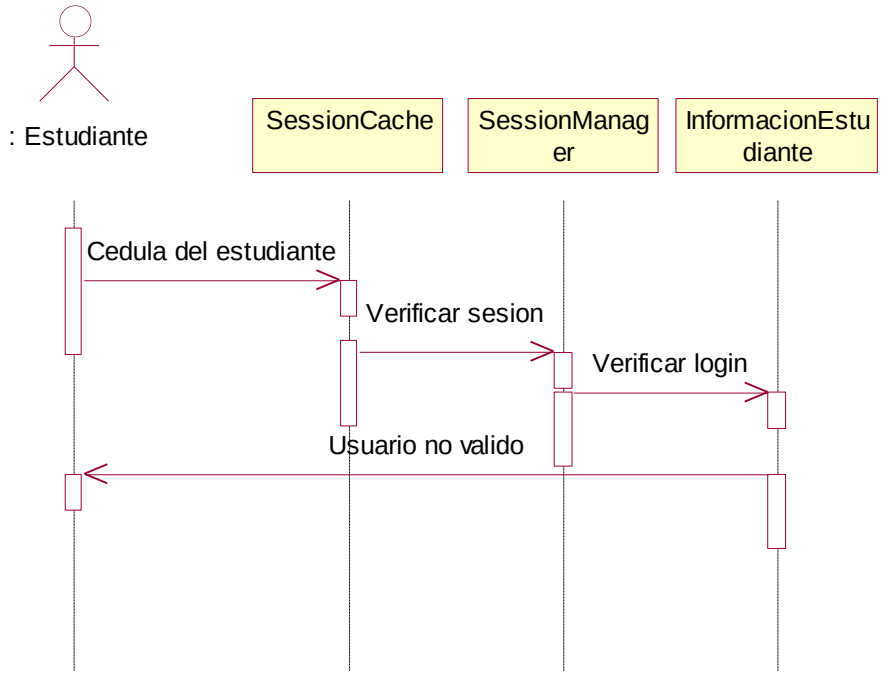
Control

a. Estudiante

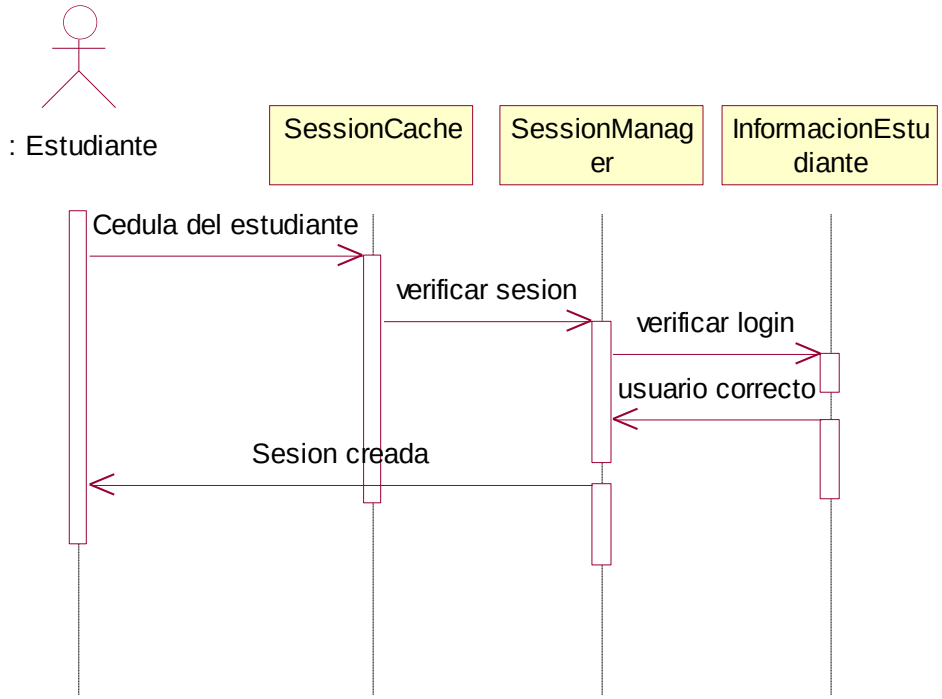
El usuario ya ha iniciado una sesión



El login es incorrecto

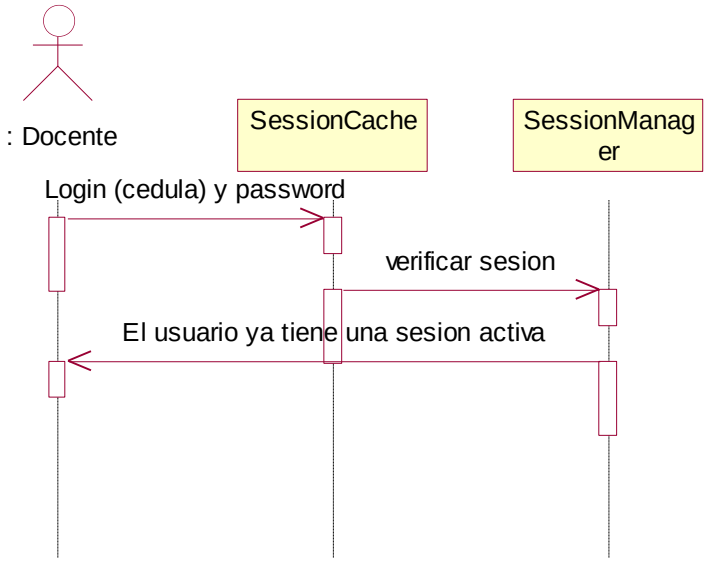


Correcto inicio de sesión

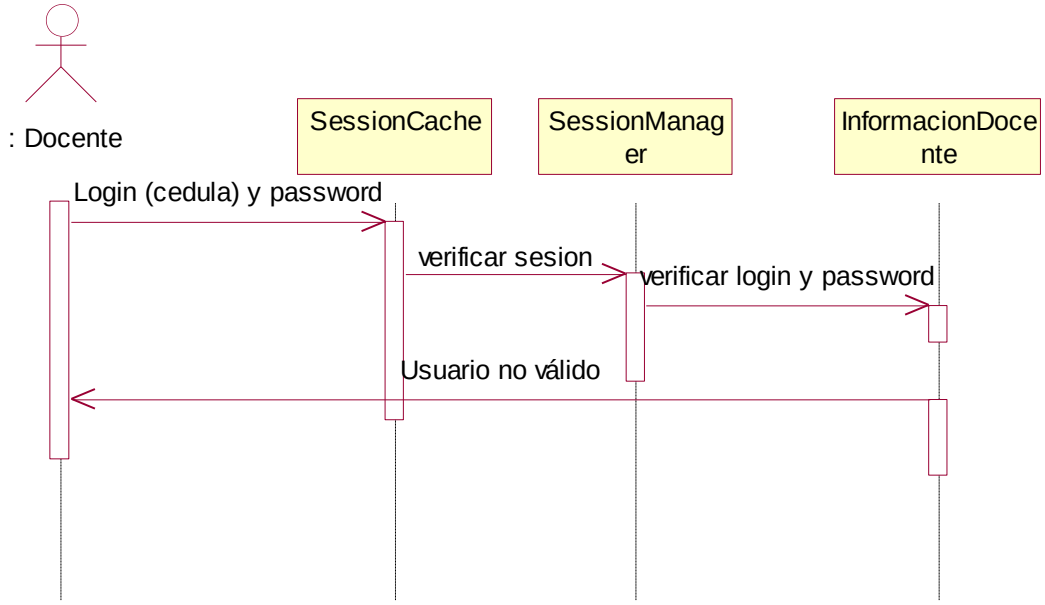


b.Docente

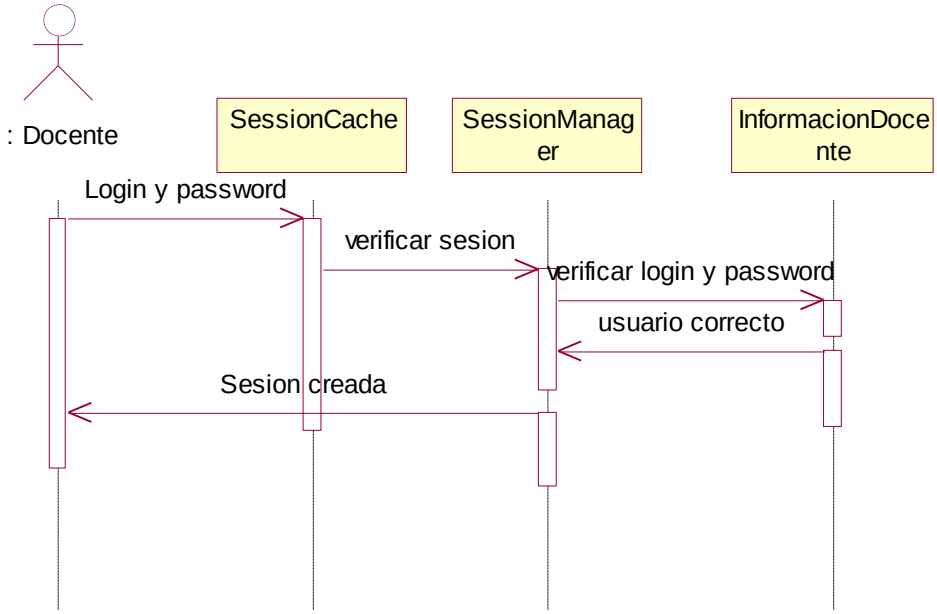
