SISTEMA PARA LA PROGRAMACIÓN DE CURSOS OFRECIDOS POR CADA ESCUELA EN LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

JORGE ARMANDO MERCADO CASTILLA

LUZ VIVIANA JAIMES HERNÁNDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIA FISICO-MECANICA

ESCUELA INGENIERA DE SISTEMAS E INFORMATICA

BUCARAMANGA

2011

SISTEMA PARA LA PROGRAMACIÓN DE CURSOS OFRECIDOS POR CADA ESCUELA EN LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

JORGE ARMANDO MERCADO CASTILLA

LUZ VIVIANA JAIMES HERNÁNDEZ

PROYECTO DE GRADO PRESENTADO COMO REQUISITO PARA OPTAR POR EL TITULO DE INGENIERO DE SISTEMAS

DIRECTOR:

Msc. Luis Ignacio González

CODIRECTOR:

Msc. Jorge Villamizar Morales

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIA FISICO-MECANICA

ESCUELA INGENIERA DE SISTEMAS E INFORMATICA

BUCARAMANGA

2011

DEDICATORIA

A Dios que siempre ha estado cuidándome, apoyándome y estando ahí en todos los momentos de mi vida

A mis padres que me han apoyado en las decisiones que he tomado

A mi compañero de proyecto que estuvo hay como un apoyo para sacar adelante este proyecto

A todos aquellos que me han acompañado en mi proceso formativo y como persona.

Luz Viviana Jaimes Hernández

DEDICATORIA

Agradezco a Dios por la vida que me ha regalado, por la sabiduría,

Fortaleza y alegría que me han rodeado en cada momento de mi existencia.

A mis padres por su apoyo incondicional, por la confianza y el amor que me han bridado,

porque a ellos debo todo lo que tengo y todo lo que soy.

A mi hermano, que más que mi hermano es mi gran amigo y siempre ha estado apoyándome en las buenas y en las malas.

A todos aquellos amigos que siempre me han acompañado y de alguna manera me han apoyado y motivado para alcanzar este objetivo.

Jorge Armando Mercado Castilla

AGRADECIMIENTOS

A la Universidad Industrial de Santander por la formación integral a nivel Profesional y como seres humanos brindado a lo largo de nuestra carrera también por el apoyo que brindado para la realización de este proyecto.

Al ingeniero Robinson Delgado, por toda su colaboración y apoyo incondicional a lo largo del desarrollo de este mismo.

Al Ingeniero Danny Felipe Vergel, por el continuo apoyo y las enseñanzas aportadas durante la realización del proyecto.

A la División Servicios de Información, por todo el apoyo y su colaboración intelectual, para el desarrollo de este proyecto.

A todos nuestros amigos que con su ánimo, apoyo y contribuciones personales y profesionales hicieron que este proyecto llegara a su meta.

Y a todas aquellas personas que de una u otra forma colaboraron o participaron en la culminación de esta meta hoy alcanzada.

TABLA DE CONTENIDO

LISTADO DE FIGURAS

INTRODUCCIÓN

Hoy en día los sistemas juegan un papel importante en muchos ámbitos de la vida de los seres humanos, permitiendo que los procesos sean realizados en una cantidad menor de tiempo y a su vez permitiéndole a las organizaciones que cuenten con información veraz.

La Universidad Industrial de Santander se encuentra en una búsqueda continua de mejora de sus procesos un ejemplo de ello es la obtención de la certificación ISO 9001, por lo tanto actualmente la División de Servicios de Información esta encarga de varios proyectos de grado encaminados a la mejora de sus actuales sistemas para que de esta manera los usuarios de los mismos tengan una interfaz más amigable y hagan de estos una herramienta imprescindible en sus respectivas labores.

Las Escuelas de la Universidad Industrial de Santander, actualmente realizan su programación de horarios para el semestre de forma manual lo cual hace que sea necesario en el uso de muchos formatos para poder tener la información a la mano, también para este proceso es importante el conocimiento por parte del Director de Escuela.

Por las razones antes expuestas se crea por primera vez una herramienta Web que facilite la creación de horarios que realizan los Directores de Escuela, esta solución fue creada con el apoyo de la División de Servicios de información.

Como resultado de este proyecto se diseñó y desarrollo un módulo que permite crear los horarios para un semestre de una forma intuitiva y agradable. Dicho módulo se hizo con los estándares de programación definidos en la División de Servicios de Información.

Este documento contiene el soporte teórico, metodológico y técnico del desarrollo Web del módulo de Horarios para las carreras del Campus principal de la Universidad Industrial de Santander.

LISTADO DE TABLAS

GLOSARIO TÉCNICO

API: es una interfaz de programación de aplicaciones, API (del inglés application programming interface) es un conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ASIGNATURA: es un componente del pensum que pertenece a una carrera, un nivel. Además posee una intensidad horaria, un contenido, unos objetivos y para su identificación tiene un código a nivel institucional.

BASE DE DATOS: es una serie de datos organizados y relacionados entre sí, los cuales son obtenidos y utilizados por los sistemas de información de una empresa o negocio en particular, en la actualidad la mayoría de las bases de datos se encuentran en formato digital.

CÓDIGO FUENTE: son líneas de texto escritas en un lenguaje de programación especifico que representan instrucciones que el computador llevara a cabo al transformarse a otro lenguaje (código objeto o el lenguaje de máquina).

COMPILACIÓN: es el proceso de traducción de un código fuente a lenguaje de máquina para que pueda ser ejecutado por la computadora.

CONTRASEÑA: es un conjunto finito de caracteres que conforman una palabra secreta que permite que un usuario o más accedan a unos determinados recursos. Las claves no aceptan todo tipo de caracteres, también existe limitación en su longitud.

CURSO: es la instancia de una asignatura que tiene un profesor a cargo y un horario. Dentro de la universidad es distinguido con una letra y un número.

ASIGNATURA MÚLTIPLE PROFESOR: es una asignatura que permite varios profesores en una misma aula a una misma hora.

ASIGNATURA MAGISTRAL: son cursos de una misma asignatura que son dictados por el mismo profesor en una misma aula y a la misma hora.

DBA: (Administrador de Base de Datos). Es la persona encargada del control general de la Base de Datos.

DBMS: (Data Base Management System). Sistemas de Gestión de Bases de Datos. Son una clase de software cuya función es servir de interfaz entre el usuario, la base de datos y las aplicaciones que utiliza.

EJB: los Enterprise JavaBeans son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J5EE de Oracle Corporation.

FRAMEWORK: se refiere a "ambiente de trabajo". Es un conjunto estandarizado de conceptos, prácticas y criterios para tratar una problemática en particular y luego se toman de base para resolver problemas de índole parecido.

HOME PAGE: (Start page, front page, main web). Página de inicio o principal de un sitio web. Suele tener el nombre: index.html, aunque también se puede usar index.php, index.asp, etc.

INFORMIX: es un gestor de base de datos creado por Informix software Inc. Incluye un RDBMS (sistema administrador de base de datos relacionales) basado en SQL.

INTERFAZ: es un medio mediante el cual el usuario puede comunicarse con el computador, para enviarle ordenes que son traducidas en respuestas por parte de la máquina.

INTERNET: es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

JAVA: es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. Este lenguaje toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen conducir a muchos errores, como la manipulación directa de punteros o memoria.

JAVA EE: (Java Platform Enterprise Edition). Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuidos, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

JBOSS DEVELOPER STUDIO (JBDS): es un entorno de desarrollo comercial con código abierto, creado y desarrollado por JBoss (una división de Red Hat) y Exadel, Inc.

JBOSS EAP: (JBoss Enterprise Application Platform). Es la plataforma líder en el mercado para las aplicaciones Java, es innovadora y escalable. Integrada, simplificada y entregada por el líder en software empresarial de open source (código abierto), incluye tecnologías open source líderes para la construcción, implementación y hosting de servicios y aplicaciones empresariales Java.

JBOSS HIBERNATE: hibernate es un potente servicio de mapeo objeto/relacional de alto desempeño para consulta y persistencia. Que permite desarrollar clases persistentes siguiendo el lenguaje orientado a objetos, incluyendo la asociación, herencia, polimorfismo, composición y colecciones.

JBOSS SEAM: es un framework que integra y unifica los distintos estándares de la plataforma Java EE 5.0, permitiendo trabajar con todos ellos siguiendo el mismo modelo de programación. Ha sido diseñado intentando simplificar al máximo el desarrollo de aplicaciones, basando el diseño en Plain Old Java Objects (POJOs) con anotaciones. Estos componentes se usan desde la capa de persistencia hasta la de presentación, poniendo todas las capas en comunicación directa.

JDK: (Java Development Kit). Es un grupo de herramientas para el desarrollo de software realizado por Sun Microsystems Inc. Incluye las herramientas necesarias para escribir, testear, y depurar aplicaciones y applets de Java.

JPA: (Java Persistence API). Es un lenguaje de programación Java que permite a los desarrolladores gestionar datos relacionales en aplicaciones que usan Java Platform Standard Edition (Java SE) y Java Platform Enterprise Edition (Java EE).

JPQL: (Java Persistence Query Language). Es un lenguaje de consulta orientado a objetos. JPQL se utiliza para hacer consultas a las entidades almacenadas en una base de datos relacional. Está fuertemente inspirado en SQL y sus preguntas se asemejan a las consultas SQL en la sintaxis, pero operan contra los objetos entidad JPA y no directamente con las tablas de base de datos.

LOGIN: es el proceso de verificar la identidad digital del remitente de una comunicación como una petición para conectarse. El remitente siendo autenticado puede ser una persona que usa un ordenador, un ordenador por sí mismo o un programa del ordenador.

MAPEO OBJETO RELACIONAL: (Object-Relational Mapping - ORM). Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la realidad esto crea una base de datos orientada a objetos virtual, sobre la base de datos

relacional, lo cual posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

PÁGINA: documento o archivo codificado en un lenguaje de marcado, para nuestro caso XHTML, que suele contener textos con hipervínculos, imágenes, sonidos, etc., que en la mayoría de los casos se alojan en un sitio o servidor web, y pueden ser consultada empleando un navegador web.

PERSISTENCIA: se refiere a la propiedad de los datos para que estos sobrevivan de forma permanente, pero también se trata de poder recuperar la información del objeto para que sea nuevamente utilizado. De forma sencilla puede entenderse que los datos tienen una duración efímera, desde el momento en que estos cambian de valor se considera que no hay persistencia de los mismos.

REGISTRO: es un tipo de dato estructurado formado por la unión de varios elementos bajo una misma estructura. Estos elementos pueden ser, o bien datos elementales (entero, flotante, real, carácter, etc.), o bien otras estructuras de datos. A cada uno de esos elementos se le llama campo.

MÉTODO: es una operación que define como se comporta un objeto.

SALONES ESPECIALES: son aquellas aulas que presentan características particulares, por ejemplo: laboratorios, salas de cómputo, salones con capacidad diferente a la estándar o con recursos adicionales (videobeam, etc.).

SALONES REGULARES: son aquellas aulas normales que presentan características similares a otras.

PROFESOR PLANTA: es aquel profesor que tiene un contrato a término indefinido con la universidad y casi siempre debe tener carga académica en caso que no se encuentre en comisión o año sabático entre otros.

PROFESOR CATÉDRA: es un profesor que aparte de trabajar en la universidad puede trabajar en otra institución, tiene un contrato con vigencia por el semestre y le cancelan su sueldo por las horas dictadas.

PROFESOR BECARIO: es un profesor que en el semestre en el cual se va a realizar programación de horarios se encuentra matriculado en una maestría o posgrado y esta beneficiado por una beca por parte de la universidad.

UML: es un lenguaje de modelado de sistemas de software, UML (en Inglés significa Unified Modeling Language) está respaldado por OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

XHTML: acrónico en ingles de e**X**tensible **H**yper**T**ext **M**arkup Language (Lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir el HTML como estándar para páginas web.

CAPÍTULO 1

1 CONTEXTO GENERAL

1.1 ESPECIFICACIONES DEL PROYECTO

1.1.1 Título proyecto SISTEMA PARA LA PROGRAMACIÓN DE CURSOS OFRECIDOS POR CADA ESCUELA EN LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

Director del proyecto:

NOMBRE: Ing. Luis Ignacio González

INSTITUCIÓN: Universidad Industrial de Santander

CARGO: Profesor adscrito a la Escuela de Ingeniería de Sistemas.

Codirector del proyecto:

NOMBRE: Ing. Jorge Villamizar Morales

INSTITUCIÓN: Universidad Industrial de Santander

CARGO: Profesor adscrito a la Escuela de Matemáticas.

Autores:

Nombre: Jorge Armando Mercado Castilla

Código: 2031540

Carrera: Ingeniería de Sistemas e Informática

Nombre: Luz Viviana Jaimes Hernández

Código: 2042565

Carrera: Ingeniería de Sistemas e Informática

Dependencias interesadas en el proyecto:

Escuelas

Dirección de Admisiones y Registro Académico

División de Servicios de Información

1.1.2 Objetivos

1.1.2.1 Objetivo General Desarrollar e implementar un sistema software orientado a la Web que facilite la programación de las asignaturas que semestralmente son ofrecidas por las Escuelas, a partir de la disponibilidad de aulas, profesores, asignaturas, franjas horarias y el plan de estudios.

1.1.2.2 ***Objetivos Específicos*** Desarrollar un sistema software orientado a la Web, el cual deberá presentar las siguientes características:

- Ser una herramienta que permita la creación de la programación para el semestre que recién inicia a partir de la programación de semestres anteriores.

- Dispone de una interfaz amigable e intuitiva que permita realizar modificaciones de manera sencilla. A su vez efectúa operaciones sobre algunas entidades del sistema tales como:

Con los profesores: agregar, modificar, eliminar docentes de un curso.

Con los cursos: agregar, modificar y eliminar.

- Permite ingresar la disponibilidad de los docentes, las materias que cada docente puede dictar y los horarios en los cuales puede dictarlas.

- Realiza consultas en donde se muestre la programación para asignaturas, aulas y docentes.

- Verificar que no existan cruces entre materias de un mismo nivel para determinado plan de estudios ofrecido por una escuela, y entre docentes en una misma franja horaria. En caso de existir cruces mostrará advertencias de la ocurrencia de estos.

- Genera informes tales como: programación semestral de asignaturas para una escuela, horarios por asignatura, horarios por profesor, horarios por nivel, cruces entre materias del mismo nivel.

- Estadísticas de utilización de las aulas de clase para cada escuela y para el campus universitario a manera de guía para la creación de cursos.

- Realizar el diseño del sistema mediante el estándar de lenguaje de modelado unificado (UML) con el fin de identificar y comprender plenamente el sistema, lo que permitirá realizar futuras modificaciones de manera ágil.

- Integrar el sistema desarrollado junto con los servicios que ofrece la División de Servicios de Información para que esté a disposición de todas las escuelas del campus principal.

- Incorporar ayudas y capacitar a algunos Directores de Escuela en el manejo de la herramienta, para el desarrollo de pruebas y validación de requerimientos.

1.1.3 Definición del problema Sin duda alguna uno de los elementos más importantes para el buen desarrollo y éxito de las organizaciones es la Gestión de la Información, puesto que permite tomar decisiones oportunas y adecuadas. El buen manejo de la información se convierte en una herramienta que suministra los recursos necesarios para realizar procesos de manera óptima y así brindar productos y servicios de calidad.

La Universidad Industrial de Santander no es ajena a esta situación y dentro de las acciones encaminadas a brindar un mejor servicio a la comunidad universitaria, ha venido apoyando el mejoramiento de sus procesos administrativos. De ahí que ha implementado sistemas de calidad y acreditación para la validación de los procesos que se realizan en el interior de la Institución. Sin embargo, existen todavía algunos de ellos relacionados con el área de la academia, que se realizan de una manera no tan óptima.

Tal es el caso de la programación semestral de cursos que realizan los directores de cada una de las escuelas, un ejemplo de ello es la Escuela de Matemáticas que ofrece una gran cantidad de materias de servicio, requiriendo gran cantidad de profesores cátedra y recursos superiores a los que tiene la escuela para satisfacer una demanda aproximada de 6800 estudiantes y 180 grupos aproximadamente.

Para entender un poco más el proceso que se lleva a cabo en la Universidad Industrial de Santander, se hará una breve explicación del mismo, pero antes de esto es necesario mencionar algunos detalles importantes a tener en cuenta a la hora de realizar la programación, por ejemplo: los Directores de Escuela deben tener presente que existen diferentes tipos de asignatura como lo son las propias de cada carrera ofrecidas en su Escuela y las asignaturas de servicio que son aquellas ofrecidas a otras carreras como el caso de la Escuela de Matemáticas en donde la mayor parte de cursos que ofrecen son de este tipo.

También existen requerimientos especiales de algunos docentes en cuanto a condiciones especiales para que ellos dicten su clase, por ejemplo, algunos no pueden dictar clases en salones que estén ubicados en pisos superiores, otros prefieren usar tablero verde y tiza en lugar de tableros acrílicos. A su vez, existen 3 tipos de docente: planta, cátedra y

becarios, inicialmente se asignan horarios a los docentes planta y se da prioridad a éstos en la asignación de franjas horarias posteriormente se asignan los docentes becarios y cátedra.

El proceso se puede describir en 5 etapas:

1. Elaborar horario por cada Escuela
2. Estudiante realiza matrícula
3. Proceso Oferta-Demanda
4. Matricula de estudiantes en cursos
5. Asignar aulas a cursos

1. Elaborar horario por cada Escuela

En esta etapa los Directores de Escuela construyen el horario para las asignaturas que se ofrecerán el próximo semestre, éste proceso se realiza aproximadamente 4 semanas antes de finalizar el semestre en curso.

Para la elaboración de éste horario se realiza un proceso dispendioso en donde debe estar disponible el horario del semestre anterior que es una cantidad significativa de hojas distribuidas a lo largo de escritorios para poder apreciar en su totalidad las asignaciones realizadas, así como resaltadores, borrador y lápiz para ir modificando el horario de acuerdo a la nueva programación.

Los Directores de Escuela también deben contar con la disponibilidad y capacidad de las aulas de clase, los formatos de disponibilidad de cada uno de los Docentes y las materias que éstos están en capacidad de dictar el próximo semestre.

Se debe tener en cuenta que algunos estudiantes no se matriculan en algunas asignaturas vía web y esperan el proceso de ajuste de matrícula para incluirlas, dato que deben tener en cuenta los Directores de Escuela al momento de ofrecer los cursos semestrales.

En esta parte del proceso el Director de Escuela asigna aulas especiales y aulas propias de la escuela, sin embargo, algunos cursos quedan sin aula, lo que genera retrasos pues más adelante es necesario buscar aulas disponibles en otras Escuelas.

Los Directores de Escuela también deben tener presente en todo momento las restricciones (tipos de tablero, ubicación de las aulas, etc.) de algunos docentes para la asignación de aulas, así como evitar que existan cruces en los horarios de los docentes, evitar cruces entre asignaturas de un mismo nivel dado que el sistema actual de la universidad solo detecta cruces entre salones.

Como resultado, se obtiene la oferta de cursos para el próximo semestre que cada Escuela presentará a los estudiantes. Ésta programación contiene información como: nombre y capacidad del curso, código de la asignatura, docente asignado, salón establecido, si tiene reserva, es decir, que solo pueden incluirse en ese curso alumnos de unas carreras en particular y en determinada franja horaria.

Todo el proceso mencionado es realizado de forma manual para luego digitarlo en el sistema principal de la Universidad.

2. Estudiante realiza matricula

Los estudiantes ingresan vía web al sistema académico de la Universidad y pueden observar las asignaturas que pueden cursar durante el semestre a matricular de acuerdo a su historial académico; seleccionan las que desean ver y como resultado de este proceso se obtiene la demanda real de cursos para el semestre que inicia.

3. Proceso de Oferta-Demanda

Se realiza un contraste entre la oferta de cursos por parte de cada Escuela y la demanda real que se obtuvo en la etapa anterior. Estos datos son suministrados por el Dirección de Admisiones y Registro Académico a cada una de las Escuelas en un documento que muestra las asignaturas programadas discriminando la cantidad de estudiantes según la prioridad escogida para éstas y la cantidad de cursos programados para cada asignatura, éste documento también muestra la diferencia entre lo ofrecido y lo demandado permitiendo que los Directores de Escuela realicen los respectivos ajustes, ya sea crear cursos, modificar su capacidad o eliminar éstos.

En esta etapa el Director continúa asignando aulas y docentes que faltaron o aulas para nuevos cursos que surgieron debido al ajuste de oferta – demanda. En caso de que existan cursos a los cuales no se les ha asignado un docente (por lo general Docentes cátedra) se utiliza un comodín para todos esos casos el cual corresponde a colocar al Director de Escuela como Docente temporal. Es pertinente mencionar que muchas veces los Directores de Escuela crean cursos según la demanda real desconociendo si existen aulas disponibles para éstos.

4. Matricula de estudiantes en cursos

Luego de hacer las modificaciones necesarias en la programación de cursos planteada por cada escuela, se procede a asignar los estudiantes en cada uno de los cursos, este proceso es realizado por la División de Servicios de Información a través del sistema de la universidad y de acuerdo a criterios establecidos por la universidad como incluir primero a estudiantes nivelados o estudiantes de acuerdo a la prioridad en las materias de niveles inferiores.

5. Asignar aulas a cursos

Para ésta etapa los directores de escuela han realizado en lo posible la asignación de aulas y docentes de acuerdo a información de semestres anteriores y de acuerdo al proceso de oferta-demanda, sin embargo, se presentan situaciones particulares en algunas escuelas, por ejemplo, en la escuela de Ingeniería de Sistemas casi siempre se realiza la asignación de aulas con éxito y sobran algunas para utilización de otras escuelas mientras que en la escuela de Matemáticas ocurre lo contrario, debido a la gran cantidad de cursos que se ofrecen siempre quedan muchos cursos sin aula asignada.

Los directores de escuela realizaron su asignación ubicando primero las aulas especiales (laboratorios, etc.) y aquellas destinadas para materias propias de su escuela, ésta información debe ser ingresada actualmente a través del sistema de la universidad.

En este proceso existe colaboración entre las escuelas para la asignación de aulas a cursos que todavía no tienen aún una asignada, por ejemplo, la escuela de Ingeniería de Sistemas informa a la escuela de Matemáticas sobre la disponibilidad de aulas para su aprovechamiento, sabiendo esto, la escuela de Matemáticas realiza la solicitud a admisiones para que sean asignadas esas aulas disponibles a los cursos de su escuela que aún no poseen aula.

Con esta información de aulas disponibles Dirección de Admisiones y Registro Académico termina de asignar éstas a todos aquellos cursos en el campus universitario que aún no tienen aula programada.

Los procesos correspondientes a los numerales 2, 3 y 4 son realizados a través del sistema actual de la universidad, los procesos del numeral 1 y numeral 5 (Elaborar horario por cada Escuela y asignar aulas a cursos respectivamente), se realizan de manera manual convirtiéndose en una situación desgastante, dispendiosa y que involucra en algunos casos algo más de cuatro semanas. Por la misma razón genera por momentos problemas e inconsistencias en la información que se suministra a las partes interesadas: por ejemplo cruces entre algunas materias de un mismo nivel o cruces en el horario de un docente.

Estos aspectos agravan la calidad del servicio ofrecido, generando retrasos y gastos de recursos que podrían evitarse si se realiza un manejo óptimo de la información con la que se cuenta.

1.1.4 Justificación La Universidad Industrial de Santander necesita que día a día sus procesos sean sistematizados para que estos sean más eficaces y eficientes, todo esto en busca de la calidad institucional. Al ser la universidad pionera en ciencia y tecnología, apoya proyectos de grado encaminados en lograr procesos más competentes, puesto que cuenta con los conocimientos y los recursos apropiados.

El desarrollo de la programación semestral de asignaturas está a cargo de los directores de escuela y para esto deben tener a la mano información de los horarios del semestre anterior, que sirven como base para la realización del horario actual, el plan de estudios de la respectiva carrera por niveles; además debe contar con las asignaturas que un profesor puede dictar y su disponibilidad horaria para dictar dichas asignaturas, las cuales se pueden encontrar en distintos niveles; esta información es recibida de los profesores por medio de papeles. Para esta labor el director cuenta con una matriz, lápiz y papel, haciendo posible que se cometan errores humanos y más adelante inconformidad por parte de los estudiantes.

En la actualidad los directores de escuela tienen gran dificultad en el desarrollo de ésta programación en sus respectivas escuelas, pues ésta sigue siendo manual y ello implica una cantidad significativa de tiempo semestre tras semestre, por tanto se hace necesario el desarrollo de una solución que facilite la construcción del módulo de horarios, entendido dicho módulo como la asignación de profesores, franjas horarias y aulas de clase a los distintos cursos ofrecidos por una Escuela.

1.1.5 Viabilidad El desarrollo de este proyecto posee el aval de la Universidad Industrial de Santander y se convierte en un proyecto institucional de una repercusión significativa para la misma.

Es por esto que se cuenta con el total apoyo de esta institución a través de la División de Servicios de información y las respectivas dependencias académicas a las cuales les compete está temática.

La universidad ofrece las instalaciones y los recursos necesarios para el desarrollo del proyecto, contando con el soporte técnico de la División de Servicios de Información y las herramientas software y hardware que garantizan un adecuado desarrollo del mismo.

CAPÍTULO 2

2.1 MARCO TEÓRICO

2.1.1. Estado del arte Actualmente en la Universidad Industrial de Santander existe un sistema el cual permite ingresar los cursos que se van a programar para el siguiente semestre, este sistema detecta los cruces que pueden existir con una misma aula, si un docente cátedra tiene contrato vigente, pero no detecta los cruces que se puedan presentar con los horarios de un docente , a su vez tampoco detecta los cruces que puedan presentarse entre las asignaturas pertenecientes a plan de estudios especifico de un programa académico permitiendo que por errores humanos sean programadas asignaturas de un mismo nivel de un plan de estudio en horarios que se cruzan, y como consecuencia de esto cuando corre la matricula académica es decir matricular a los estudiantes en los respectivas asignaturas que ellos solicitaron no puedan tener una matrícula al cien por ciento (100%) es decir exitosa y por lo tanto los estudiantes tienen que en el momento del ajuste de matricula plantear una solución acorde a sus necesidades.

Según las políticas actuales planteadas por la Universidad no es permitida la copia de los horarios del semestre anterior ocasionando que los Directores de Escuela o a su vez las secretarias de las mismas tengan que ingresar al sistema actual de la Universidad todos los datos de los cursos que se van a ofrecer para el semestre a programar, aparte de lo anteriormente mencionado este proceso de programación de cursos es realizado en papel antes de ser digitado, también cabe resaltar que el sistema actual no posee una interfaz web.

1.2 ESQUEMA DE DESARROLLO

2.2.1. Diagramación UML (Unified Modeling Language) es una consolidación de varias notaciones y conceptos más utilizados orientados a objetos. Comenzó como una consolidación del trabajo realizado por Grady Booch, James Rumbaugh, e Ivar Jacobson,

los cuales fueron los creadores de tres de las metodologías orientadas a objetos más populares; este lenguaje está respaldado por el OMG(Object Managament Group).

El Lenguaje Unificado de Modelado permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos a su vez determina una serie de notaciones y diagramas estándar. En la actualidad los modeladores sólo necesitan aprender una única notación. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Esta notación ha sido ampliamente aceptada gracias al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos en los que se basa (principalmente Booch, OOSE y OMT).

UML no puede compararse con la programación estructurada, puesto que UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

Además este lenguaje de modelado no sólo fue desarrollado para entender mejor los requerimientos del Sistema (o del Software) sino también para contar con un prototipo orientado a objetos; vital en el desarrollo de Software dado que se identifica con el paradigma de programación orientada a objetos. Este lenguaje puede ser usado para modelar distintos tipos de sistemas: sistemas de hardware, sistemas de software y organizaciones del mundo real.

En la División de Servicios de Información (DSI) se han establecido estándares concernientes al diseño de sistemas, los cuales deben ser desarrollados por módulos y deben seguir el estándar definido por el Lenguaje Unificado de Modelado 2.1 (UML).

El diseño de un módulo, desarrollado en la DSI, debe contar con los siguientes diagramas:

- Diagrama de Casos de Uso
- Diagrama de Clases

- Diagrama de Secuencia

2.2.1.1 Diagrama de Casos de Uso El diagrama de casos de uso tiene el objetivo de mostrar todas las funcionalidades del sistema y además debe llevarse a cabo desde la perspectiva del usuario, ya que modela la interacción directa de este con el sistema; de su análisis se puede concluir si el sistema cumple satisfactoriamente con los requisitos de los usuarios.

Para el diagrama de casos de uso se identifican tres elementos básicos:

- **Actores:** Corresponden a los diferentes roles que los usuarios del sistema pueden tener. Cada rol debe ser único, es decir, distinguible de los demás, contando con un nombre específico y responsabilidades particulares al momento de interactuar con el sistema. Por lo tanto, es necesario aclarar que un rol representa a un tipo o categoría de usuarios del sistema e incluso un usuario puede tener asignado más de un rol en el sistema. Un actor es usualmente identificado como se muestra la figura 1:

Figura 1. Actor

- **Caso de Uso:** Es una secuencia de transacciones relacionadas, ejecutadas por uno o más actores y el sistema en un diálogo determinado. Un caso de uso describe una funcionalidad del sistema que produce un resultado apreciable para el usuario, sin entrar en detalles sobre cómo la realiza. Su comportamiento puede especificarse como un flujo de eventos en texto formal, o en pseudocódigo. En su conjunto, los casos de uso son los que describen todas las funcionalidades del sistema. En la figura 2 indica cómo se grafican los casos de uso en el diagrama.

Figura 2. Caso de Uso

Un caso de uso especifica dos tipos de secuencias o comportamientos:

a) **Secuencia Básica o Comportamiento Normal:** Son las acciones que ejecuta el actor sobre el sistema en el orden establecido en cada caso de uso.
b) **Secuencia o Comportamiento Alternativo:** Es el camino que el Actor invoca cuando este no lleva a cabo la secuencia básica en forma satisfactoria, es decir el actor no sigue los pasos correctos al momento de interactuar con el sistema.

- ***Relaciones:*** Son los elementos que conectan los anteriores elementos en el diagrama (Actores y Casos de Uso), los cuales confieren un significado a cada vínculo que establecen.

En un diagrama de casos de uso pueden existir los siguientes enlaces:

- **Relación de Generalización entre Actores:** Se utiliza cuando un actor hereda ciertas características de otro más general.

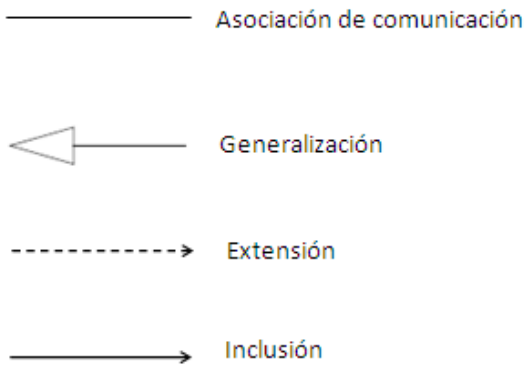
- **Relación de Generalización entre casos de Uso:** Indica cuando un caso de uso hereda y adiciona características de otro más general.
- **Relación de Extensión:** Es un enlace entre casos de uso que define cuando un caso de uso es extendido por otro, es decir, cuando un caso de uso tiene

variantes en su secuencia básica, la cual indica una extensión hacia la secuencia básica de otro.

- **Relación de Asociación:** Se utiliza para conectar un actor con un caso de uso e implica la existencia de una comunicación entre ellos.
- **Relación de Inclusión:** Señala cuando el comportamiento normal de un caso de uso por su naturaleza incorpora el comportamiento normal de otro.

La siguiente figura 3 ilustra los diferentes tipos de relaciones que pueden presentarse en un diagrama de casos de uso:

Figura 3. Tipos de Relaciones Casos de uso



2.2.1.2 Diagrama de Clases Sirve para modelar las clases que involucra el sistema, pero es preciso aclarar que el diagrama de clase representa el prototipo estático del sistema, ya que no explica el comportamiento del sistema en el tiempo.

Un diagrama de clases se compone de dos elementos: Clases y Relaciones

- **Clases:** Una clase es una construcción que modela un objeto perteneciente a un sistema. Comprende una serie de atributos para representar el estado de los objetos que pertenezcan a ella.

Estas clases presentan ciertas características:

- **Estado:** Es determinado por los atributos que contiene la clase y por sus posibles relaciones con otras clases.
- **Comportamiento:** Explica la funcionalidad de una clase, señalando todas las operaciones que esta puede realizar.
- **Identidad:** Implica la unicidad de cada objeto así comparta el mismo estado con otro(s).

Una clase es representada en la figura 4:

Figura 4. Representación de clase

Nombre Clase
- Atributos
+ Operaciones o Métodos

➢ **Relaciones** Es la connotación entre los objetos de dos o más clases. Las clases pueden ser interrelacionadas por medio de las siguientes relaciones:

- **Relación de Asociación:** Representa un conjunto de enlaces entre dos clases distintas, los cuales pueden ser unidireccionales o bidireccionales. Además contempla la Multiplicidad de Asociaciones que es la cantidad de objetos de cada clase en una relación.

- **Relación de Agregación:** Es la relación que existe entre una clase que envuelve o incluye a otras clases, cuyos tiempos de vida no dependen del tiempo de vida de la clase que las incluye.

- **Relación de Composición:** Cuando el tiempo de vida de un objeto de una clase depende del tiempo de vida de otro objeto que lo incluye.

- **Relación de Herencia:** Es el vínculo entre una súper clase y una o más subclases hijas, quienes heredan atributos y comportamientos de su clase padre y pueden extender las propiedades de dicha súper clase adicionando atributos que proporcionan un mayor detalle de lo que la clase padre representa.

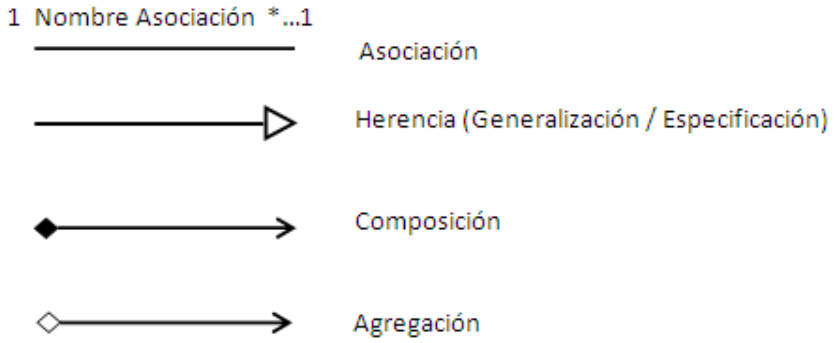
La herencia puede darse de dos formas:

- **Generalizada:** Ocurre cuando la Súper clase encapsula las propiedades y comportamientos de una o más subclases. En este tipo de relación las subclases no pueden heredar.