El usuario ya ha iniciado una sesión



El login y/o password es incorrecto

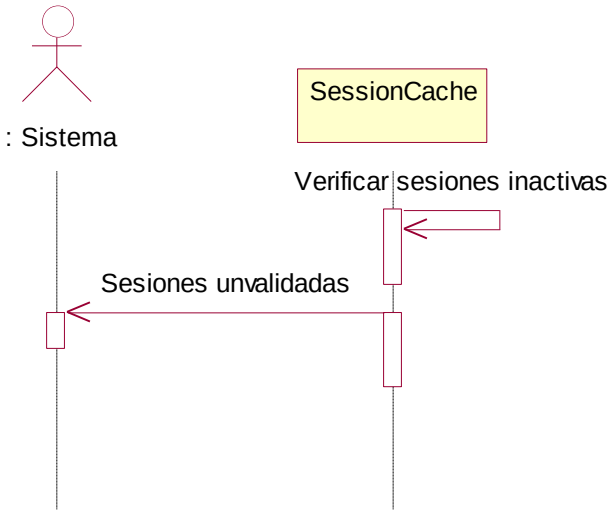


Correcto inicio de sesión



c. Estudiante y Docente

Invalidar sesiones inactivas por más de cinco minutos

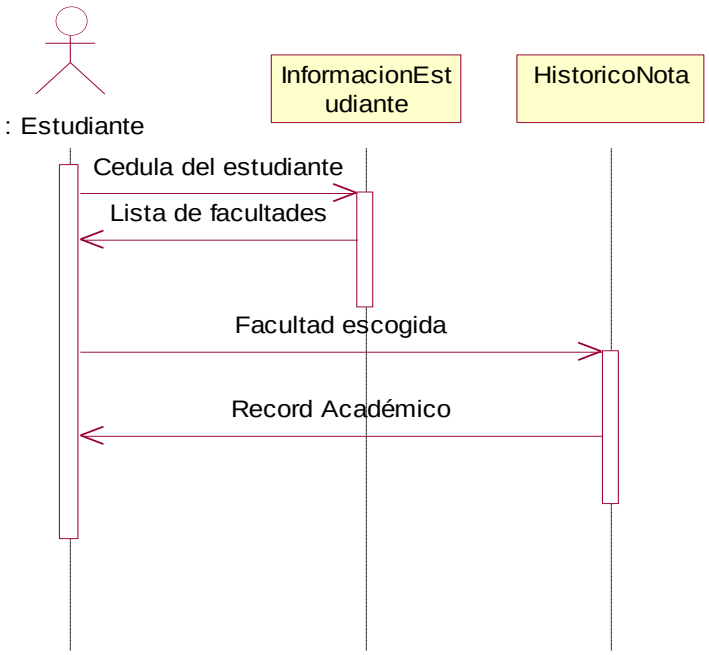


Servicios Virtuales

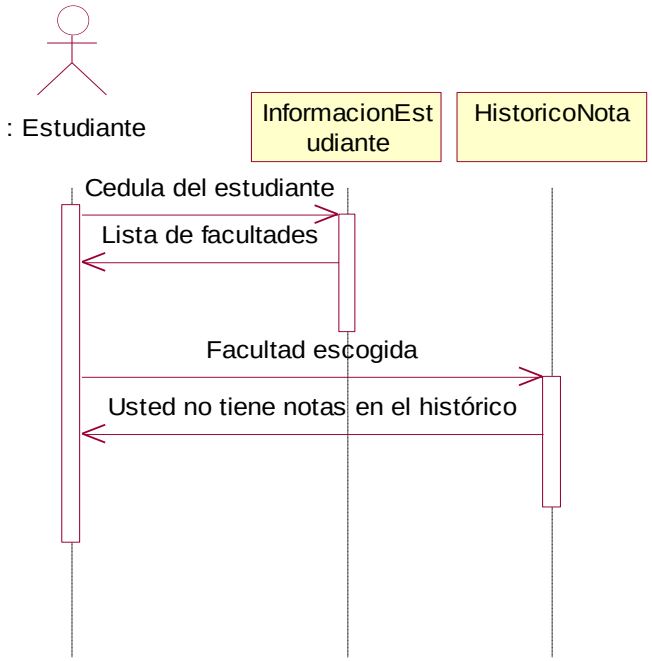