- **Especializada:** Se da cuando las subclases heredan las propiedades y comportamientos de una clase mayor dándole a esta última un mayor nivel de detalle.

La figura 5 exhibe las relaciones que pueden existir en el diagrama de clases, anteriormente descritas:

Figura 5. Tipos de relaciones



2.2.1.3 Diagrama de Secuencia El diagrama de secuencia muestra la dinámica del sistema, también en él se puede observar los módulos o clases que forman parte del programa y las llamadas que se hacen a cada uno de ellos para realizar una labor específica. Para describir la dinámica del sistema se hace necesario modelar un conjunto de diagramas de secuencia, los cuales pueden estar vinculados generalmente a una subfunción del sistema.

El diagrama de secuencia es uno de los diagramas más efectivos para modelar las interacciones entre objetos en un sistema. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase, este diagrama contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

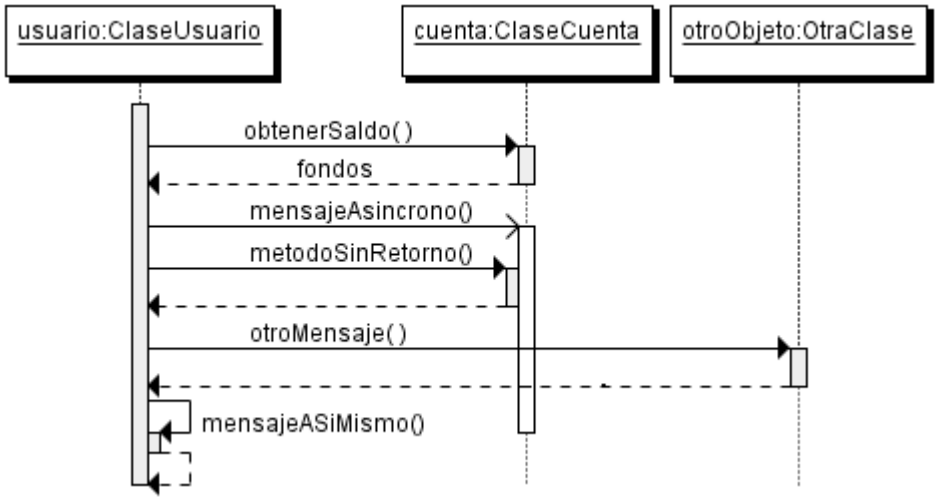
Normalmente se revisa la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se tiene modelada la descripción de cada caso de uso como una secuencia de varios pasos, entonces puedes seguir estos pasos para descubrir qué objetos son necesarios.

Para interactuar entre sí, los objetos se envían mensajes. Durante la recepción de un mensaje, los objetos se vuelven activos y ejecutan el método del mismo nombre. Por lo tanto se puede decir que un envió de mensaje es, por tanto, una llamada a un método.

El actor es el responsable de iniciar la secuencia esto se da en el momento que efectúa un mensaje u operación sobre el sistema. El diagrama de secuencia debe ser leído de izquierda a derecha y de arriba abajo; cada caso de uso tiene asociado un diagrama de secuencia, aunque puede tener más de uno dependiendo de los posibles comportamientos alternos por los cuales q el caso de uso puede optar.

En el diagrama de secuencia, los objetos que intervienen en el escenario se pueden representados por líneas discontinuas verticales, y los mensajes transmitidos por los objetos como flechas horizontales.

Figura 6. Diagrama de secuencias



Los tipos de mensajes que existes son los siguientes: síncronos y asíncronos. La función de los mensajes síncronos son los que ocurren cuando se hace una llamada a un método de un objeto el cual recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensaje es representado con flechas de cabeza llena. Mientras los mensajes asíncronos terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia a seguir. Son representados con flechas de cabeza abierta.

También se representa la respuesta a un mensaje con una flecha discontinua. Los mensajes son dibujados de forma cronológica desde la parte superior del diagrama hasta la parte inferior.

2.2.2 Tecnologías de Desarrollo de Aplicaciones Web

2.2.2.1 JAVA EE 5

2.2.2.1.1 Generalidades acerca de JAVA EE 5 En los últimos años gran cantidad de servicios y aplicaciones informáticas han migrado hacia los entornos web en pro de la facilidad de utilización por parte de los usuarios y a su vez la ejecución en múltiples plataformas e incluso se ha ampliado su cobertura hacia aplicaciones móviles. Los desarrolladores reconocen puntos comunes indispensables en las aplicaciones entre ellos se encuentran los siguientes: Distribuidas, eficientes, seguras, robustas y con alta capacidad transaccional. La respuesta de Java para estas necesidades es Java Enterprise Edition el cual que busca agilizar el proceso de desarrollo ofreciéndole a los desarrolladores toda una variada oferta de APIs, frameworks y tecnologías para responder a distintas necesidades.

Java Enterprise Edition 5 (Java EE 5) se centra en hacer más fácil el desarrollo. Sin embargo, conserva la riqueza de la plataforma J2EE 1.4. Ofrece funciones como Java Server Faces (JSF), componentes de transacción (EJB 3.0), persistencia de bases de datos (Java Persistence) y la tecnología de servicios web API, también reduce enormemente la necesidad de archivos XML de configuración (configuración del despliegue de las aplicaciones), lo que repercute en rapidez en el desarrollo ante la eliminación de código repetitivo. Java EE 5 hace que la codificación sea más simple y directa, pero mantiene el poder que ha establecido a Java EE como la primera plataforma para servicios web y desarrollo de aplicaciones empresariales.

2.2.2.1.2 Características principales de JAVA EE 5 A continuación se mencionarán algunas de las características más importantes de la tecnología JAVA EE 5.

- Es una aplicación robusta, comercial, gratis para el desarrollo, despliegue, y la redistribución.

- Contiene EJB 3.0 que apoyan a los POJOs, (Plain Old Java Objects), lo que significa que hay menor código a escribir, lo cual conlleva a un mantenimiento más ágil. EJB es una plataforma para la construcción portátil, reutilizable y escalable de aplicaciones de negocio utilizando el lenguaje de programación JAVA.

- La nueva Java Persistence (API) hace que el mapeo objeto-relacional sea más claro y más fácil.

- Tiene un mayor rendimiento, el tiempo de respuesta es más rápido, y la mejora de las características de administración para simplificar el despliegue es más significativa.

- Su tiempo de inicio es un treinta por ciento más rápido, con un treinta por ciento menos de memoria, respecto a versiones anteriores.

2.2.2.1.3 El modelo de aplicación JAVA EE La plataforma empresarial de Java fue diseñada para soportar aplicaciones que brinden servicios empresariales para clientes, empleados, proveedores, socios y demás personas o entidades que interactúan con la empresa.

El modelo de aplicación de Java EE empieza con el lenguaje de programación Java y la máquina virtual de Java. La seguridad, portabilidad y productividad de desarrollo probada, las cosas anteriormente mencionadas constituyen lo básico del modelo de aplicación. Java EE está diseñado para soportar aplicaciones que implementen servicios empresariales para clientes, empleados, proveedores, socios y otros que hacen demandas o contribuyen a la empresa.

El modelo de aplicación Java EE define una arquitectura para implementar servicios como aplicaciones de múltiples capas que distribuyen la escalabilidad, accesibilidad y facilidad de manejo que se necesita para aplicaciones empresariales.

2.2.2.1.4 Seguridad La plataforma Java EE dispone de reglas estándar para controlar el acceso. Estas son definidas por el desarrollador y se implementan al desplegar la aplicación en el servidor. Java EE pone a disposición de los desarrolladores mecanismos de acceso predefinidos para que ellos no tengan que implementarlos en sus aplicaciones. Una sola aplicación trabaja en diversos ambientes de desarrollo sin necesidad de modificar el código fuente.

2.2.2.1.5 Componentes Java EE Un componente es una unidad de software que está contenida y ensamblada en una aplicación Java EE, posee sus clases relacionadas e interactúa con los demás componentes que así lo especifiquen.

La plataforma Java EE define los siguientes componentes:

- Aplicaciones cliente y Applets, estos componentes se ejecutan en el cliente.

- Los componentes Web: Servlets, JavaServer Faces, y tecnología JavaServerPagesTM (JSPTM); se ejecutan en el servidor.

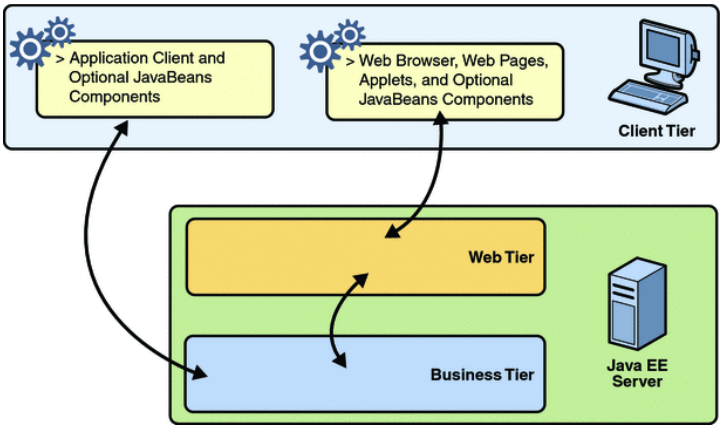
- Los componentes JavaBeansTM (EJBTM) empresariales (beans empresariales), son llamados componentes de negocios y se ejecutan en el servidor.

Los componentes Java EE se desarrollan con el lenguaje de programación Java y su compilación es igual que cualquier programa en este lenguaje. A diferencia de las clases Java estándar, los componentes Java se ensamblan en una aplicación y se verifica que estén bien formados, de acuerdo a las especificaciones de Java EE, se despliegan en ejecución donde son manejados por el servidor.

2.2.2.1.6 Comunicaciones del servidor Java EE La siguiente imagen ilustra los componentes que pueden formar la capa cliente, en los casos en que la interacción del

cliente con la capa del negocio se lleva a cabo en el servidor Java EE de forma directa, y en el caso de un cliente que se ejecuta en un navegador mediante una página JSP o un Servlet que se ejecuta en la capa web.

Figura 7. Comunicación del servidor[1]



En caso de que la aplicación Java EE haga uso de un navegador o una aplicación cliente pesada se debe tener cuidado en la decisión de cuál de los dos utilizar para equilibrar la funcionalidad en el cliente y acercarse al usuario para cargarle al servidor toda la funcionalidad que sea posible, esto facilita la distribución, despliegue y manejo de la aplicación.

2.2.2.1.7 Contenedores Java EE La arquitectura basada en componentes e independiente de la plataforma de la arquitectura Java EE hace que las aplicaciones Java EE sean fáciles de escribir ya que la lógica de negocio está organizada en componentes reutilizables. Adicionalmente el servidor Java EE brinda servicios de capas bajas en forma de contenedores para cada tipo de componente.

1 Oracle. *Documentación Java EE 5.* {En línea}. {20 de Marzo de 2011}. Disponible en: http://java.cabezudo.net/trabajos/JEE5/manual/jee5.v0.01.00/ababac.html

2.2.2.1.8 Soporte para servicios web Los servicios web son aplicaciones empresariales que utilizan un estándar abierto basado en XML y protocolos de transporte para intercambiar datos entre clientes. La plataforma proporciona una API para XML y las herramientas que se necesitan para diseñar, desarrollar, probar y desplegar rápidamente servicios web y clientes que operan totalmente con otros servicios web y clientes, los cuales se ejecutan en plataformas no propias de Java.

Para escribir servicios web y clientes con las APIs de XML de Java EE se deben pasar los datos en los parámetros de las llamadas a los métodos y procesar los datos retornados; en el caso de servicios web orientados a documentos, es necesario enviar documentos que contengan la comunicación del servicio en ambos sentidos. No se necesita programación a bajo nivel dado que la API de XML hace el trabajo de traducir los datos de la aplicación desde y hacia el flujo de datos basado en SML que es enviado a través de los protocolos estandarizados de transporte basados en XML.

La traducción de los datos a un flujo de datos estandarizado basado en XML es lo que hace que los servicios web y clientes escritos con las APIs de Java EE puedan operar entre ellos totalmente. Esto no necesariamente significa que los datos transportados incluyan etiquetas XML porque el transporte de los datos puede ser texto plano, datos XML o cualquier tipo de dato binario como audio, vídeo, mapas, ficheros de programa, documentos CAD o lo que sea requerido.

2.2.2.1.9 Ensamblaje y despliegue de una aplicación Java EE Una aplicación Java EE es empaquetada en una o más unidades estándar para ser desplegada en cualquier sistema compatible con la plataforma Java EE. Cada unidad contiene:

- Un componente o componentes funcionales (como un bean empresarial, página JSP, servlet o Applet).

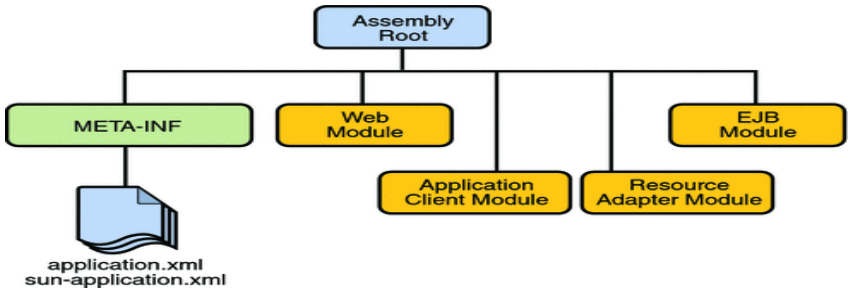
- Un descriptor de despliegue que describe su contenido.

Una vez que una unidad Java EE ha sido producida, está lista para ser desplegada. Su despliegue típicamente involucra el uso de una herramienta para especificar la

información de ubicación específica, como una lista de usuarios locales que pueden acceder a ésta y el nombre de la base de datos local. Una vez desplegado en una plataforma local, la aplicación esta lista para ejecutarse.

2.2.2.1.10 Empaquetado de aplicaciones

Figura 8. Estructura de un fichero EAR[2]



Una aplicación Java EE es distribuida en un Archivo Empresarial (EAR) que es un Archivo Java estándar (JAR) con una extensión .ear. El uso de archivos EAR y módulos hace posible ensamblar una gran cantidad de aplicaciones Java EE utilizando alguno de los mismos componentes. No se necesita codificación extra; es solo un tema de ensamble (o empaquetado) de varios módulos Java EE en u fichero EAR de Java EE.

Un fichero EAR contiene, como muestra en la figura 9, módulos Java EE y descriptores de despliegue.

Un descriptor de despliegue es un documento XML con una extensión .xml que describe la configuración de despliegue de una aplicación, un módulo o un componente. Dado que la información en el descriptor de despliegue es declarativa, esta puede ser cambiada sin la necesidad de modificar el código fuente. En tiempo de ejecución, el servidor Java EE lee el descriptor de despliegue y actúa sobre la aplicación, módulo o componente como corresponde.

---

2 Oracle. *Documentación Java EE 5.* {En línea}. {21 de Febrero de 2011}. Disponible en: http://java.cabezudo.net/trabajos/JEE5/manual/jee5.v0.01.00/acadad.html

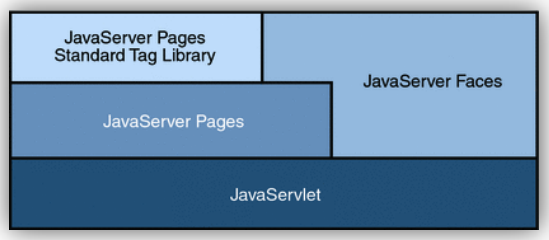
2.2.2.2 Aplicaciones web Una aplicación web es una extensión dinámica de una web o servidor de aplicación. Hay dos tipos de aplicaciones web:

- **Orientada a la presentación**: Una aplicación web orientada a la presentación genera páginas web interactivas que contienen varios tipos de lenguajes de marcas (HTML, XML y demás) y contenido dinámico en respuesta a las peticiones.

- **Orientadas a los servicios**: Una aplicación web orientada a los servicios implementa el punto final de un servicio web. Las aplicaciones orientadas a la presentación son a menudo clientes de aplicaciones orientadas a servicios.

Desde la introducción de la tecnología de Servlets y JSP, han sido desarrollados marcos de trabajo (frameworks) para construir aplicaciones web interactivas. La figura muestra estas tecnologías y sus relaciones.

Figura 9: Tecnologías Java para aplicaciones web[3]



2.2.2.3 Módulos web En la arquitectura Java EE, los componentes web y los ficheros con contenido estático como imágenes son llamados recursos web. Un módulo web es la unidad más pequeña de un recurso web que se puede utilizar y desplegar. Un módulo web Java EE corresponde con una aplicación web como se define en la especificación de Java Servlet. (Numeral 2.4.4)

3 Oracle. *Documentación Java EE 5.* {En línea}. {21 de Marzo de 2010}. Disponible en: http://java.cabezudo.net/trabajos/JEE5/manual/jee5.v0.01.00/acadab.html

Además de los componentes web y los recursos web, un módulo web puede contener otros ficheros:

- Clases utilitarias del lado del servidor (beans para bases de datos y demás). A menudo estas clases cumplen con la arquitectura JavaBeans.

- Clases del lado del cliente (applets y clases utilitarias).