a.Estudiantes

Consulta de Récord Académico

Correcta consulta del record

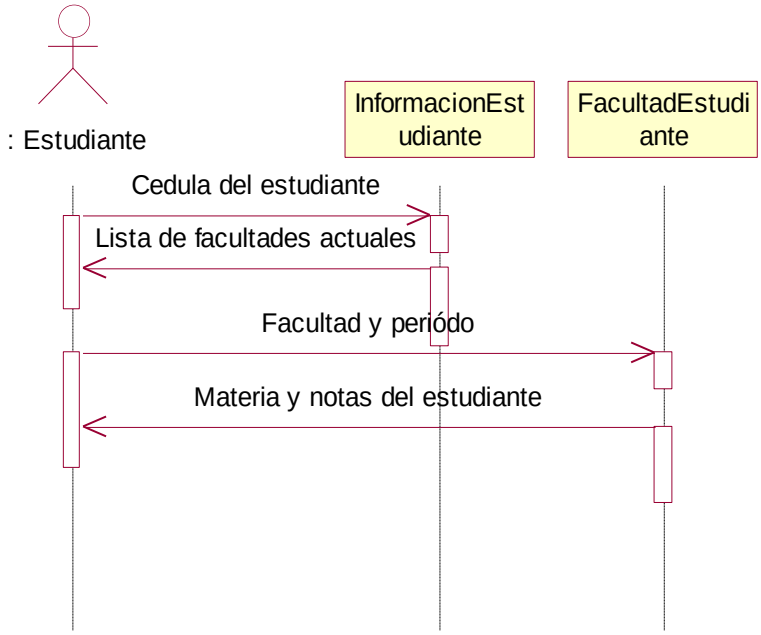


No existe record académico para ese alumno

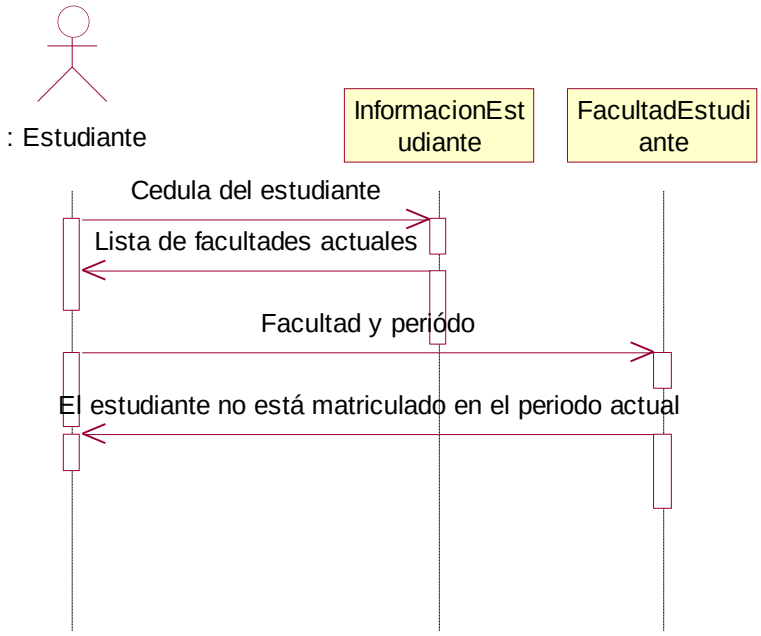


Consulta de Notas

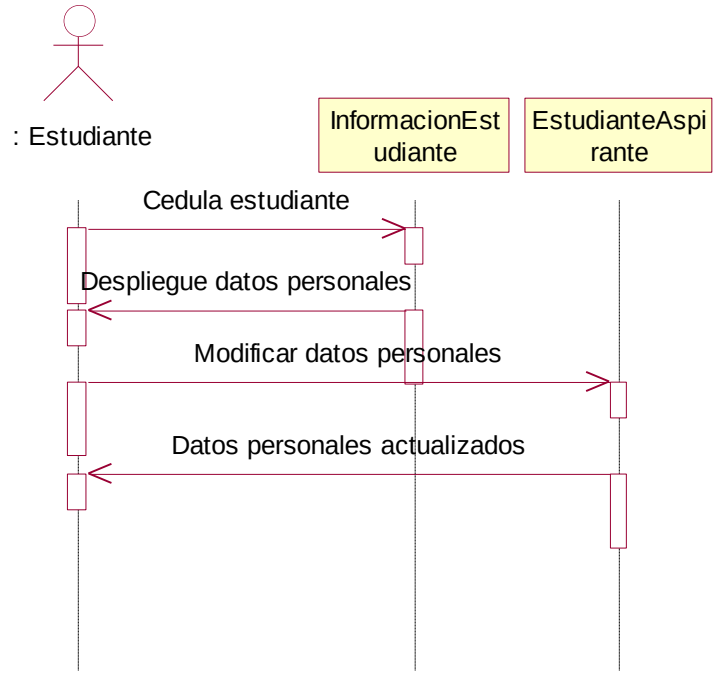
Correcta consulta de notas del estudiante



El estudiante no tiene notas en el periodo activo

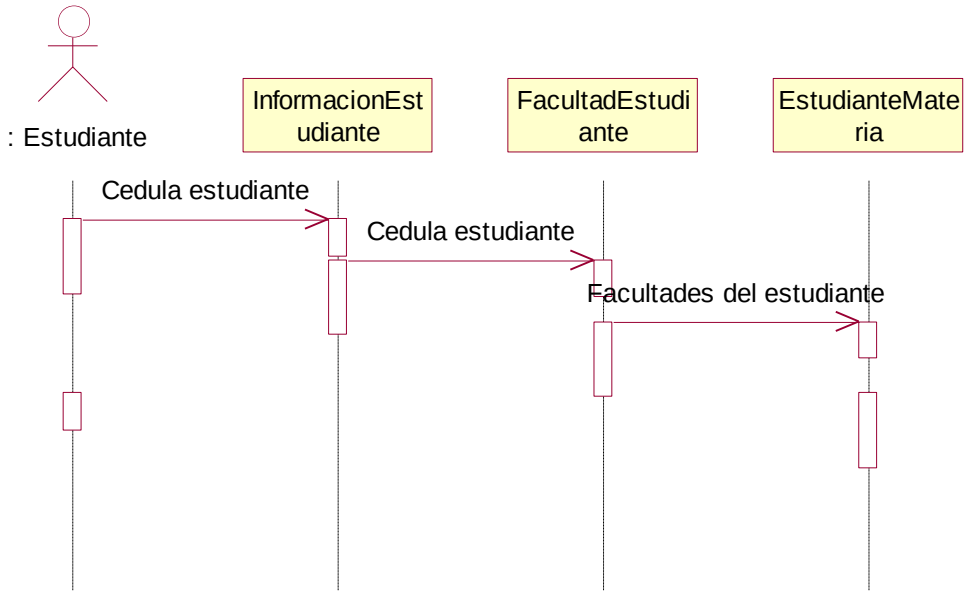


Consulta y/o Actualización de Datos Personales

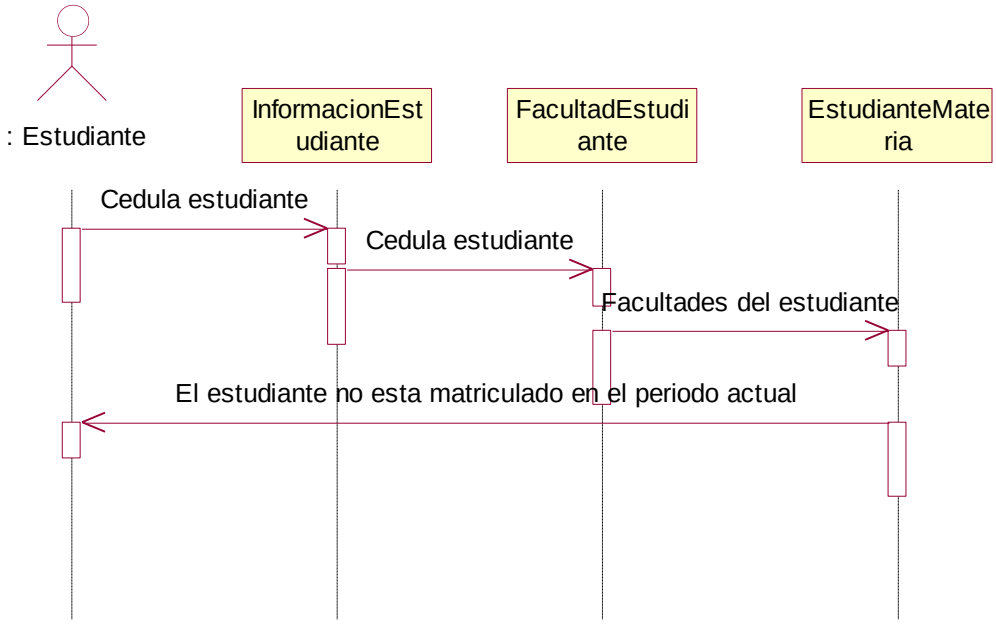


Consulta de Datos Académicos

Correcta consulta de los datos académicos del alumno



Consulta no realizada, estudiante no matriculado en periodo activo

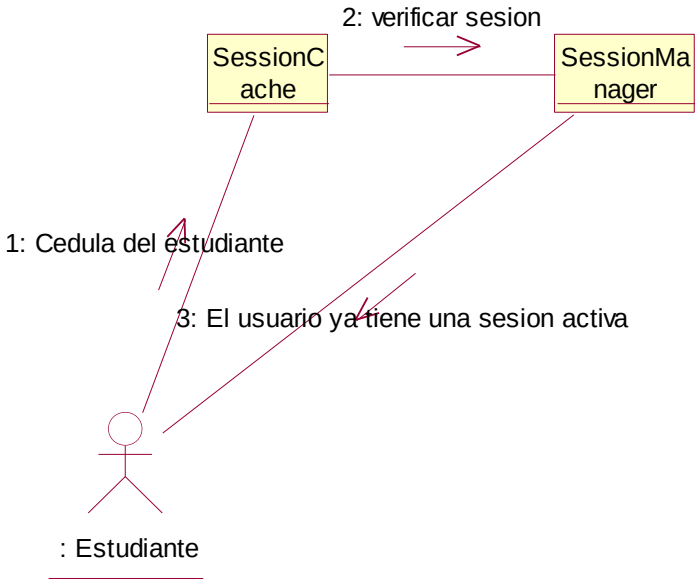


4.2.2 Diagramas de Colaboración

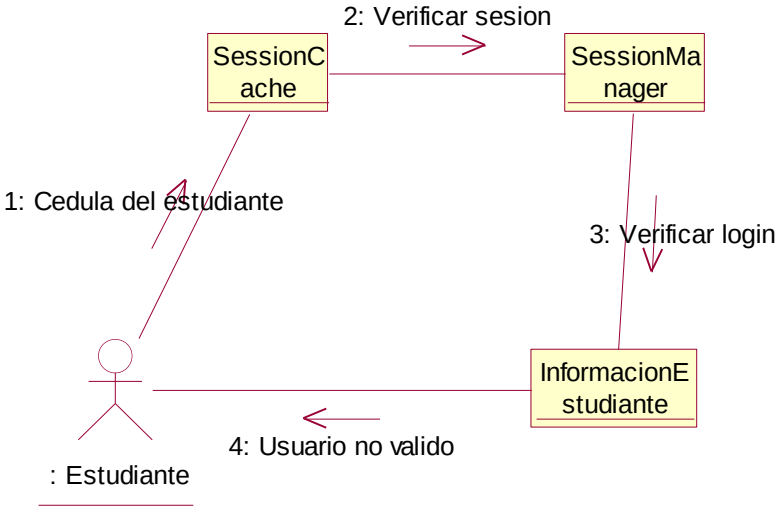
Control

a.Estudiantes

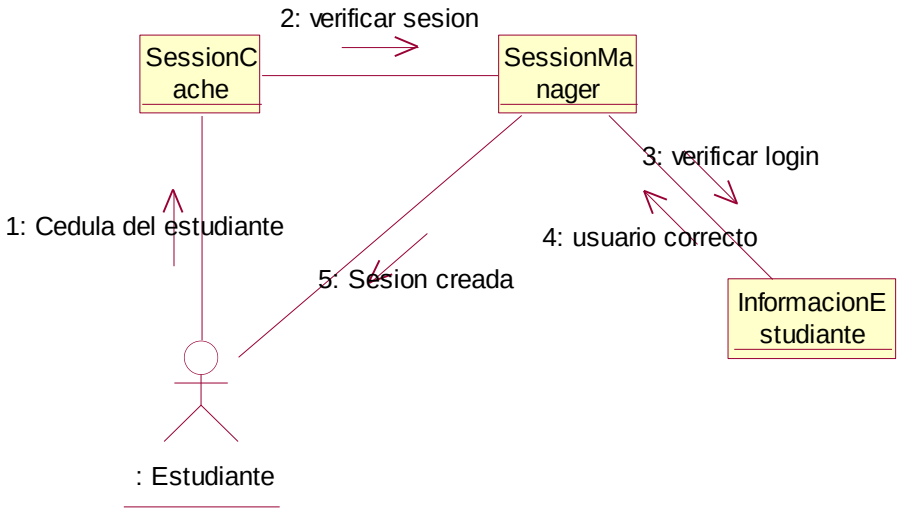
El usuario ya ha iniciado una sesión



Login incorrecto

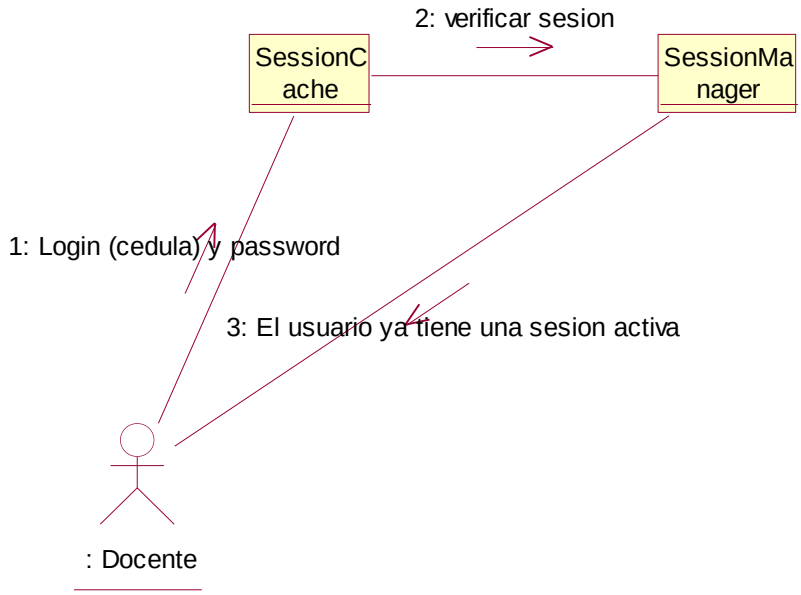

2: Verificar sesion
SessionCache
SessionManager
1: Cedula del estudiante
3: Verificar login
InformacionEstudiante
4: Usuario no valido
: Estudiante