Un módulo web tiene una estructura específica. El directorio más alto de la jerarquía de directorios de un módulo web es la raíz de documento de la aplicación. Es donde las páginas JSP, clases y archivos del lado del cliente y los recursos estáticos son almacenados.

La raíz de documentos contiene un subdirectorio llamado WEB-INF, que contiene los siguientes ficheros y directorios:

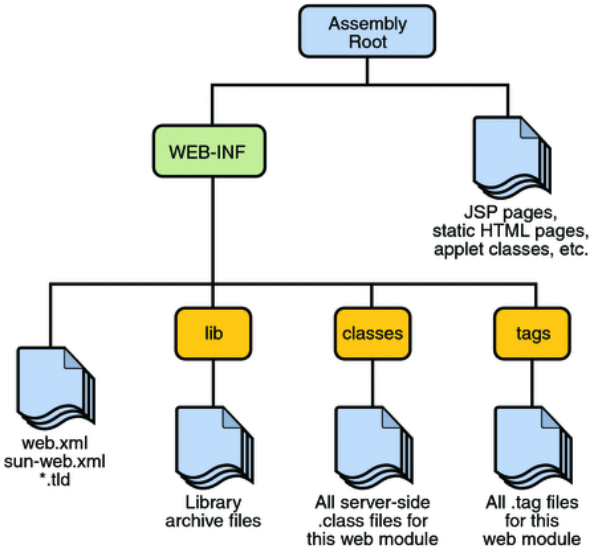
- web.xml: El descriptor de despliegue de aplicación.
- Los ficheros descriptores de las librerías de etiquetas.
- classes: Un directorio que contiene las clases del lado del servidor: componentes Servlets, clases utilitarias y JavaBean.
- tags: Un directorio que contiene ficheros de etiquetas, que son implementaciones de librerías de etiquetas.
- lib: Un directorio que contiene los archivos JAR de las librerías llamadas por las clases del lado del servidor.

Se pueden crear subdirectorios específicos de la aplicación (esto es, directorios de paquete) tanto en la raíz de documentos como en el directorio WEB-INF/classes/.

Un módulo web puede ser desplegado como una estructura de ficheros sin empaquetar o puede ser empaquetado en un fichero JAR conocido como un archivo web (WAR). Dado

que el contenido y uso de los ficheros WAR difieren de aquellos ficheros JAR, el nombre del fichero WAR utiliza una extensión .war. El módulo web descrito es portátil, se puede desplegar en cualquier contenedor web que cumpla con la especificación Java Servlet.

Figura 10: Estructura de un módulo web[4]



- **Manteniendo el estado del cliente**

Muchas aplicaciones necesitan que una serie de solicitudes de un cliente sean asociadas con otro. Las aplicaciones basadas en Web son responsables por mantener este estado, llamado sesión, ya que HTTP no tiene estado. Para soportar las aplicaciones que necesitan mantener el estado, la tecnología Servlet de Java proporciona una API para manejo de sesiones y permite varios mecanismos para implementar sesiones.

- **Accediendo a una sesión**

4 Oracle. *Documentación Java EE 5.* {En línea}. {25 de Marzo de 2011}. Disponible en: http://java.cabezudo.net/trabajos/JEE5/manual/jee5.v0.01.00/acadad.html

Las sesiones son representadas por un objeto HttpSession. Se accede a la sesión llamando el método getSession de un objeto request. Este método retorna la sesión actual asociada con esa solicitud, aunque si dicha solicitud no tiene una sesión entonces esta última es creada.

2.2.2.4 Tecnología de JavaBeans empresarial Un componente JavaBean empresarial (EJB), es un cuerpo de código que tiene campos y métodos para implementar módulos de lógica de negocio. Se puede pensar en un bean empresarial como un componente escrito en lenguaje Java encapsulado, en el lado del servidor, que puede ser utilizado solo con otros beans empresariales para ejecutar lógica de negocio en el servidor Java EE.

Hay dos tipos de beans empresariales: los beans de sesión y los beans manejadores de mensajes.

- Un bean de sesión representa una conversación transitoria con un cliente. Cuando el cliente termina de ejecutarse, el bean de sesión y sus datos desaparecen.

- Un bean manejador de mensajes combina las características de un bean de sesión y uno de escucha de mensajes permitiendo a un componente de negocio recibir mensajes de forma asincrónica. Comúnmente, estos son mensajes JMS (Java MessageService).

En Java EE5, los beans de entidad han sido reemplazados por entidades de la API de persistencia de Java (JPA). Una entidad representa datos persistentes almacenados en una fila de una tabla de la base de datos. Si el cliente termina o el servidor se apaga, el manejador de persistencia asegura que los datos de la entidad sean guardados.

2.2.2.5 Tecnología Java Servlet

- **¿Qué es un Servlet?**

Un servlet es una clase del lenguaje de programación Java que es utilizada para extender las habilidades de los servidores que guardan aplicaciones a las cuales se accede mediante el modelo petición-respuesta.
A pesar de que los servlets pueden devolver a cualquier tipo de respuesta, estos son comúnmente utilizados para extender las aplicaciones almacenadas en servidores web. Para estas aplicaciones, la tecnología Servlet Java define las clases servlets específicas para HTTP.

2.2.2.6 Tecnología de Java Server Pages La tecnología de JavaServerPages (JSP) nos permite colocar partes del código servlet directamente en un documento de texto. Una página JSP es un documento de texto que contiene dos tipos de texto: datos estáticos y elementos JSP, que determinan como la página construye el contenido dinámico.

- **API Java de servicio de mensajes**

La API del servicio de mensajes de Java (JMS) es un estándar de mensajería que permite a componentes de aplicación Java EE crear, enviar, recibir y leer mensajes. Para habilitar comunicación distribuida con bajo acoplamiento, confiable y asincrónica.

- **API Java de transacciones**

La API Java de transacciones (JTA) proporciona una interface estándar para distinguir transacciones. La arquitectura Java EE proporciona un commit automático para manejar commit y rollbacks. Un auto commit significa que otras aplicaciones que están viendo estos datos verán los datos actualizados luego de que cada base de datos realice la operación de lectura o escritura.

Sin embargo, si su aplicación realiza dos operaciones de acceso a base de datos que dependen una de otra se deseará utilizar la API JTA para distinguir donde la transacción completa incluyendo ambas operaciones, comienza, se regresa o se completa.

- **API JAVA de conectividad a bases de datos**

La API Java de conectividad a base de datos (JDBC) permite invocar comandos SQL desde métodos del lenguaje de programación Java. Se utiliza la API JDBC en un bean empresarial cuando se tiene un bean de sesión accediendo a la base de datos. Se puede utilizar también la API JDBC desde un servlet o una página JSP para acceder a la base de datos directamente sin pasar a través de un bean empresarial.

La API JDBC tiene dos partes: una interface a nivel de aplicación usado por los componentes de aplicación para acceder a la base de datos y una interface de proveedor de servicio para anexar un controlador JDBC (Java Database Connectivity) a la plataforma Java EE.

- **API Java de persistencia**

La API Java de persistencia es una solución Java basada en estándares para persistencia. La persistencia utiliza una estrategia de "mapeo" objeto relacional para unir la brecha entre el modelo orientado a objetos y la base de datos relacional. La persistencia de Java consiste en tres áreas:

- La API Java de persistencia
- El lenguaje de consultas
- Los datos de alto nivel con el mapeo objeto relacional

2.2.2.7 JAVA SERVER FACES (JSF)

La tecnología Java Server Faces constituye un marco de trabajo (framework) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

Los principales componentes de la tecnología Java Server Faces son:

- Una API y una implementación de referencia para:
    - Representar componentes de interfaz de usuario (UI-*User Interface*) y manejar su estado.
    - Manejar eventos, validar en el lado del servidor y convertir datos.
    - Definir la navegación entre páginas.
    - Soportar internacionalización y accesibilidad.
    - Proporcionar extensibilidad para todas estas características.

Una librería de etiquetas JavaServerPages (JSP) personalizadas para dibujar componentes UI (User-Interface) dentro de una página JSP.

Este modelo de programación bien definido junto con la librería de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones web con UIs en el lado servidor. Con un mínimo esfuerzo, es posible:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos en el lado servidor.
- Construir una interfaz de usuario con componentes reutilizables y extensibles.

Como se puede apreciar en la figura 13, la interfaz de usuario que se crea con la tecnología JavaServer Faces se ejecuta en el servidor y se renderiza en el cliente.

- **Beneficios de la Tecnología Java Server Faces**

Una de las ventajas de que JSF sea una especificación estándar es que pueden encontrarse implementaciones de distintos fabricantes. Esto permite no vincularse exclusivamente con un proveedor concreto, y poder seleccionar el más adecuado según los requerimientos de la aplicación; según el número de componentes que suministra, el rendimiento de éstos, soporte proporcionado, precio, política de evolución, etc.

JSF trata la vista (la interfaz de usuario) de una forma algo diferente a lo que se está acostumbrado en las aplicaciones web, donde la programación de la interfaz se desarrolla a través de componentes y está basada en eventos (pulsación de un botón, cambio en el valor de un campo, etc.).

JSF es muy flexible ya que permite personalizar tanto los componentes como la recarga de la vista de las páginas, con el fin elaborar interfaces de usuario en la forma que más nos convenga.

Figura 11: Diagrama de una aplicación JSF[5]

[5] SICUMA: Sistemas de Información Cooperativos Universidad de Málaga. *Tutorial de JavaServer Faces.*P. 5 {En línea}. {10 de abril de 2011}. Disponible en: http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf

La tecnología Java Server Faces permite construir aplicaciones web que introducen realmente una separación entre el comportamiento y la presentación, separación sólo ofrecida tradicionalmente por arquitecturas UI del lado del cliente y parcialmente por la tecnología JSP.

Separar la lógica de negocio de la presentación también permite que cada miembro del equipo de desarrollo de la aplicación web se centre en su parte asignada del proceso de diseño, y proporciona un modelo sencillo de programación para enlazar todas las piezas.

Otro objetivo importante de la tecnología Java Server Faces es mejorar los conceptos asociados con componente-UI y capa-web sin limitarse a una tecnología de *script* particular o un lenguaje de marcas. Aunque la tecnología Java Server Faces incluye una librería de etiquetas JSP personalizadas para representar componentes en una página JSP, las APIs de Java Server Faces se han creado directamente sobre el API JavaServlet. Esto permite, teóricamente, hacer algunas cosas avanzadas: usar otra tecnología de presentación junto a JSP, crear componentes propios directamente desde las clases de componentes, y generar salida para diferentes dispositivos cliente; entre otras.

- **¿Qué es una aplicación Java Server Faces?**

En su mayoría, las aplicaciones Java Server Faces son como cualquier otra aplicación web Java. Se ejecutan en un contenedor de servlets de Java y, típicamente, contienen:

• Componentes JavaBeans conteniendo datos y funcionalidades específicas de la aplicación.

• Oyentes de Eventos.

• Páginas, (principalmente páginas JSP).

• Clases de utilidad del lado del servidor, como beans para acceder a las bases de datos.

Además de estos ítems, una aplicación Java Server Faces también tiene:

• Una librería de etiquetas personalizadas para implementar componentes UI en una página.

• Una librería de etiquetas personalizadas para representar manejadores de eventos, validadores y otras acciones.

• Componentes UI representados como objetos con estado en el servidor.

Toda aplicación Java Server Faces debe incluir una librería de etiquetas personalizadas que define las etiquetas que representan componentes UI, así como una librería de etiquetas para controlar otras acciones importantes, como validadores y manejadores de eventos. La implementación de Java Server Faces, de Sun Microsystems, proporciona estas dos librerías. La librería de etiquetas de componentes elimina la necesidad de codificar componentes UI en HTML u otro lenguaje de marcas, lo que se traduce en el empleo de componentes completamente reutilizables. Y la librería principal (core) hace fácil registrar eventos, validadores y otras acciones de los componentes.

Finalmente, la tecnología Java Server Faces permite convertir y validar datos sobre componentes individuales e informar de cualquier error antes de que se actualicen los datos en el lado del servidor.

2.2.2.8 Modelo Vista Controlador en JSF El patrón MVC (Modelo Vista Controlador), permite separar la lógica de control (qué cosas hay que hacer pero no cómo), la lógica de negocio (cómo se hacen las cosas) y la lógica de presentación (cómo interactuar con el usuario).

Utilizando este tipo de patrón es posible conseguir más calidad, un mantenimiento más fácil, perder el miedo al folio en blanco (existe un patrón de partida por el que empezar un

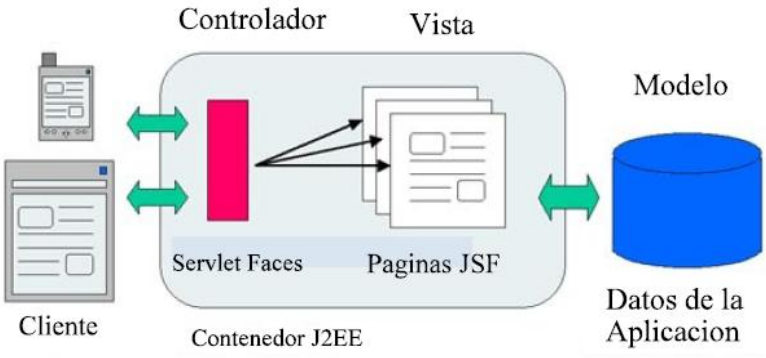
proyecto), entre otras ventajas. Al margen de todo esto, una de las cosas más importantes que permite el uso de este patrón consiste en normalizar y estandarizar el desarrollo de Software.

Este modelo de arquitectura presenta otras importantes ventajas: • Hay una clara separación entre los componentes de un programa; lo cual admite implementarlos por separado.

• Se cuenta con una API muy bien definida; cualquiera que use la API, podrá reemplazar el modelo, la vista o el controlador, sin dificultad.

• La conexión entre el modelo y sus vistas es dinámica: se produce en tiempo de ejecución, no en tiempo de compilación.

Figura 12: Modelo vista controlador en JSF[6]



2.2.2.8.1 Modelo El modelo es el objeto que representa y trabaja directamente con los datos del programa: gestiona los datos y controla todas sus transformaciones. El modelo no tiene conocimiento específico de los diferentes controladores y/o vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuándo deben reflejar un cambio en el modelo.

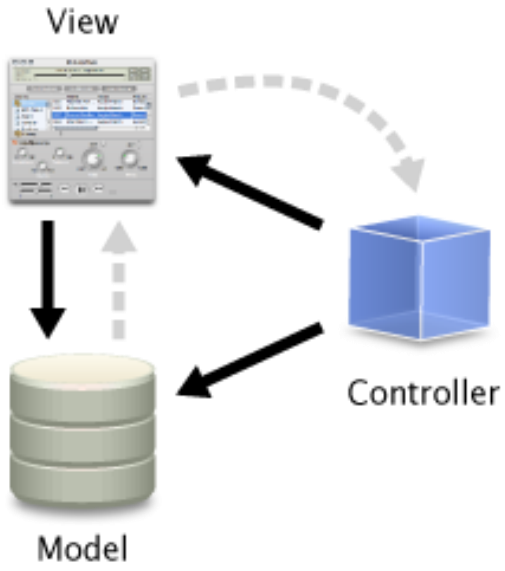
[6] SICUMA: Sistemas de Información Cooperativos Universidad de Málaga. *Tutorial de JavaServer Faces.*P. 18 {En línea}. {28 de Marzo 2011}. Disponible en: http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf

2.2.2.8.2 Vista La vista es el objeto que maneja la presentación visual de los datos gestionados por el Modelo. Genera una representación visual del modelo y muestra los datos al usuario e interacciona con el modelo a través de una referencia al propio modelo.

2.2.2.8.3 Controlador El controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo. Entra en acción cuando se realiza alguna operación, ya sea un cambio en la información del modelo o una interacción sobre la Vista. Se comunica con el modelo y la vista a través de una referencia al propio modelo.

Además, JSF opera como un gestor que reacciona ante los eventos provocados por el usuario, procesa sus acciones y los valores de estos eventos, y ejecuta código para actualizar el modelo o la vista.

Figura 13. Modelo Vista-Controlador



En la figura 13[7] se puede observar las relaciones entre el Modelo, la Vista y el Controlador. Cabe destacar en la figura las líneas continuas que significan una relación directa como también las líneas discontinuas que implican una relación indirecta. También es

[7] {En línea}. {30 de Marzo 2011}. Disponible en: http://blogdeaitor.wordpress.com/2008/10/20/model-view-controller/

importante tener en cuenta que aunque puede haber cierta “referencia indirecta” entre el Modelo y la Vista, el primero sigue sin saber nada del segundo.

El modo en que interactúan los tres elementos del modelo MVC se describe a continuación:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta dirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (del modelo) a la vista aunque puede dar la orden a la vista para que se actualice.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

2.2.2.9 Servidor de Aplicaciones – Jboss El servidor de aplicaciones JBoss, es una herramienta certificada para el desarrollo de aplicaciones empresariales Java. Su madurez y el esfuerzo de muchos desarrolladores, e incluso las sugerencias que han realizado sus usuarios, han permitido que JBoss AS (Application Server) se popularice ampliamente y sea común en los currículos de desarrolladores.

Es reconocido por soportar los estándares más recientes. De hecho, es el primer servidor de aplicaciones en alcanzar la certificación J2EE 1.4 cuando salió su versión 4.0. Pero JBoss no sólo marca la pauta en la adopción de estándares con su servidor de aplicaciones, sino en la imposición de los mismos. Hace parte del *Java Community Process* (JCP).