Correcto inicio de sesión

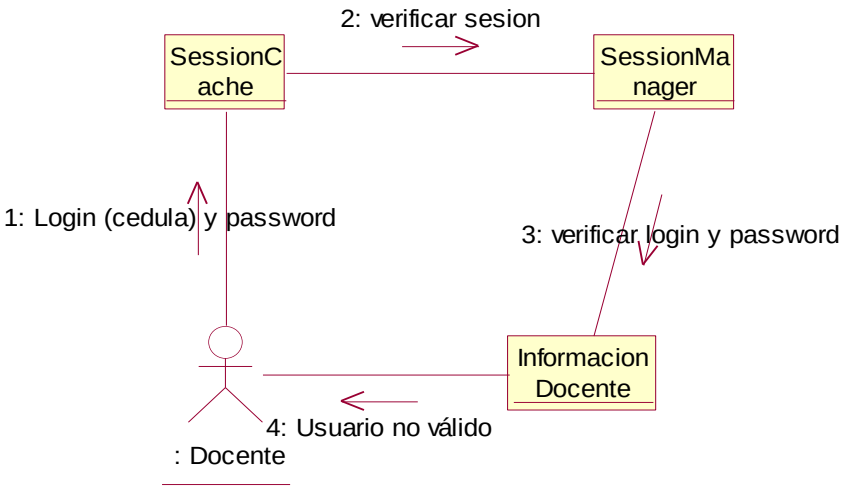


b.Docente

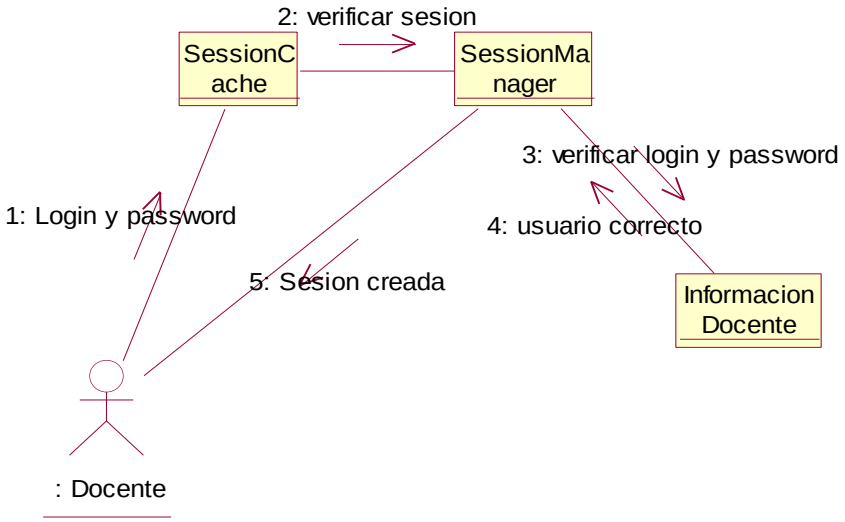
El usuario ya ha iniciado una sesión



Login incorrecto

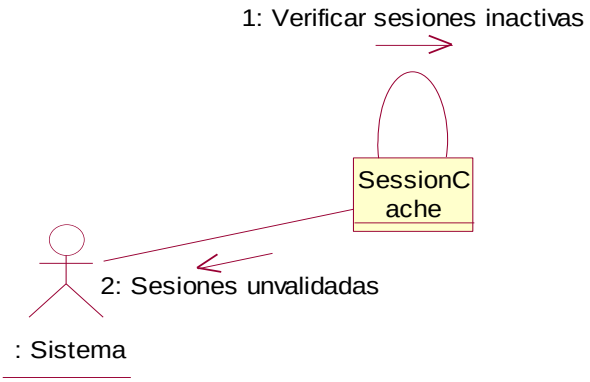


Correcto inicio de sesión



c.General

Validar sesiones inactivas

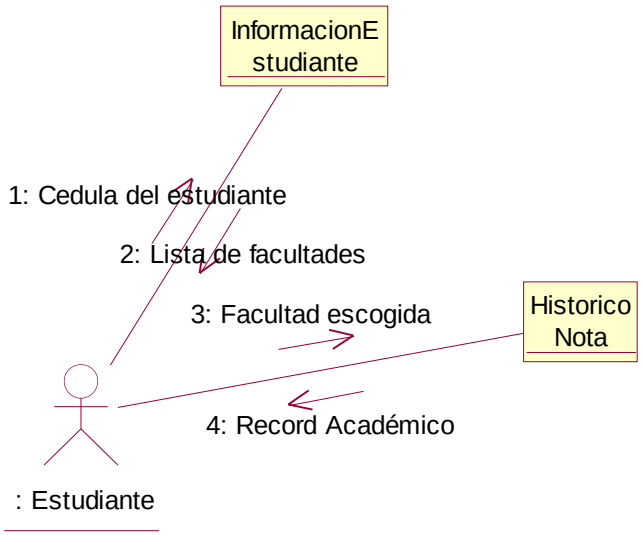


Servicios Virtuales

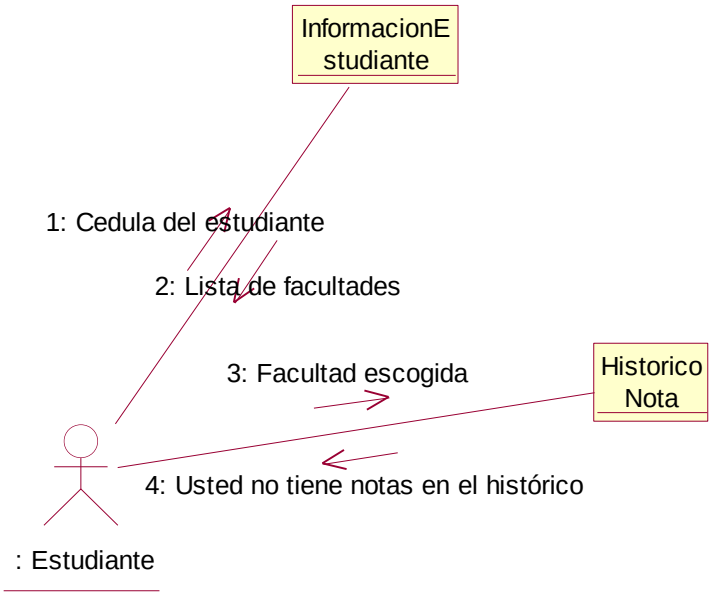
a.Estudiantes

Record académico

Correcta consulta del record

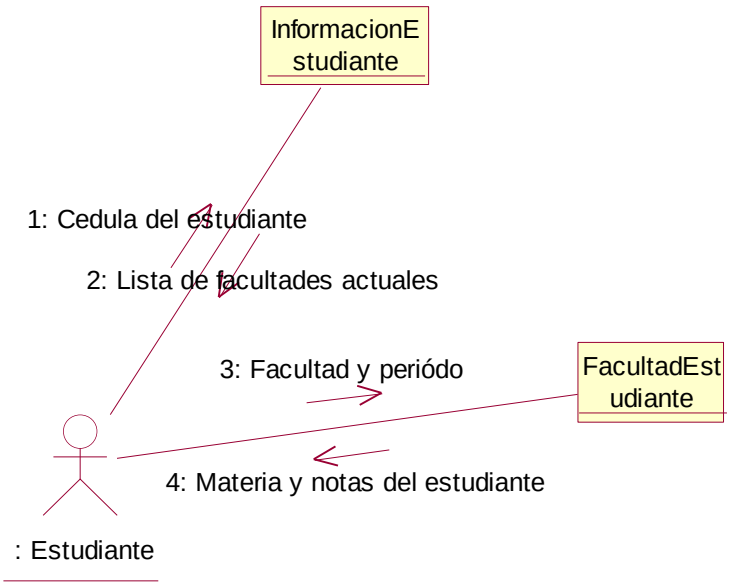


No existe record para ese alumno

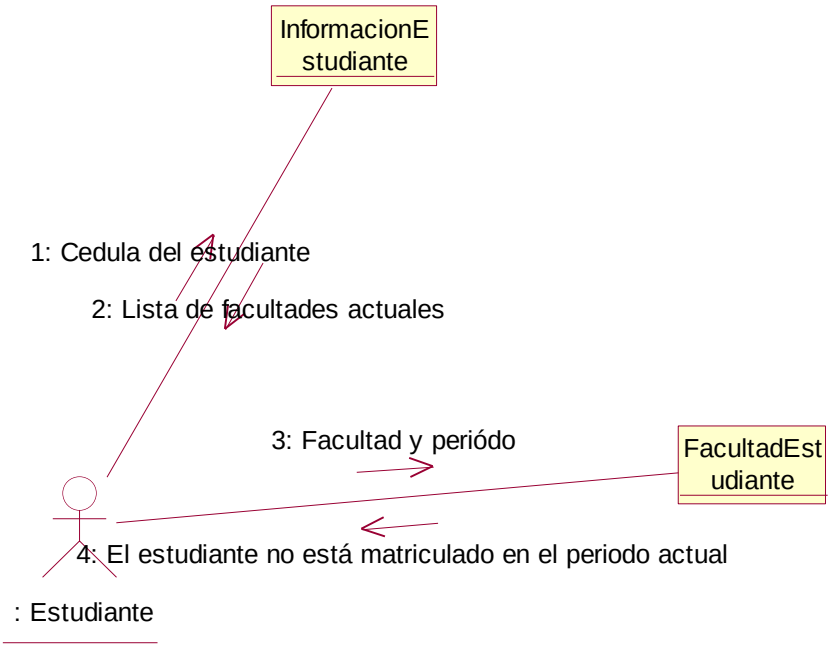


Consulta de notas

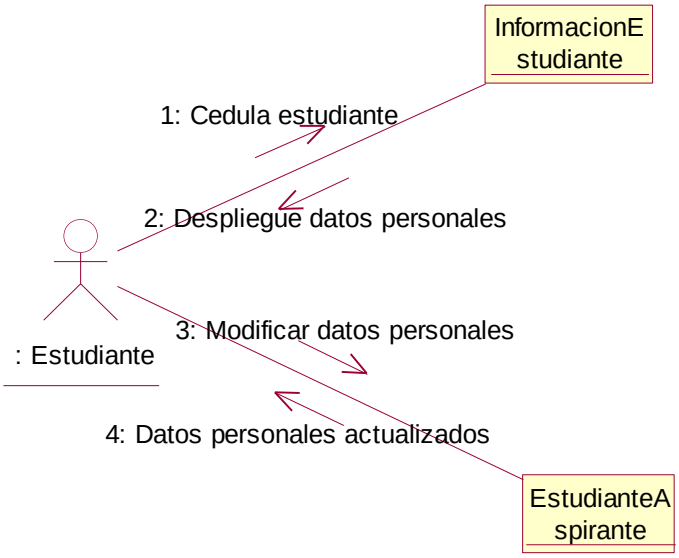
Correcta consulta de notas



No hay notas del estudiante en el periodo activo

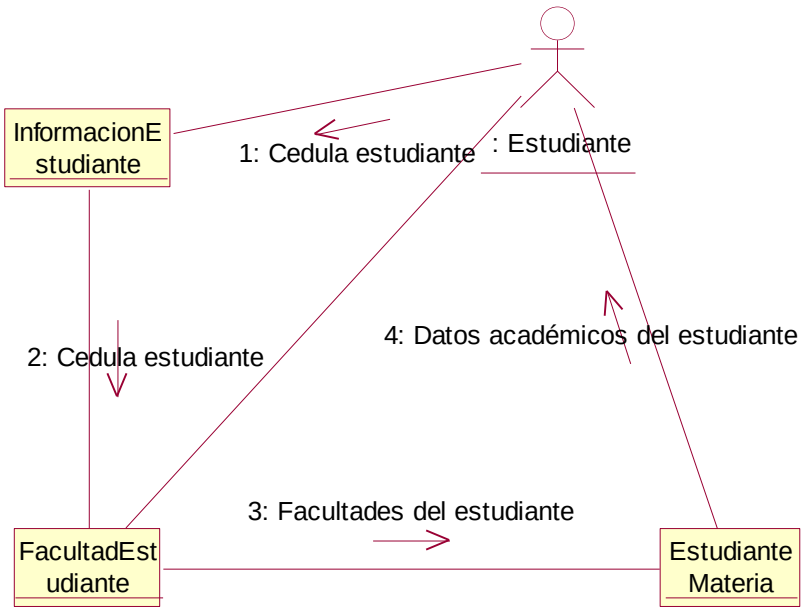


Consulta y/o Actualización de Datos Personales

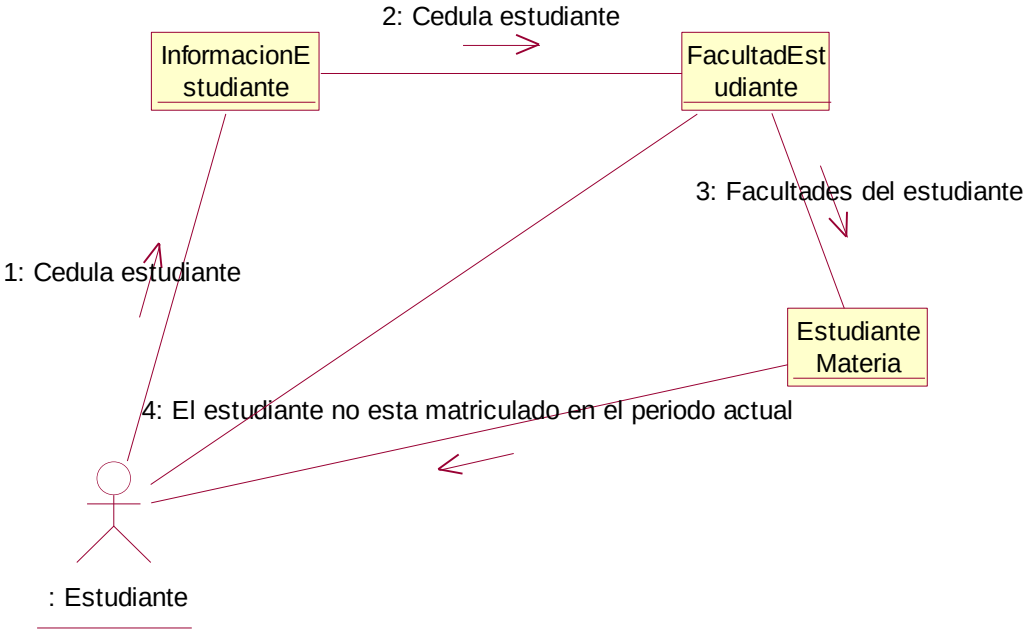


Consulta de Datos Académicos

Correcta consulta de los datos académicos del alumno



Consulta no realizada, estudiante no esta matriculado en periodo activo


2: Cedula estudiante
InformacionEstudiante
FacultadEstudiante
3: Facultades del estudiante
1: Cedula estudiante
Estudiante Materia
4: El estudiante no esta matriculado en el periodo actual
: Estudiante

V. IMPLEMENTACIÓN

5.1 Diagrama de Implementación

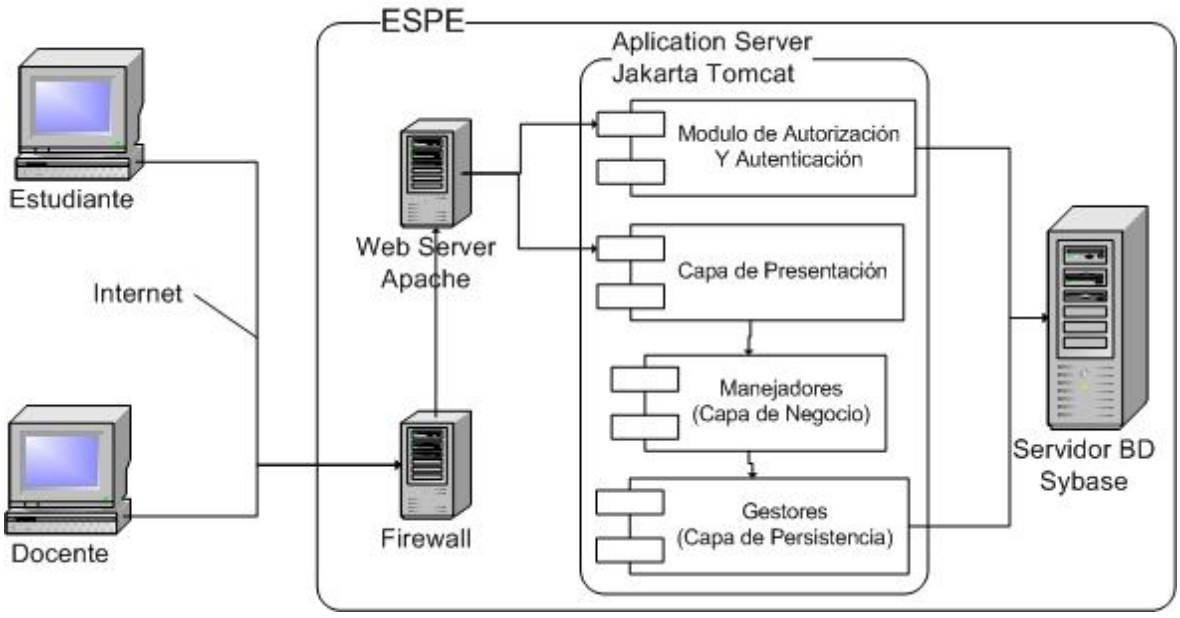


Figura 5.1 Diagrama de implementación de la aplicación

5.2 Pruebas de la Aplicación

La aplicación desarrollada en esta tesis, se entregó como grupos de funcionalidad y por lo tanto, una vez que una funcionalidad se encontraba lista, como por ejemplo consulta del Record Académico, se sincronizaban los cambios realizados e ingresaba a un período de pruebas internas en un servidor de desarrollo, sin afectar el uso del sistema en producción, en estas pruebas se comparan los resultados contra documentos físicos de estudiantes y docentes, y también se compara, en los casos que es posible, con los resultados del sistema Académico de PowerBuilder. Una vez aprobadas estas pruebas se lanzaba la

funcionalidad como piloto para cierto grupo de usuarios, periodo durante el cual se logró depurar la aplicación y actualizar la información.