2.2.2.10 SEAM Seam es un framework de aplicaciones de Java Enterprise Edition inspirado en los siguientes principios:

Seam define un modelo de componentes uniformes para la lógica del negocio en su aplicación. Un componente Seam puede tener un estado, que se encuentra asociado a cualquiera de los diversos contextos bien definidos, incluyendo el de larga duración, el contexto de persistencia, de procesos de negocio y el contexto de la conversación, que se conserva en todas las solicitudes múltiples en una interacción con el usuario.

En Seam no hay distinción entre los componentes del nivel de presentación y componentes de la lógica del negocio. El desarrollador puede estratificar la aplicación de acuerdo con la arquitectura que se haya diseñado. Los componentes Seam pueden simultáneamente tener acceso al estado asociado a la solicitud web y al estado de recursos transaccionales.

- ✓ **Integrar JSF con EJB 3.0**

EJB 3 es un modelo de componentes a nivel de lógica de negocio y de persistencia, del lado del servidor, mientras que JSF es un modelo de componentes de la capa de presentación.

JSF y EJB3 funcionan mejor juntos, a pesar de que Java EE5 no proporciona un estándar para integrar estos modelos de componentes. No obstante los creadores de ambos modelos (JSF y EJB3) proporcionan puntos de extensión estándar para permitir la integración con otros marcos.

Seam unifica los modelos de componentes de JSF y EJB3, dejando al desarrollador el único problema de diseñar la lógica del negocio.

- ✓ **Integrar AJAX**

Seam soporta mejor las soluciones de código abierto basado en AJAX JSF: JBossRichFaces e ICEfaces. Estas soluciones le permiten agregar la capacidad de AJAX para la interfaz de usuario sin necesidad de escribir código JavaScript.

Por otra parte, Seam proporciona una capa incorporada del Javascript que le permite llamar a los componentes JavaScript de forma asincrónica del lado cliente sin la necesidad de una capa de acción intermedia.

Estos enfoques funcionan bien porque Seam incorporó la gestión de concurrencia y el estado, que aseguran que las peticiones asincrónicas Ajax se manejan de forma segura y eficiente en el servidor.

- ✓ **Procesos de negocio como el primer constructor de clase**

Seam ofrece transparencia en la gestión de procesos de negocio a través de JBPM, incluso permite definir el flujo de páginas de la capa de presentación utilizando el mismo lenguaje que jBPM utiliza para la definición de procesos de negocio.

- ✓ **Declarativa administración del estado**

Tradicionalmente, las aplicaciones J2EE implementan la administración de estado de forma manual. Este enfoque es la fuente de muchos errores y pérdidas de memoria cuando las aplicaciones no pueden limpiar los atributos de sesión, o cuando los datos de sesión asociados a diferentes flujos de trabajo chocan en una aplicación de múltiples ventanas. Seam tiene el potencial de eliminar casi por completo esta clase de errores.

La declarativa de administración del estado se ha logrado gracias a que Seam extiende el modelo de contexto definido por la especificación de servlets (petición, sesión, aplicación) con dos nuevos contextos (conversación y procesos de negocio).

- ✓ **Biyección**

La biyección es dinámica y bidireccional, se puede pensar en esto como un mecanismo de alias para variables contextuales (nombres en alguno de los contextos enlazados al hilo de ejecución actual) a atributos del componente.

La biyección permite auto ensamblaje de componentes con estado por el contenedor, lo que incluso permite a un componente asegurar y fácilmente manipular el valor de una variable contextual, simplemente asignándola a un atributo del componente.

- ✓ **Preferir anotaciones a XML**

La comunidad Java ha estado confundida sobre el tipo de meta información con la que cuenta la configuración; las anotaciones de Java han logrado cambiar esto.

EJB 3.0 abarca las anotaciones y la configuración de excepción como la forma más fácil de proporcionar información al contenedor en forma declarativa. Seam extiende las

anotaciones de EJb3 con una serie de anotaciones para la administración del estado declarativo y demarcación del contexto declarativo.

- ✓ **La prueba de integración es fácil**

Los componentes de Seam, siendo simples clases Java, son por naturaleza comprobables. Sin embargo, para aplicaciones complejas, las pruebas unitarias por sí solas son insuficientes. Seam proporciona la capacidad de prueba de aplicaciones seam como una característica central del framework. El usuario puede escribir las pruebas JUnit o TestNG que reproducen una interacción completa con un usuario. Estas pruebas pueden ser ejecutadas directamente en el IDE, donde Seam automáticamente implementa los componentes EJB con JBoss Embebido.

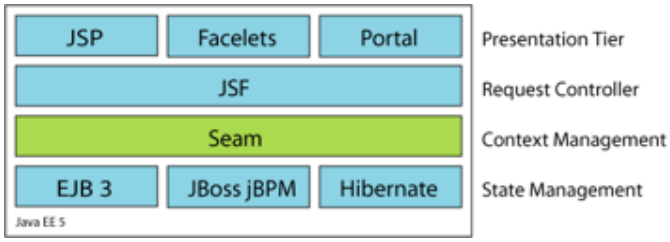
- ✓ **Hay más de una aplicación web que sirve páginas HTML**

Un framework de aplicaciones web realmente completo debe abordar problemas como la persistencia, la concurrencia, asincronía, administración del estado, seguridad, correo electrónico, mensajería, pdf, generación de gráficos, servicios web, caché, entre otros.

Seam integra JPA e Hibernate3 para persistencia, EJB TimerService y Quartz para ligeras asincronías, jBPM para flujo de trabajo, JBoss rules para reglas del negocio, Meldware Mail para correo electrónico, HibernateSearch y Lucene para búsqueda de texto, JMS para mensajes y JBoss Caché para la página de almacenamiento en caché.

Seam funciona en cualquier servidor de aplicaciones Java EE y también en Tomcat. Si el IDE no admite EJB 3.0 puede utilizar Seam incorporado en la gestión de transacciones con JPA o Hibernate3 para la persistencia. También se puede utilizar JBoss embebido en Tomcat, y tener un soporte completo para EJB 3.0.

Figura 14. Framework Seam[8]



2.2.2.11 El modelo de componentes contextuales Para conocer el núcleo del Framework de Seam es necesario entender dos conceptos esenciales: los contextos y los componentes.

Contextos de Seam:

Los contextos son contenedores gestionados por el mismo framework de Seam en los cuales residen todos los componentes instanciados y que pueden ser demarcados por medio de anotaciones.

Contexto Sin Estado o Stateless: Es un contexto que contiene únicamente componentes sin estado (Stateless) lo que significa una ausencia de contexto ya que las resoluciones de una instancia de Seam no son almacenadas. No obstante se han desarrollado y utilizado ya que son una parte importante de cualquier aplicación de Seam.

Contexto de Evento: Considerado como el contexto de estado "más estrecho", proporciona una generalización de la noción de contexto de petición Web para cubrir otros tipos de eventos. Un claro ejemplo de contexto de evento tiene que ver con el ciclo de vida de una petición JSF en el cual los componentes invocados por una solicitud JSF en un contenedor de evento, son eliminados inmediatamente después responder con la petición.

Contexto de Página: Este contexto permite asociar el estado de un componente con la instancia del llamado de una página, es decir, Seam crea el contexto al momento de

8 JBoss. Página oficial Seam. *Community documentation.* {En línea}. {25 de Abril de 2011}. Disponible en: http://docs.jboss.org/seam/2.1.2/reference/en-US/html_single/#Book-Preface

direccionar una página. Es muy útil al momento de requerir listas de hacer clic (Combo Box) donde cada lista es devuelta por el cambio en los datos en el servidor. Los componentes perduran mientras se permanezca en la misma página.

Contexto de Conversación: Es posiblemente el lugar ideal para guardar el estado de una aplicación ya que permite al desarrollador implementar casos de uso relativamente extensos o realizar transacciones relativamente largas. Una "conversación" es una unidad de trabajo desde el punto de vista del usuario, lo que abarca varias peticiones e incluso varias transacciones con la base datos. La conversación mantiene su estado asociándolo a cada ventana en forma individual con el fin de evitar posibles colisiones entre conversaciones, ya que un usuario puede estar manejando múltiples conversaciones al mismo tiempo (por ejemplo en el caso de tener la misma página en más de una instancia del navegador).

Las conversaciones también se conocen como "tareas", en consecuencia, una tarea es una conversación significativa en términos de un proceso de negocio extenso y tiene el potencial para disparar una transición del estado de un proceso de negocio cuando este es satisfactoriamente culminado. Seam controla el estado de las conversaciones mediante un tiempo de espera que se puede configurar con el propósito de evitar el incremento de conversaciones inactivas de un usuario en sesión; dado el caso que un usuario abandone la conversación. Otra característica particular del contexto de conversación es la posibilidad de tener conversaciones anidadas; en otras palabras una conversación contenida dentro de otra mayor.

Contexto de Sesión: Mantiene el estado de los componentes asociados al inicio de sesión de un usuario. Este contexto puede ser asociado con la interface HttpSession, sin embargo el contexto de sesión de Seam fue diseñado para manejar la Sincronización (Serialización de peticiones para evitar colisiones de solicitudes) y la Clusterización (facilidad en la distribución de componentes); ventajas que le confieren una verdadera robustez a cualquier aplicación web.

Contexto de Proceso de Negocio: Se encuentra asociado con una ejecución de proceso de negocio largo que abarca múltiples interacciones entre múltiples usuarios lo que implica

que el estado sea compartido, manejado y persistido por el motor BPM (Business Process Management). Seam carece de anotaciones para realizar la demarcación de este contexto por lo que se debe manejar en forma externa utilizando un Lenguaje de Definición de Procesos.

Contexto de Aplicación: Es el contexto más general de la especificación de servlets. Este contexto se utiliza generalmente para guardar información estática como los datos de configuración, de referencia o meta-modelos. Seam hace uso de este contexto al establecer su propia configuración y meta-modelos.

- ✓ **Prioridad de búsqueda de los Contextos Seam**

Algunas veces las instancias de componentes son obtenidas desde un ámbito (Scope) particular conocido, pero cuando no se conoce el ámbito del componente a instanciar Seam consulta en todos los Scopes con estado bajo un cierto orden de prioridad:

- Contexto de Evento.
- Contexto de Página.
- Contexto de Conversación.
- Contexto de Sesión.
- Contexto de Proceso de Negocio.
- Contexto de Aplicación.

- ❖ **Componentes Seam:**

Los componentes son objetos con estado (Stateful), generalmente son EJBs (Enterprise JavaBeans), cuya instancia implica una asociación con un contexto en la cual a cada objeto se le asigna una única identidad.

Los componentes Seam son POJOs (acrónimo de Plain Old Java Object), es decir, son instancias de clases que no extienden o implementan nada adicional (como la implementación de interfaces en Hibernate). A pesar de que Seam es un framework desarrollo para integrarse profundamente con el estándar EJB 3.0, sus componentes pueden utilizarse por fuera del contenedor EJB 3.0.

Bean de Sesión con Estado: Estos componentes no solo son capaces de mantener el estado de la aplicación a través de múltiples invocaciones a un Bean sino también a lo largo de múltiples peticiones. Una de las características exclusivas de Seam es la manera como se mantiene la información de una conversación en curso, ya que esta es almacenada en variables de instancia de Sesión Stateful asociadas a este contexto, en lugar de adherirla directamente en el HttpSession.

A menudo los Bean de Sesión con Estado son utilizados como oyentes de acciones JSF aunque nunca deberían ser enlazados ni con el contexto de página ni con el contexto Stateless. Además en el ámbito de sesión, las peticiones concurrentes de Beans de Sesión Stateful estarán serializadas evitando posibles colisiones; siempre y cuando los interceptores de Seam no estén deshabilitados para el Bean.

Beans de Entidad: Los Beans de Entidad pueden ser ligados a una variable de contexto o a una función como componentes Seam. Como las entidades tienen una identidad de persistencia adicional a su identidad contextual, las instancias de entidad están explícitamente ligadas al código Java, en lugar de ser creadas implícitamente por el framework.

Como los Beans de Entidad son componentes que no soporta la biyección no suelen usarse como oyentes de acciones JSF pero si pueden emplearse como Beans de soporte para proveer propiedades a los componentes JSF tanto en el envío como en el despliegue de formularios.

Los Beans de Entidad están destinados al contexto de conversación por defecto y nunca debe pertenecer a un contexto Stateless; también hay que tener en cuenta que es más eficiente tener una referencia al Bean de entidad en un Bean de Sesión Stateful en ambientes distribuidos.

Java Beans: Estos componentes pueden ser utilizados con los Beans de Sesión Stateful, sin embargo no cuenta con las mismas funcionalidades de este último, entre las que se encuentran:

- Demarcación de transacciones declarativa.
- Seguridad declarativa.
- Replicación del estado distribuido eficiente.
- Persistencia EJB 3.0.
- Métodos de Tiempo de Espera.

❖ Modelo de Concurrencia

En la actualidad las aplicaciones web deben lidiar con la concurrencia de solicitudes ya que estas deben soportar procesos asíncronos como las peticiones AJAX y en consecuencia Seam debe gestionar algunos Contextos contemplando la posibilidad de una colisión entre solicitudes.

Los contextos de Sesión y Aplicación son multi-hilo en tanto que los contextos de evento y de página son naturalmente de un solo hilo. No obstante en el contexto de conversación se debe proteger de las solicitudes concurrentes y es por ello que Seam maneja un modelo de un sólo hilo por proceso para cada conversación, identificando cada solicitud con un serial para tener un control adecuado sobre las peticiones concurrentes.

❖ Eventos Seam

El modelo de componentes Seam fue desarrollado para su uso con aplicaciones orientadas a eventos, específicamente para permitir el desarrollo de componentes de grano fino, débilmente acoplado en un modelo de grano fino. Los eventos en seam son de varios tipos:

- JSF eventos
- jBPM eventos de transición
- Seam acciones de página
- Seam impulsado en componentes eventos
- Seam contextuales eventos

❖ ***Página de acciones***

Una acción de página seam es un evento que ocurre antes de redirigir una página.

El método de acción de página puede devolver un resultado JSF. Si el resultado no es nulo, seam utiliza las reglas de navegación que se hayan definido para navegar a una vista.

La identificación mencionada en el elemento <page> no tiene por qué corresponder a una página real o una página JSP Facelets, por lo tanto, se puede reproducir la funcionalidad de un marco tradicional orientado a la acción como Struts o WebWork con acciones de página. Esto es muy útil para hacer cosas complejas en respuesta a la non-faces (por ejemplo, las solicitudes HTTP GET).

❖ **Página de parámetros**

Una petición a JSF Faces (envío de un formulario) encapsula una acción y los parámetros de entrada. Los parámetros de página pueden utilizarse con o sin acción.

Navegación: Puede utilizar las reglas estándar de navegación JSF se define en el faces-config.xml en una aplicación de seam. Sin embargo, las reglas de navegación JSF tienen una serie de limitaciones:

- No es posible especificar parámetros de la petición usada para redirigir la página.
- No es posible iniciar o finalizar las conversaciones de una regla.
- En la evaluación del método de retorno no es posible evaluar una expresión EL arbitraria.

❖ **Eventos impulsados por componentes**

Los componentes de seam pueden interactuar simplemente llamando a cada uno de los demás métodos. Los componentes con estado pueden incluso poner en práctica el modelo observador/observable. Pero permitir que los componentes que interactúen de una manera débilmente acoplado es posible cuando los componentes llaman a otros métodos directamente. Seam proporciona eventos impulsados por componentes.

❖ **Eventos Contextuales**

Seam define una serie de funciones que pueden ser usados para tipos especiales de integraciones con el framework. Algunas de ellas son:

- org.jboss.seam.validationFailed: invocada cuando falla la validación JSF.
- org.jboss.seam.noConversation: invocada cuando no hay una conversación larga y es requerida.
- org.jboss.seam.postDestroyContext.<SCOPE> : invocada después de<SCOPE> el contexto es destruido.
- org.jboss.seam.beforePhase: llamada antes de iniciar una fase JSF.
- org.jboss.seam.security.notLoggedIn: llamada cuando el usuario no se ha autenticado y se requiere la autenticación.

❖ **Administrar excepciones**

JSF es limitado en el manejo de excepciones, para esto seam permite definir una clase particular para anotar la clase de excepción o se declara en un archivo .xml. Este servicio

se combina con la anotación ApplicationException EJB 3.0 que especifica si la excepción debe provocar una reversión en la transacción.

- **Conversaciones y gestión de espacio de trabajo**

Modelo de conversación Seam

Seam se propaga de forma transparente en el contexto de conversación mediante las devoluciones de datos JSF y las redirecciones. Si no se hace nada especial o no realiza una petición el contexto de la conversación no se propaga y se procesa en una nueva conversación temporal; este es el comportamiento deseado.
Si desea propagar la conversación seam a través de una solicitud non-faces se debe codificar explícitamente la conversación seam identificando el parámetro de la petición.

El modelo de conversación seam facilita crear aplicaciones con múltiples ventanas; algunas de estas aplicaciones requieren adicionalmente:

La conversación se extiende por unidades más pequeñas, que se ejecutan en serie o a la vez.
El usuario puede alternar en diferentes conversaciones dentro de una misma ventana, esto se llama gestión del área de trabajo.

- **Conversaciones anidadas**

Se crean invocando el método @Begin(nested=”true”) dentro del ámbito de la aplicación de una conversación existente. Una conversación anidada posee su propio contexto de conversación, pero puede leer los valores de contexto de la conversación exterior que en este caso son de solo lectura.