Una vez finalizadas estas pruebas de funcionalidad e integración, se abre la funcionalidad para todos los usuarios de modalidad presencial.

En las pruebas se encontraron errores en la aplicación los cuales fueron solucionados inmediatamente, además de algunos cambios en funcionalidad.

5.3 Documentación Técnica

La documentación técnica de la aplicación se la adjunta en el CD, esta contiene todos los diagramas UML (de casos de uso del sistema, secuencia, colaboración, clases, etc.), documentación de cada una de las clases que conforman la aplicación en formato HTML (Javadocs), esto incluye las clases desarrolladas para la aplicación y las clases de terceros que también han sido utilizadas como el driver de conexión a la Base de Datos.

5.4 Manual de Configuración de la Aplicación

El manual de configuración de la Aplicación se lo adjunta en el CD, contiene la estructura de directorios que contiene el proyecto y la definición de los archivos de configuración del servidor de aplicaciones.

5.5 Manual de Usuario

El manual de usuario de la aplicación se lo adjunta en el CD, el manual de usuario contiene instrucciones de usuario para estudiantes y para docentes.

VI. CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- El proceso de registro y publicación de horarios, calificaciones, cartillas, récord académico y materias que toma el alumno en el período vigente ha sido implementado con éxito; con lo cual la ESPE ha logrado brindar a la comunidad universitaria una serie de servicios que facilitan sus labores tanto académicas como administrativas.

- Se ha conseguido integrar el sistema de modalidad a distancia con el de modalidad presencial con mediante la estandarización de un mismo módulo de Autorización y Autenticación. Se reutilizaron, modificaron y refactorizaron algunos componentes ya desarrollados para la modalidad a distancia, utilizando los mismos estándares de programación como modelo de persistencia y manejadores. Esto ha permitido la interacción e integración de maestros y alumnos entre los diferentes campus de la ESPE a nivel nacional.

- En el proceso de desarrollo de este sistema, se realizaron algunas actividades que se encontraban fuera del alcance de esta tesis, pero que eran esenciales para alcanzar los objetivos de la Dirección de Organización y Sistemas y la ESPE. Una de estas actividades fue implementar pools de conexiones para el acceso a diferentes esquemas de datos basados en el perfil de usuario que ingresa a la aplicación. Esto

significó la participación en el análisis, diseño y construcción del módulo de Autorización y Autenticación por lo que se debió omitir el desarrollo del modulo de manejo de archivos.

- Al utilizar una metodología Orientada a Objetos con ciclos de desarrollo Iterativos-Incrementales permitió tener en el análisis y diseño fases y herramientas muy claros, pero en la fase de implementación se utilizó el modelo de trabajo eXtreme Programing, para poder soportar el ritmo rápido que se requería en el desarrollo de cada uno de los módulos, y las modificaciones en requerimientos y definiciones, haciendo que el proyecto tenga un ciclo de desarrollo más corto y permitiendo alcanzar los objetivos establecidos.

- El sistema tiene un esquema de programación sencillo, claro y con responsabilidades bien definidas, además de estándares y reutilización de componentes, lo que permite que se puedan realizar modificaciones y / o actualizaciones de manera fácil y rápida, reduciendo el costo de mantenimiento.

6.2 Recomendaciones

- La ESPE debe seguir con su actual política de mejoramiento y optimización de procesos, incrementando los servicios virtuales y facilidades a todos los usuarios que requieran interactuar con la institución. Para esto, se recomienda continuar con la implementación y utilización de estándares de programación definidos por la Dirección de Organización y Sistemas.

- Se sugiere la implementación de un modelo MVC estándar, que puede ser Struts, el cual no solo dará todos los beneficios de una arquitectura que separa claramente sus componentes de presentación, sino que además facilitará el mantenimiento de las aplicaciones y la distribución del conocimiento, eliminando cualquier tipo de dependencia.

- El problema más grave y esencial encontrado durante la realización de esta tesis, fue implementar diferentes pools de conexiones para el acceso a diferentes esquemas de datos basados en el perfil de usuario que ingresa a la aplicación. Por lo cual se recomienda la utilización del módulo de Autorización y Autenticación, por todas las nuevas aplicaciones WEB desarrolladas para y por la ESPE.

- Se recomienda realizar una mejor documentación de los procesos y sistemas actualmente existentes en la ESPE, ya que eso facilitará la migración o el desarrollo de sistemas nuevos, acelerando el tiempo de

desarrollo y evitando realizar modificaciones a los sistemas por errores de definición.

- A futuro, se sugiere cambiar de versión de Base de Datos o de motor de Base de Datos en sí, puesto que se ha determinado que existen cierto tipo de limitantes con grandes cargas de usuarios que se reflejan como un mayor volumen de transacciones. De esta manera se podrá mejorar el rendimiento de todos los sistemas de la ESPE, ya sean estos sistemas WEB o cliente-servidor.

BIBLIOGRAFÍA

LIBROS:

Bernd Oestereich. (2000). Developing Software with UML Primera edición. Estados Unidos. Addison Wesley.

Martin Fowler. (2000). UML Distilled Segunda edición.

Estados Unidos. Addison Wesley.

Duffey Goyal. (2001). Professional JSP Site Design. Primera Edición.

Estados Unidos. Wrox

Avedal Kart (2001). Professional Java Server Programming J2EE Edition. Segunda Edición. Estados Unidos. Wrox

Berry Craig (2002) J2EE Design Patterns Applied. Primera Edición. Estados Unidos. Wrox

Fowler Martin (2003) Patterns of Enterprise Application Architecture. Primera Edición. Estados Unidos. Addison Wesley

WEB:

http://www.java.sun.com/products/jdbc/ *JDBC*

http://www.extremeprogramming.org *eXtreme Programming Method*

http://jakarta.apache.org/tomcat/index.html *Servidor de Aplicaciones Jakarta Tomcat*

http://www.sybase.com/ *Servidor de BD Sybase*

http://trevinca.ei.uvigo.es/~jcmoreno/is/omt.html *El desarrollo OO usando OMT (Object Modeling Technique)*

http://www-306.ibm.com/software/awdtools/rup/ *Racional Unified Preocess*

BIOGRAFÍA

David Alberto Meza Giraldo

Nacido el 9 de Junio de 1976, sus estudios secundarios los realizó en el Colegio Particular ¨Andino¨ en la ciudad de Quito, obteniendo el título de Bachiller en Físico-Matemático. Actualmente ocupa el cargo de Jefe de Proyectos en la empresa VimeWorks Ltda. dedicada a desarrollo, capacitación y asesoría en J2EE.