La anidación de una conversación se inicia en un contexto dentro de la conversación externa, esta es considerada como la conversación padre.
Los objetos cargados directamente en el contexto de la conversación anidada no afectan los objetos accesibles en la conversación padre.
La inyección en un contexto de búsqueda en la primera conversación, busca el valor en el contexto de la conversación en curso, si no encuentra ningún valor continúa por la conversación en pila si la conversación es anidada.

La conversación anidada se destruye y se reanuda la conversación externa cuando encuentra la etiqueta @End.

Las conversaciones pueden ser anidadas a cualquier profundidad arbitraria.

- **Iniciando conversaciones con la solicitud GET**

JSF no define ningún tipo de detector de acción cuando una página se accede a través de una solicitud non-faces, esto puede ocurrir si el usuario marca la página o si accede a ella a través de un link.

Cuando se desea iniciar una conversación al acceder a una página, dado que no existe un método de acción JSF no se puede resolver el problema con la anotación @Begin.

Si la página necesita algún estado de la variable de contexto y este se lleva a cabo en un componente de seam, el estado puede alcanzarse en el método @Create. Si no, puede definirse el método @Create.

Si ninguna de las opciones funciona, seam permite definir una página de acciones en el archivo pages.xml.

2.2.2.12 Navegación en SEAM

- **Modelo de navegación Stateless**

Existen dos formas para definir un modelo de navegación Stateless: (a) por medio de las reglas de Seam o (b) utilizando las reglas de JSF.

El modelo Stateless define un conjunto de salidas lógicas y salidas de nombre de un evento directamente a la página resultante de la vista. Las reglas de navegación son totalmente ajenas a cualquier estado en poder de la aplicación aparte de la página fuente del evento. Esto implica que los métodos de acción oyente (actionlistenermethods) deben en algunos escenarios tomar decisión con respecto al flujo de página, ya que sólo tienen acceso al estado actual de la aplicación.

Cuando una aplicación utiliza la paginación Stateless, Seam le permite al usuario navegar libremente por medio de los botones: *Regresar, Adelante* o *Refrescar*; siempre y cuando la

aplicación se responsabilice de permanecer en el estado conversacional internamente consistente cuando se utilizan los botones anteriormente mencionados.

Una ventaja de implementar este tipo de navegación radica en utilizar reglas simples, de ser flexible, y además, de permitir el retorno de IDs de vista desde los métodos de acción oyente. No obstante, la paginación Stateless no soporta procesos de negocios complejos.

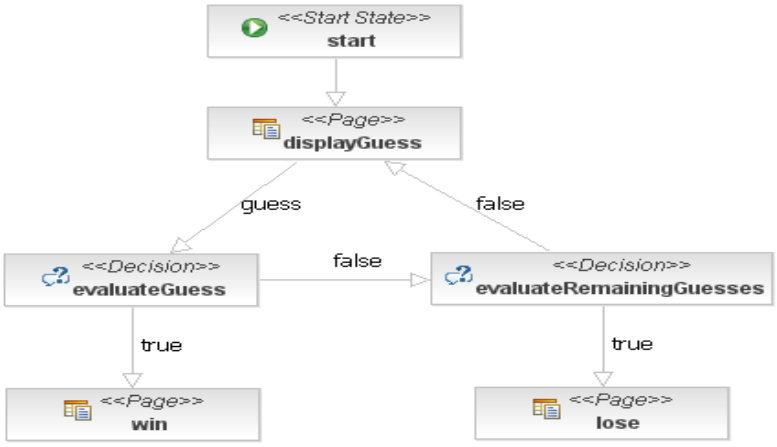
- **Modelo de Navegación Stateful**

Define un conjunto de transiciones entre un conjunto de estados lógicos y estados de nombres de la aplicación. Este modelo está orientado a los procesos de negocio en el que es posible establecer el flujo producido por cualquier interacción del usuario, en su totalidad, con la definición del flujo de página de JPDL e incluso escribir métodos de acción oyentes que desconocen por completo el flujo de la interacción. JPDL es un lenguaje xml simple y de fácil lectura, el cual es instaurado por el Motor de Gestión de procesos de Negocios de Jboss (JBPM). JPDL logra resolver los siguientes problemas:

- Definición del flujo de página involucrado en complejas interacciones de usuario (definir el flujo de página de una conversación en particular).
- Definición de los procesos de negocio globales (cuando los procesos de negocios involucran múltiples conversaciones con múltiples usuarios simultáneamente).

A pesar de que la navegación Stateful tiene un diseño para soportar cualquier interacción del usuario, al mismo tiempo es un poco más restringida que la Stateless, ya que todos los posibles eventos deben estar definidos por JPDL y no pueden interactuar con métodos de acción oyente que personalizan las interacciones del usuario sobre la aplicación.

Figura 15. Modelo de Navegación Stateful[9]



2.2.2.13 SEAM y el mapeo Objeto-Relacional Seam provee un extenso soporte para las Arquitecturas de Persistencia más importantes de Java: (a) Hibernate 3 y (b) Java Persistence API presentada por EJB 3.0. Entender cuál es la relación de Hibernate 3 y EJB 3.0 con Seam, o como se integran estos ORM con el Framework es de vital importancia para conocer las ventajas más importantes de Seam.

Una de las razones que impulsaron el desarrollo de Seam fue la dificultad que tenían otros Frameworks como Spring (que utilizaba Hibernate) para persistir los objetos, ya que cada transacción con la base de datos era atómica, es decir, que cada vez que una transacción finalizaba no sólo implicaba una interacción directa con la base de datos sino que también marcaba la pérdida del contexto de persistencia.

2.2.2.14 El marco de aplicaciones SEAM Con seam es más sencillo crear aplicaciones escribiendo clases java con anotaciones. El marco de aplicaciones de seam permite reducir la cantidad de código necesario para acceder a la base de datos en una aplicación web, para esto se usa Hibernate o JPA.

- **Objetos Principales**

9 JBoss. Página oficial Seam. *Community documentation.* {En línea}. {26 de abril 2011}. Disponible en: http://docs.jboss.org/seam/2.1.2/reference/en-US/html_single/#d0e6695

Proporcionan operaciones de persistencia para una entidad en particular, las operaciones pueden ser: persist (), remove (), update () y getInstance (). Antes de usar remove o update se debe establecer el identificador del objeto con el método SetId().

- **Controlador de objetos**

Una parte opcional del frameworkseam es la clase del controlador y sus subclases EntityControllerHibernateEntityController y BusinessProcessController, que ofrecen algunos métodos de conveniencia para el acceso a los componentes integrados.

2.2.2.15 Almacenamiento en caché de SEAM En la mayoría de las aplicaciones empresariales, la base de datos es el principal cuello de botella como también es la capa menos escalable en el entorno de ejecución. En conclusión es casi imposible que una aplicación sea escalable mientras la interacción con la base de datos sea ostensible. Por lo tanto la escalabilidad de cualquier aplicación se puede favorecer si se disminuyen las transacciones directas con la base de datos, y esa es la principal misión de la caché.

- **Almacenamiento en Caché multi-capa**

  Con Seam se puede planificar para configurar un almacenamiento en una caché de manera individual para cada una de las capas de la aplicación.

- **Caché de la Base de Datos**

  La base de datos tiene su propia caché asignada pero a diferencia de la caché en la capa de Aplicación no es tan escalable.

- **Caché de segundo nivel**

  Independientemente del ORM que se emplee, en una aplicación Seam se dispone de una caché de segundo nivel de los datos de la base de datos, sin embargo es a menudo defectuosa. Su diseño la hace favorable en ambientes distribuidos en donde múltiples usuarios podrían utilizar los datos de esta caché siempre y cuando las modificaciones en dichos datos sean muy pocas.

- **Caché a nivel de Contexto**

  El contexto de conversación en Seam es una caché del estado conversacional. Los componentes que son inyectados en el contexto de conversación pueden mantener almacenado el estado relacionado con la interacción del usuario actual.
  En particular, el contexto de persistencia actúa como un caché de los datos que han sido leídos en la conversación actual. Seam optimiza las respuestas de sus propios contextos de persistencia en un ambiente distribuido y no hay ningún requisito para mantener la consistencia transaccional con la base de datos.

  Las aplicaciones pueden guardar estado transaccional el componente *cacheProvider* (Proveedor de caché) de Seam y este estado puede ser visible si la caché soporta el trabajo en un ambiente clusterizado. También pueden almacenar un estado no transaccional en el contexto de aplicación de Seam, el cual es invisible para los otros nodos del cluster.

- **Caché a nivel de JSF**

  Seam permite el almacenamiento en caché de los fragmentos recargados de una página JSF. A diferencia del segundo nivel de caché del ORM, esta caché no es automáticamente inhabilitada cuando los datos cambian, así que su invalidación debe ser explícita en el código de la aplicación o establecer las políticas de expiración apropiadas.

2.2.2.16 Hibernate y ORM A pesar de la innegable popularidad de los lenguajes orientados a objetos, las bases de datos relacionales han dominado el mercado por mucho tiempo. Esto ha dado lugar a un desfase entre las tecnologías y herramientas que procesan la información de un sistema y las que la almacenan. Se han intentado crear iniciativas de bases de datos orientadas a objetos pero no han tenido resultados afortunados. Entonces se ha optado por una alternativa diferente: El software de mapeo objeto-relacional (ORM por Object-relationalmapping).

Un software de ORM permite crear una correspondencia entre la información en una base de datos y objetos del lenguaje de programación en que se desee desarrollar una aplicación. Así, se crea la ilusión de una base de datos en memoria que el desarrollador puede manipular mediante técnicas y estructuras de datos típicas de la POO: Clases, métodos, casting, etc.

Los ORM también minimizan el impacto de diferencias radicales entre los objetos y las entidades de una base de datos relacional. Por ejemplo, un objeto es una estructura de datos que no sólo almacena valores, sino que se comporta diferente de acuerdo a los valores que tenga o reciba en un método. El ORM también se encarga de hacer las validaciones y conversiones necesarias entre tipos de datos de la base de datos y la máquina que ejecuta la aplicación, considerando que los tipos de datos primitivos de datos son de diferentes tamaños en diversos sistemas operativos y que las máquinas que albergan la aplicación y la base de datos pueden ser diferentes. El software ORM se vale de las ventajas de la encapsulación y los métodos en POO para ofrecer tal robustez.

Para Java, existe la librería Hibernate, software libre distribuido bajo la Licencia pública general reducida de GNU. Hibernate permite mapear las entidades de una base de datos a clases Java. También maneja todo tipo de cardinalidad de relaciones entre entidades, incluso relaciones reflexivas. Gracias a Hibernate, el código de lenguaje de consultas a introducir se simplifica en cantidad y complejidad de manera considerable, y facilita interactuar con la base de datos como si fuera ésta tan sólo un objeto más en memoria.

Además, como todo ORM, Hibernate se encarga del flujo de datos de la aplicación a la base de datos, efectuando todas las operaciones necesarias para que las claves, los datos, sus tipos y tamaños correspondan a las reglas de la base de datos establecidas en el mapeo, garantizando el éxito de las transacciones.

Hibernate permite el mapeo de entidades mediante archivos descriptores XML o mediante JPA (Java Persistence API). La segunda es la usada en el presente trabajo de grado.

2.2.2.17 JPA La API (ApplicationProgrammers Interface) de persistencia de Java es el esfuerzo del grupo Java EE por brindar una solución integradora de todos los motores de ORM que existen para Java, puesto que Hibernate es sólo uno de una gran variedad. Su funcionalidad se basa en POJOs (Plain Old Java Objects) que no son más que objetos tradicionales de Java. El corazón de la funcionalidad del API se basa en anotaciones (palabras clave que inician con @) que le dan una característica especial a cada miembro

de la clase, correspondiente con la base de datos relacional. Por ejemplo, la anotación @Id en la línea superior de la definición de un miembro, especifica la llave primaria de la entidad. De la misma manera, las anotaciones sirven para definir llaves foráneas, cardinalidad de las relaciones, restricciones del valor de los datos, entre otras funciones. Todo esto reduce signficativamente el código Java a escribir.

Una característica fascinante de JPA, es que permite que se hagan cambios al diseño de la base de datos sin tener que reescribir enteramente las aplicaciones. Para esto JPA introduce:

2.2.2.18 JPQL Java PersistenceQueryLanguage el cual es el lenguaje de consultas que se usará dentro de la aplicación. Con él, el desarrollador jamás se refiere a las entidades directamente al momento de hacer las consultas, sino a los objetos mismos que fueron mapeados. Si el diseño de la BD cambiara, sólo habría que modificar las anotaciones de las entidades. El resto del código quedaría intacto y seguiría funcionando tal y como antes. Esta facilidad permite optimizar las bases de datos cuando se considere necesario, migrar a bases diferentes, o sencillamente corregir diseños defectuosos con un esfuerzo mínimo.

2.2.2.19 Enterprise Architect Enterprise Architect es una herramienta desarrollada por Sparx Systems que ofrece la capacidad de realizar el modelado de un proyecto y apoyar el desarrollo del mismo durante todo su ciclo de vida. Enterprise Architect logra esto usando UML, pero también puede generar código en varios lenguajes de acuerdo a algunos de estos diagramas. Su valor como herramienta radica en la capacidad de permitir a los ingenieros desarrolladores comunicar sus ideas y su visión sobre los proyectos facilitando la administración y la redistribución de esta información.

2.2.2.20 Axure es una aplicación ideal para crear prototipos y especificaciones muy precisas para páginas web. Se trata de una herramienta especializada en la tarea, así que cuenta con todo lo que se puede necesitar para crear los prototipos de forma más eficiente.

CAPÍTULO 3

3 METODOLOGÍA DE DESARROLLO

3.1 Ciclo de vida del proyecto

3.1.1 Análisis de Requerimientos El análisis de requerimientos es la labor que plantea la asignación de software a nivel de sistema y el diseño de programas. Permite la representación de la información y las funciones que pueden ser traducidas en datos, arquitectura y diseño procedimental. La especificación de requerimientos suministra los medios para valorar la calidad de los programas, una vez que se haya construido.

En el análisis de requerimientos se especificará, junto con los directores y usuarios directos de las Escuelas de Sistemas e Informática y a su vez la Escuela de Matemáticas las cuales serán las Escuelas piloto, la función y comportamiento que deberán tener los programas a desarrollarse en el transcurso del proyecto, se indicará la interfaz con otros elementos del sistema y se establecerán los estándares de diseño que debe cumplir éste.

Es en esta etapa se hicieron reuniones periódicas con los funcionarios interesados en el software, para que sean ellos los que definan las características del software que se desea desarrollar y que se adapte plenamente a las exigencias de las Escuelas del campus principal de la Universidad Industrial de Santander.

También se hizo un seguimiento continuo a los prototipos desarrollados para acercarlos cada vez más a los requerimientos y para que la DSI y las Escuelas del campus principal tuvieran un acercamiento al producto final y pudieran especificar cualquier requisito que no estuviera siendo cumplido o que pudiera refinarse.

3.1.2 Diseño El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos: estructura de datos, arquitectura de software, representaciones de interfaz y detalle procedimental (algoritmo).
En esta etapa de diseño se hace una traducción de los requisitos a una representación del software donde se pueda evaluar su calidad antes de que comience la codificación.

El diseño se efectuará, mediante modelos UML (Lenguaje de Modelado Unificado) que incluirá los diagramas que han sido seleccionados dentro de los estándares de desarrollo de software utilizados en la División de Servicios de Información, que son: de casos de uso, de clases y de secuencia, utilizando la herramienta de modelado Enterprise Architect.

3.1.3 Implementación de la Aplicación En esta etapa se procede a generar el software que se ha diseñado teniendo en cuenta los parámetros establecidos por la División de Servicios de Información, en cuanto a los estándares técnicos y de calidad que caracterizan las aplicaciones que son generadas para el servicio de la Universidad, teniendo como base la Arquitectura de Desarrollo de Aplicaciones de JAVA EE5 y la plataforma Informix como motor de base de datos.

3.1.4 Pruebas de Software Son los procesos que permiten verificar la calidad de un producto software y el cumplimiento de los requerimientos establecidos en la fase de análisis de requerimientos.

Las pruebas de software se integran dentro de las diferentes fases del ciclo de desarrollo del software establecidas en la ingeniería de software.

Una vez terminada la codificación, comienzan las pruebas, proceso utilizado para identificar posibles fallos de implementación, calidad, o usabilidad de un programa. Las pruebas se centran en los procesos lógicos internos del software, asegurando que todas las sentencias sean probadas y asegurar que la entrada definida produce los resultados esperados.

Las pruebas serán de carácter permanente a lo largo del desarrollo del proyecto por parte del equipo de trabajo, y habrá un periodo de tiempo para que los usuarios finales interactúen con la aplicación y detecten posibles ajustes.

3.1.5 Ajustes Después de las Pruebas, cada uno de los errores detectados o las observaciones hechas por los usuarios debidamente analizadas, deben ser tenidos en cuenta para ajustar el sistema, de tal manera que se adapte plenamente a las necesidades de los mismos.

También estos se harán permanentemente a lo largo del desarrollo del proyecto a la par con las pruebas, en la medida en que se detecten errores o inconsistencias en el sistema que se ha desarrollado.

3.2 Metodología de Desarrollo

3.2.1 Modelo de construcción por prototipos Se eligió esta metodología debido a que es muy frecuente que los usuarios que están solicitando el sistema, definan un conjunto de objetivos generales para el software, pero no identifican los requisitos detallados de entrada, proceso o salida.

En otros casos, el responsable del desarrollo del software puede no estar seguro de la eficacia de un algoritmo, o no haber comprendido plenamente el requerimiento del usuario.

Para éstas y otras muchas situaciones, un *paradigma de construcción por prototipos* puede ofrecer el mejor enfoque, ya que la entrega de prototipos, que hacen parte integral del proyecto en su conjunto, permitirán la corrección temprana de errores o la redefinición del sistema en caso de ser necesario, y los prototipos funcionales permitirán la familiarización del usuario con el sistema que se está desarrollando.

A continuación se observa la estructura del modelo:

Figura 16. Modelo Construcción Por Prototipos

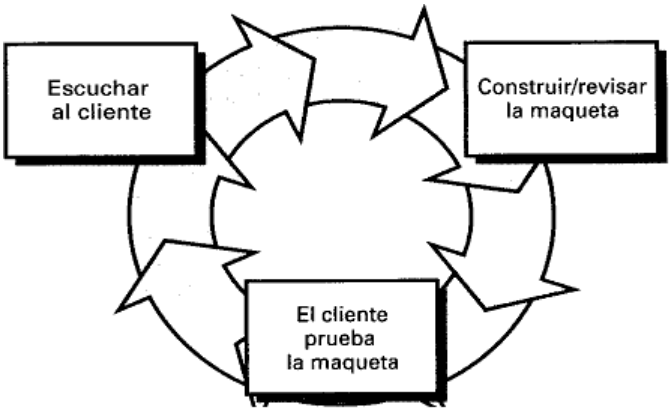
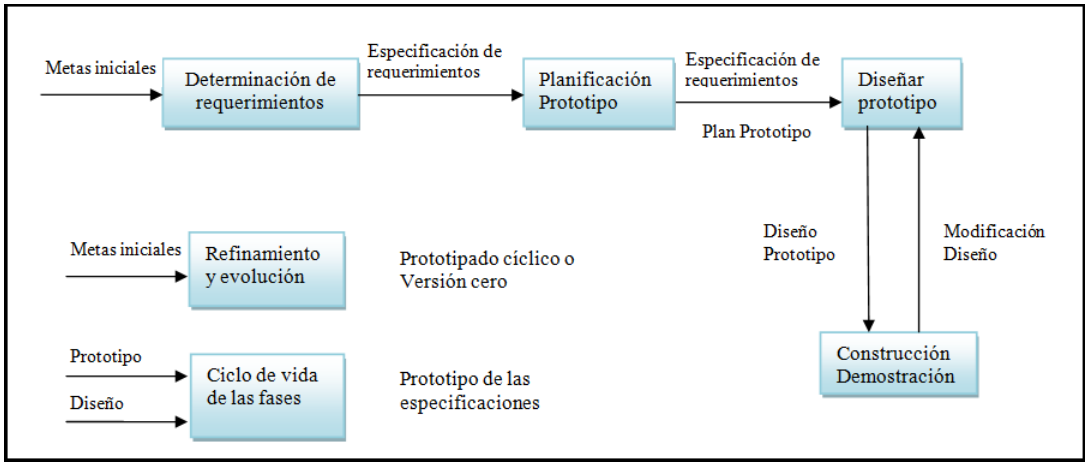


Figura 17. Estructura metodología



Esta metodología es viable para el desarrollo del proyecto debido a que:

- En la creación de los prototipos iniciales se debe trabajar con unas ideas aproximadas de lo que desean las Escuelas del campus principal, para presentar un producto o prototipo inicial, el cual puede evolucionar y refinarse dando como resultado un prototipo más maduro, que cumpla con todos los requerimientos de los usuarios.

- Con el uso del modelo de prototipos se da la facilidad de mejorar, de manera temprana los prototipos, teniendo en cuenta las sugerencias del usuario solicitante del proyecto, de tal forma que se cubran a cabalidad sus requerimientos.

- En este sistema de desarrollo se debe validar la versión actual del prototipo, para proceder a generar una nueva versión que contemple los nuevos requerimientos, con el fin de evitar retrocesos en el proceso de desarrollo.

3.3 APLICACIÓN DE LA METODOLOGIA

3.3.1 Levantamiento requisitos

Figura 18. Ejemplo de la estructura de los sistemas Web UIS

Funcionalidad del producto

- El sistema debe ser una herramienta que ayude a los Directores de Escuela en la realización de los horarios por lo tanto debe contemplar unos parámetros para su buen funcionamiento entre ellos encontramos los siguientes.
    - El sistema debe detectar cruces horarios de los docentes, es decir un docente en una franja horario solo puede tener un curso excepto sean asignaturas magistrales.

    - El sistema debe detectar los cruces de cursos esto significa que dos cursos o mas no pueden estar en una misma aula en una franja horaria excepto sean asignaturas magistrales.

    - El sistema debe informar cuando dos o más cursos de un mismo nivel de un plan de estudios se cruzan.

    - El sistema debe dar informes entre ellos encontramos los siguientes:
      De la programación de cursos, horario por salón, horario por profesor, horario por nivel, horario por asignatura, horario por curso.

    - El sistema debe permitir aumentar o disminuir la capacidad de los cursos.

    - El sistema debe permitir eliminar cursos, crear cursos, modificar franjas horarias de los cursos.

    - El sistema debe tener una interfaz intuitiva que guie al usuario por el uso de cada una de las herramientas.

    - El sistema debe estar en la plataforma web de la Universidad.

    - El sistema debe realizarse bajo los estándares de DSI.

    - El sistema permite asignar docentes a los cursos.

    - El sistema asigna salones especiales y regulares de la escuela.

- El sistema permite que el docente ingrese las horas en las cuales puede impartir materias y las materias que está en capacidad de dictar.

- El sistema usa como base el horario de semestres anteriores y de ahí se van haciendo todas las modificaciones para el actual.

- El sistema debe permitir asignar aulas especiales y regulares de la escuela a los cursos.

Características de los usuarios

Los principales usuarios de este sistema son los Directores de Escuela, los cuales son docentes planta por lo tanto tienen un nivel de educación superior, un nivel de manejo del computador medio.

Otros de los usuarios de este sistema son las secretarias de las distintas escuelas, las cuales cuentan con una educación intermedia y un nivel medio en el manejo del computador.

Los administradores van a ser Dirección de Admisiones y Registro Académico, los cuales cuentas con un nivel educativo alto y a su vez cuentan con experiencia técnica.

Restricciones

- Un profesor no puede estar en dos cursos en un mismo horario excepto este dictando una asignatura magistral.

- Materias de un mismo nivel no deben estar en un mismo horario.

- Dos profesores no pueden dictar en la misma aula a la misma hora excepto sean asignaturas múltiple profesor.

La metodología que se emplea es prototipado evolutivo, ésta es la utilizada por la División de Sistemas de Información de la Universidad. El lenguaje de programación que se debe utilizar es Java 5 y el IDE es Jboss Developer Studio, la base de datos se encuentra en Informix.

Suposiciones y dependencias

Si se cambia la política de la Universidad en cuestión al lenguaje de programación que emplean, se debe amoldar el software desarrollado, si la Universidad no permite usar el insumo del semestre anterior el sistema debe cambiar su forma de planteamiento.

Evolución previsible del sistema

Un software a futuro puede aparte de plantear la programación de horarios usando como insumo la programación de semestres anteriores, podría realizar una programación que comience casi de cero o la genere según unos parámetros dados de forma automática.

Requisitos específicos

Tabla 1. Requisitos específicos

Número de requisito	RF 1
Nombre de requisito	Editar Docentes
Tipo	Requisito
Descripción del requisito	El sistema permitirá la asignación de éstos a determinados cursos.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 2
Nombre de requisito	Editar Cursos
Tipo	Requisito
Descripción del requisito	El sistema permitirá crear, modificar o eliminar cursos. Aquí se podrán editar características como la capacidad de los cursos, franja horaria, docente asignado, salón asignado.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 3
Nombre de requisito	Asignar curso
Tipo	Requisito
Descripción del requisito	El sistema permitirá asignar un curso a un salón en particular.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 4
Nombre de requisito	Cruce curso
Tipo	Restricción
Descripción del requisito	El sistema debe detectar si se intenta asignar un curso a un salón en una franja horaria que ya posea otra asignación de curso excepto sean asignaturas magistrales.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 5
Nombre de requisito	Cruce Docente
Tipo	Restricción
Descripción del requisito	El sistema debe detectar si se intenta asignar un docente a un curso que tenga una franja horaria en la que el docente ya tiene asignado otro curso excepto este vaya a dictar una asignatura magistral.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 6
Nombre de requisito	Cruce Nivel
Tipo	Restricción
Descripción del requisito	El sistema debe emitir una alerta si se intenta asignar cursos de un mismo nivel en una misma franja horaria.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 7
Nombre de requisito	Login Usuario
Tipo	Requisito
Descripción del requisito	Para poder realizar cualquier tipo de modificación es necesario ingresar al sistema mediante un login, el cual dependiendo del tipo de usuario le mostrará las opciones a las cuales tiene acceso.

Prioridad del requisito	Alta/Esencial

Número de requisito	RF 8
Nombre de requisito	Ingreso Carga Horaria docente
Tipo	Requisito
Descripción del requisito	El sistema debe permitir que los docentes hagan login en el sistema para que éstos ingresen la disponibilidad y los cursos que pueden dictar en un semestre.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 9
Nombre de requisito	Generar Informes
Tipo	Requisito
Descripción del requisito	El sistema debe generar informes como: Horario por nivel, horario por docente, horario por salón, programación general de horario de la escuela, cursos sin salón asignado, asignación de cursos a salones.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 10
Nombre de requisito	Advertencias
Tipo	Requisito
Descripción del requisito	El sistema debe mostrar las siguientes advertencias tales como: si existen cruces en materias del mismo nivel, cruces en horarios de docentes, si materias que no son requisitos se cruzan en franja horaria entre otras.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 11
Nombre de requisito	Disponibilidad aulas
Tipo	Requisito
Descripción del requisito	El sistema mostrará la ocupación de las aulas para la escuela que realiza la programación, así como la disponibilidad de las aulas de la universidad en una franja horaria en particular.
Prioridad del requisito	Alta/Esencial

Número de requisito	RF 12
Nombre de requisito	Insumo programación anterior
Tipo	Requisito

Descripción del requisito	El sistema deberá permitir utilizar la programación de semestres anteriores y modificar está para crear la programación del próximo semestre.
Prioridad del requisito	Alta/Esencial

Requisitos comunes de los interfaces

ENTRADAS

- La programación de cursos del semestre anterior por escuela.
- La disponibilidad de los profesores y las materias que pueden dictar.
- Las franjas horarias de un aula.

SALIDAS

- La programación del semestre que inicia que incluye cursos con aulas (las correspondientes a la escuela).
- La programación por docente
- La programación por asignatura.
- La programación por salón.
- La programación por nivel.

3.3.2 Diagramas UML En la División de Servicios de Información (DSI) se han establecido los estándares concernientes al diseño de sistemas, el cual debe ser desarrollado por módulos y debe ceñirse al estándar definido por el Lenguaje Unificado de Modelado 2.1 (UML).

El diseño de un módulo, desarrollado en la DSI, debe contar con los siguientes diagramas:

- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Secuencia

Teniendo en cuenta la dimensión de los diagramas UML elaborados para el proyecto, se ha tomado un ejemplo de cada uno de ellos para ser descritos de forma detallada.

3.3.3 Diagrama de Casos de Uso Según los estándares definidos por la División de Servicios de Información UIS, por cada módulo del sistema que vaya a ser implementado debe realizarse un diagrama de casos de uso.
El caso de uso seleccionado como ejemplo, corresponde al módulo de gestionar programación.

En este módulo se llevan a cabo las siguientes acciones:

- Creación, modificación y eliminación de cursos programados.
- Copia de la programación de semestres anteriores.

Las siguientes tablas describen los casos de uso más importantes para el proceso.

Tabla 2. Descripción caso uso Gestionar cursos

Gestionar Cursos	
Actor	Usuario
Propósito	El caso de uso se encarga de gestionar los cursos, requeridos para la realización de la programación de grupos de un semestre académico.
Resumen	La gestión de cursos contempla: • Creación de Cursos • Consulta de Cursos
Flujo Principal	**GESTIÓN DE CURSOS EXITOSA** El usuario puede elegir alguna de las siguientes opciones: 1. Consultar un Curso. 2. Crear un Curso. **SELECCIÓN**

	El usuario: Decide realizar una consulta el cual invoca el caso de uso Consultar Curso después de este caso de uso puede activar alguno de los siguientes casos de uso Modificar Curso Eliminar Curso. Elige crear un curso el cual dispara el caso de uso Crear Curso.
Precondición	El usuario debe: Estar autenticado Seleccionar una escuela.

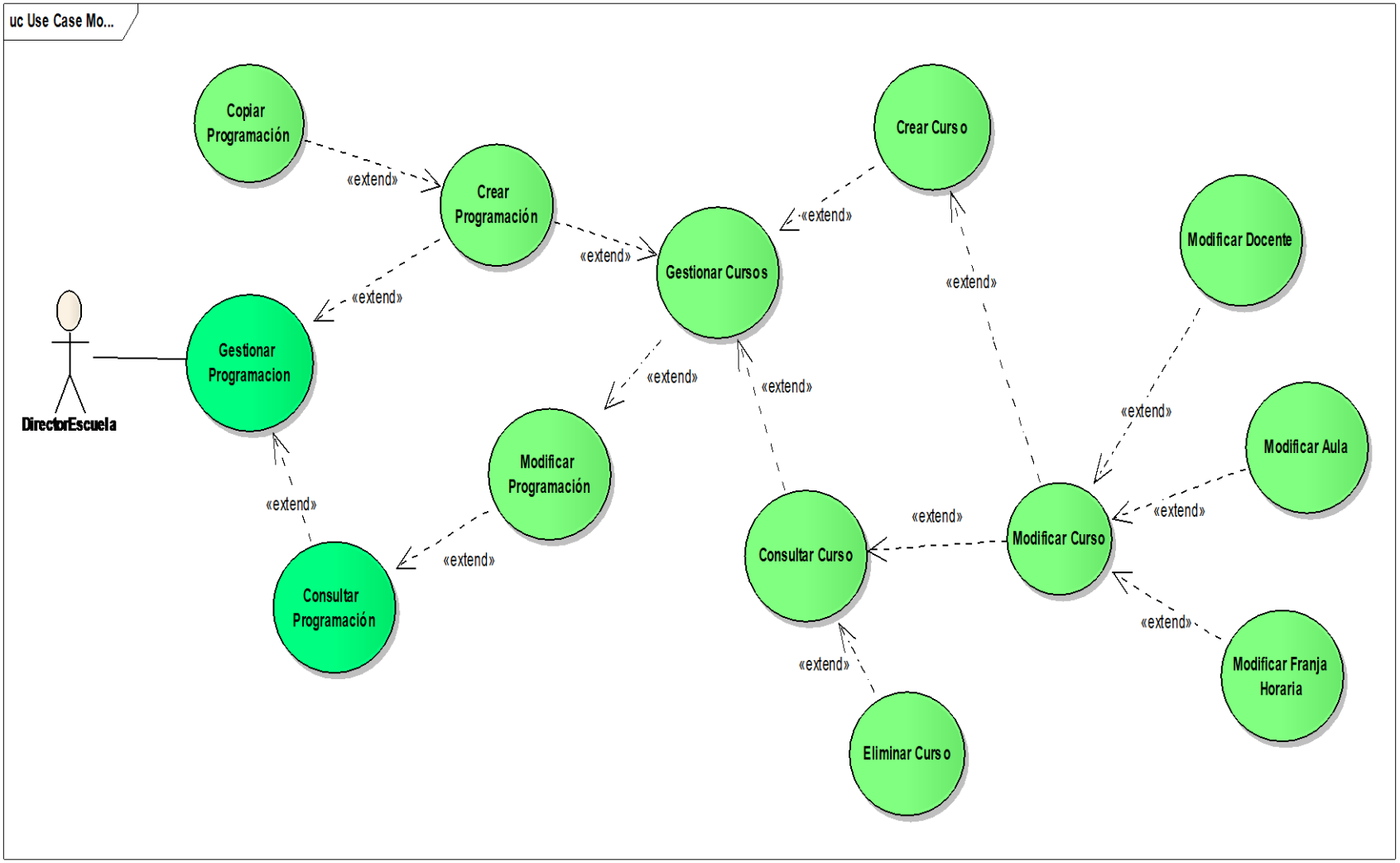


Figura 19. Diagrama Casos de Uso Gestionar Programación

Copiar Programación	
Actor	Usuario
Propósito	La copia de la programación tiene como intención facilitar la labor a los usuarios, permitiéndole usar la programación realizada para semestres anteriores al cual se está programando.
Resumen	El usuario puede comenzar a programar los cursos para el semestre a programar partiendo de la base de la programación realizada en semestres anteriores.
Flujo Principal	**COPIA DE LA PROGRAMACIÓN EXITOSA** El usuario realiza la siguiente ruta : **Ruta Básica:** Elige “Copiar Programación” lo que implica: 1.Seleccionar la facultad 2. Desplegar en pantalla el listado de todas las escuelas de la Universidad d que pertenezcan a la facultad seleccionada, de esta lista se selecciona la escuela. 3. Seleccionar un periodo y un año del cual se va a copiar la programación. 4. Luego de seleccionar todos los datos anteriores se muestra estadísticas de los datos copiados y se procede a ubicar al usuario en la interfaz de creación de cursos. **Ruta Alterna:** El usuario entra en la ruta alterna si: Ha elegido el paso 1 de la ruta básica y no se muestra la información de la consulta entonces se despliega en pantalla un mensaje de advertencia informándole que hubo un error en la consulta. Escoge el paso 2 de la ruta básica y no se muestra la información de

	la consulta entonces se despliega en pantalla un mensaje de advertencia informándole que hubo un error en la consulta. Selecciona el paso 3 de la ruta básica y no se muestra la información de la consulta entonces se despliega en pantalla un mensaje de advertencia informándole que hubo un error en la consulta. El error en la ruta alterna está asociado a problemas de conectividad con la Base de Datos o inconvenientes con el servidor.
Precondición	En el modulo seleccionado se deben tener identificados: El usuario debe estar logueado. Debe haber seleccionado un año y periodo para realizar la copia de la programación. Debe haber seleccionado una escuela.

Tabla 3. Descripción caso uso Copiar programación

Crear Programación	
Actor	Usuario
Propósito	Realizar la gestión de los cursos a programar para el semestre que inicia.
Resumen	El usuario visualiza todos los cursos creados o copiados de programaciones pasadas.
Flujo Principal	**CREACION EXITOSAMENTE** El usuario lleva a cabo el siguiente camino: **Ruta Básica:** Elige “Crear Programación” lo que implica: El usuario visualiza el periodo y el año del cual se va a realizar la

	creación de la programación. Se procede a realizar la creación de cursos para el semestre que se está programando. **Ruta Alterna:** El usuario entra en la ruta alterna si: Ha elegido el paso 1 de la ruta básica y no se muestra la información de la consulta en donde se muestra en un mensaje de advertencia informándole que hubo un error en la consulta. Escoge el paso 2 de la ruta básica y no se muestra la información de la consulta en donde se muestra en pantalla un mensaje de advertencia informándole que hubo un error en la consulta. Si en el paso 3 de la ruta básica no se muestra la información de la consulta en donde se muestra en pantalla un mensaje de advertencia informándole que hubo un error en la consulta. El error en la ruta alterna está asociado a problemas de conectividad con la Base de Datos o inconvenientes con el servidor. **EXCEPCIONES EN LA CREACIÓN** La información del formulario es incompleta. El sistema dispara avisos indicando el problema. Existen problemas de conexión ya sea a nivel de base de datos o con el servidor. El sistema recarga la pantalla para mostrar el mensaje de excepción en la transacción.
Precondición	El usuario debe: Estar autenticado.

Tabla 4. Descripción caso uso Crear programación

3.3.4 Diagramas de Clases El diagrama de clases seleccionado como ejemplo, corresponde al Sistema de Horarios UIS, dicho diagrama se aprecia en la figura 20.

Clase Ocupación Docente: Esta clase muestra los datos de la disponibilidad con que cuenta un docente para dictar clases.

Tabla 5. Atributos de la Clase Ocupación Docente

Nombre	Tipo de dato	Descripción
año	Smallint	Corresponde al año para el que se va a realizar la programación de Horarios.
periodo	Smallint	Corresponde al periodo para el que se le va a realizar la programación de Horarios.
documento_id	Decimal	Es el número del documento de identificación del docente que está ingresando su disponibilidad
tipo_doc_id	Char	Es el tipo de documento de identificación del docente que está ingresando su disponibilidad.
hora_inicial	Smallint	Es la hora a la cual empiezan a estar disponibles los docentes.
dia	Smallint	Es el día en el q se encuentran disponibles los docentes para dictar las asignaturas.
duración	Smallint	Es el tiempo que se encuentran disponibles en un día para dictar clases.

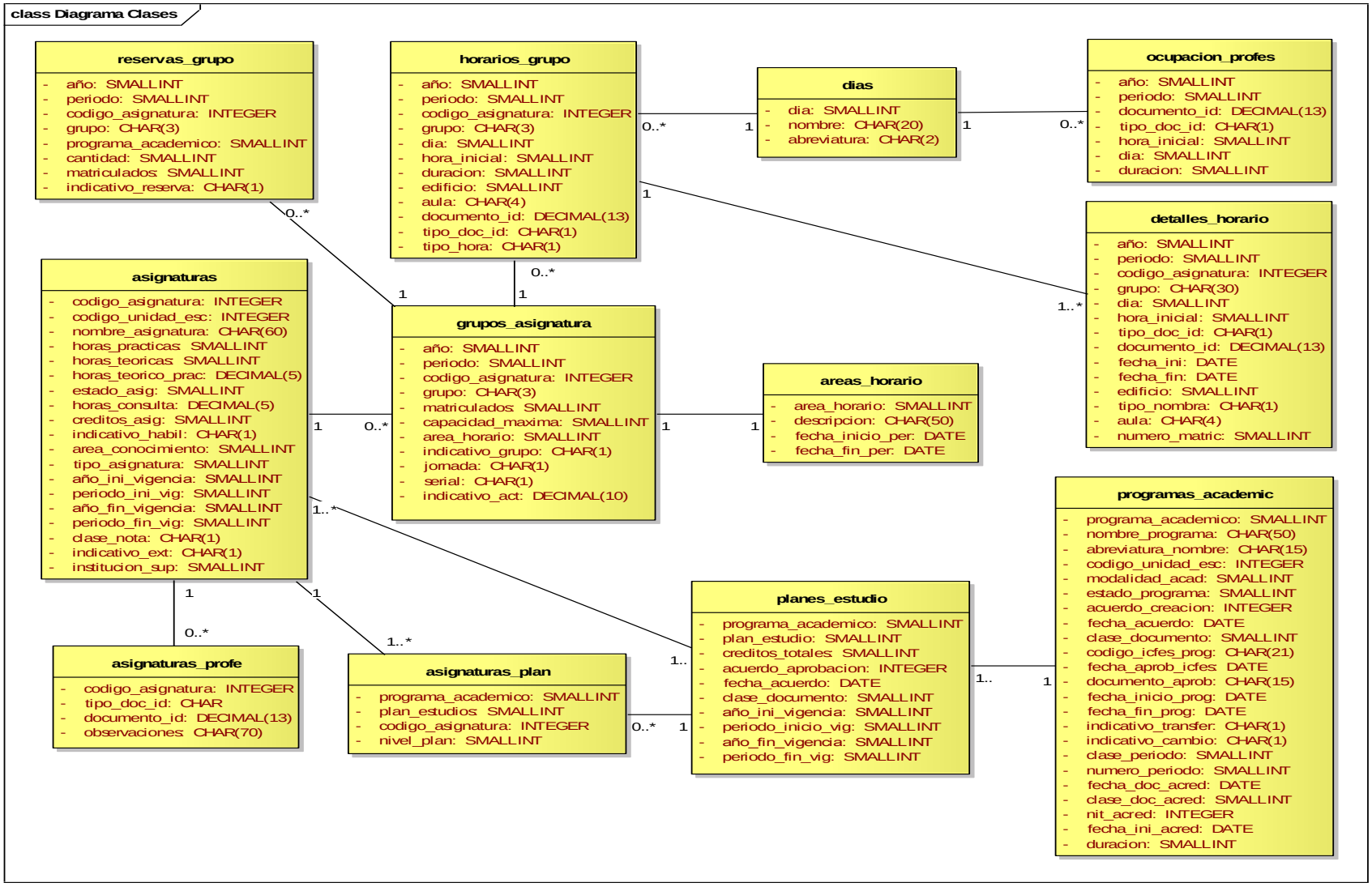


Figura 20. Diagrama de Clases Académico utilizadas para el sistema de Programación Horarios

Clase Asignaturas_Profe*:* Esta entidad contiene información acerca de las asignaturas que un docente se encuentra en la capacidad de dictar.

Tabla 6. Atributos de la clase Asignaturas Profe

Nombre	Tipo de datos	Descripción
codigo_asignatura	Integer	Es el código de las asignaturas que un docente se encuentra en capacidad para dictar.
tipo_doc_id	Char	Es el tipo de documento de identificación del docente que está ingresando las asignaturas que está en capacidad de dictar.
documento_id	Decimal	Es el número del documento de identificación del docente que está ingresando las asignaturas que está en capacidad de dictar.
observaciones	Char	Anotaciones que el docente considere importantes realizar para que el director de escuela tenga en cuenta en el momento de la programación de cursos.

Clase Horarios Grupo: Esta clase muestra los datos de los horarios de un grupo

Tabla 7. Atributos de la Clase Horarios Grupo

Nombre	Tipo de dato	Descripción
año	Smallint	Corresponde al año que se está programando el grupo

periodo	Smallint	Corresponde al periodo académico para el cual se está programado.
Documento_id	Decimal	Es el número del documento de identificación del docente que está a cargo para este grupo en ese horario.
Tipo_documento	Char	Es el tipo de documento de identificación del docente que está a cargo para ese horario grupo.
hora_inicial	Smallint	Es la hora a la cual empieza el horario del grupo programado.
dia	Smallint	Es el día en el que se va a impartir la clase para ese horario grupo.
duración	Smallint	Es el tiempo que va a demorar la clase para ese horario grupo en particular
edificio	Smallint	Es el edificio al cual pertenece el aula en la que se va a impartir la asignatura para ese horario grupo
aula	Char	Es el número del aula donde se va a impartir ese horario grupo.

3.3.3.5 Diagrama de Secuencias De acuerdo a los estándares de la DSI se elaboran los diagramas de secuencia que sean considerados de mayor importancia.

En los diagramas de secuencias que se presenta a continuación se da una visión más amplia acerca del funcionamiento y la interacción del sistema con el usuario, para los siguientes casos de usos: Creación de cursos ver figura 21, Copiar Programación cursos ver figura 23, Consultar Horarios Cursos ver figura 22.

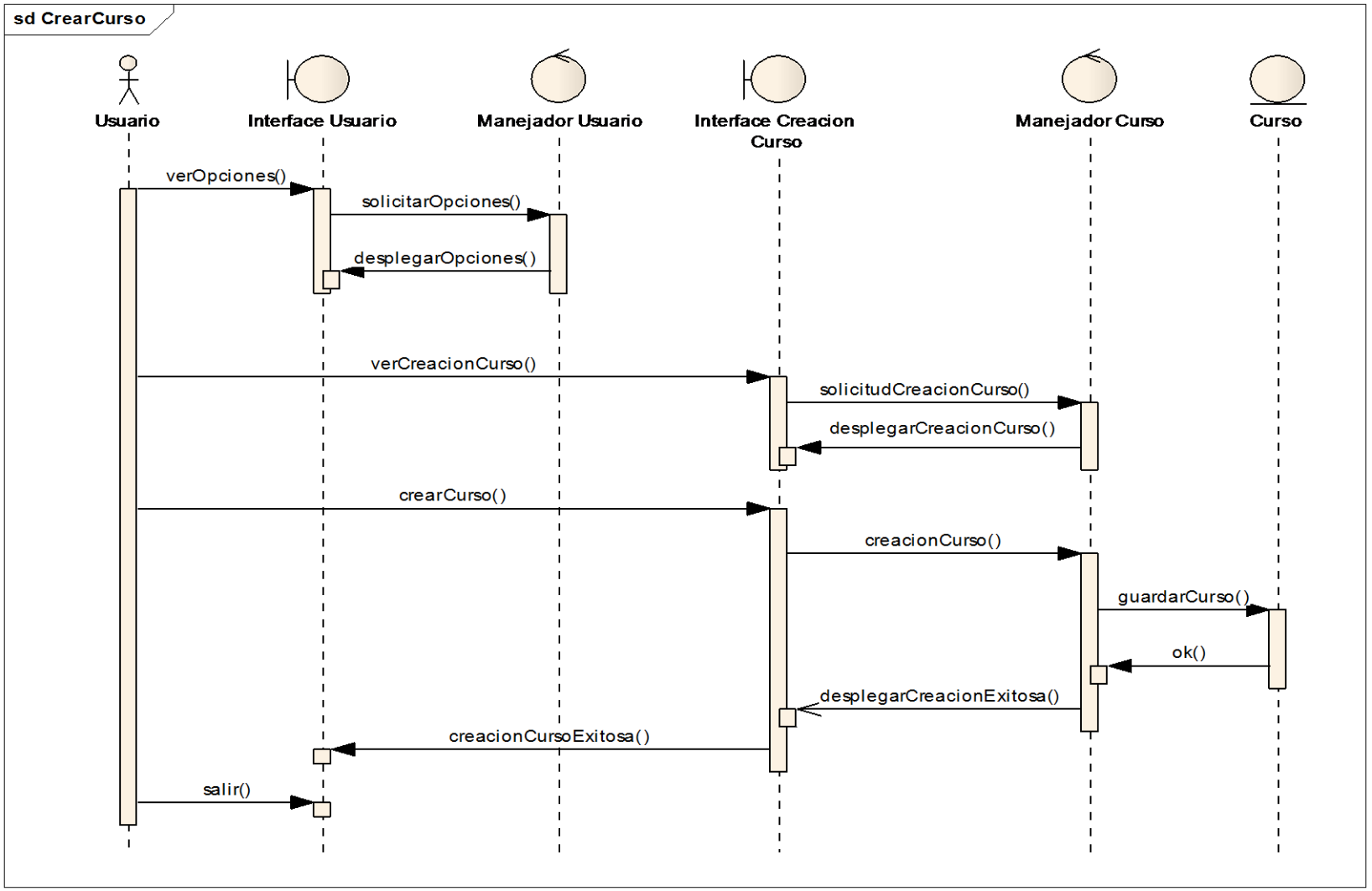


Figura 21. Diagrama de Secuencia para la creación de cursos

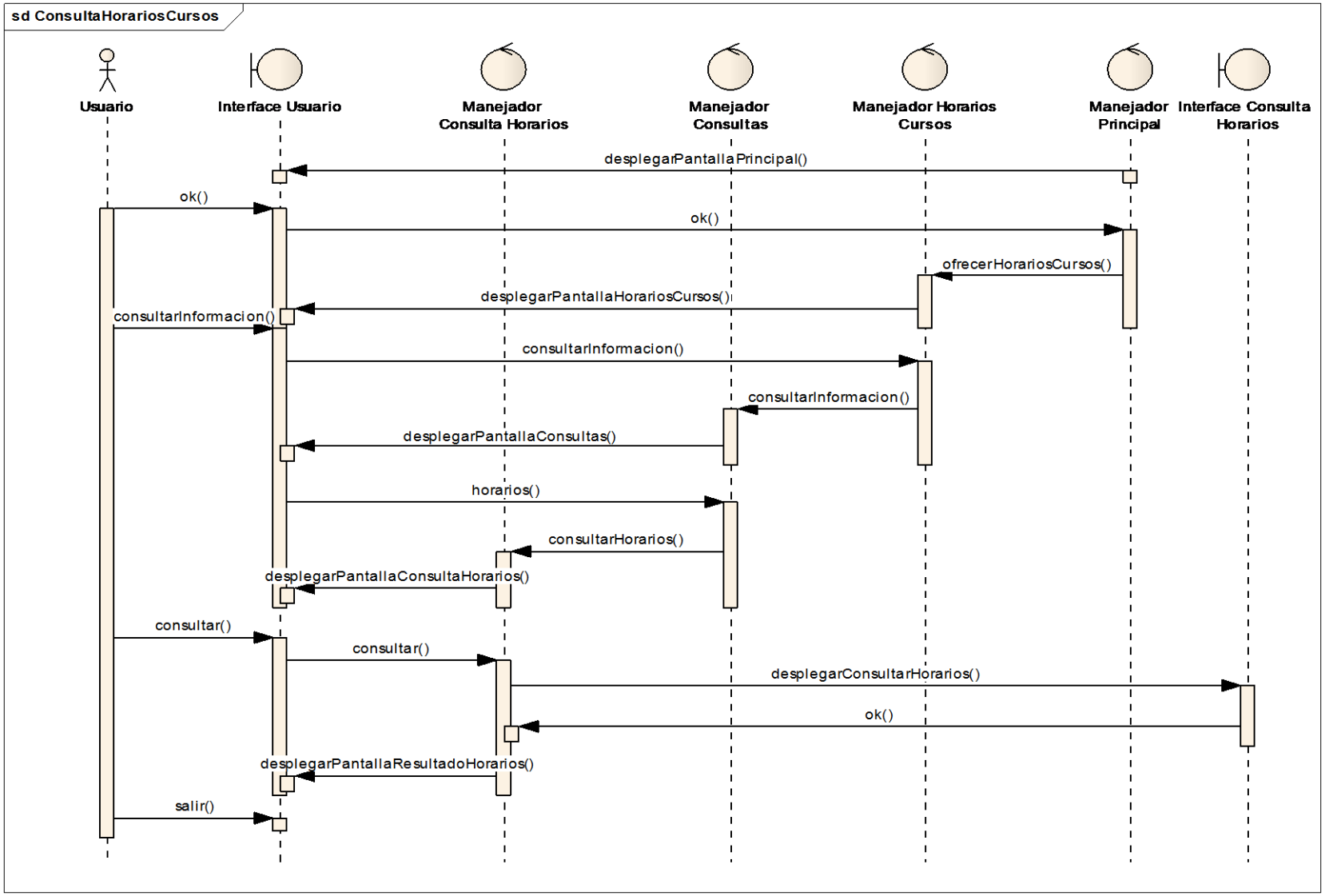


Figura 22. Diagrama de Secuencia para la consulta de los horarios de un curso

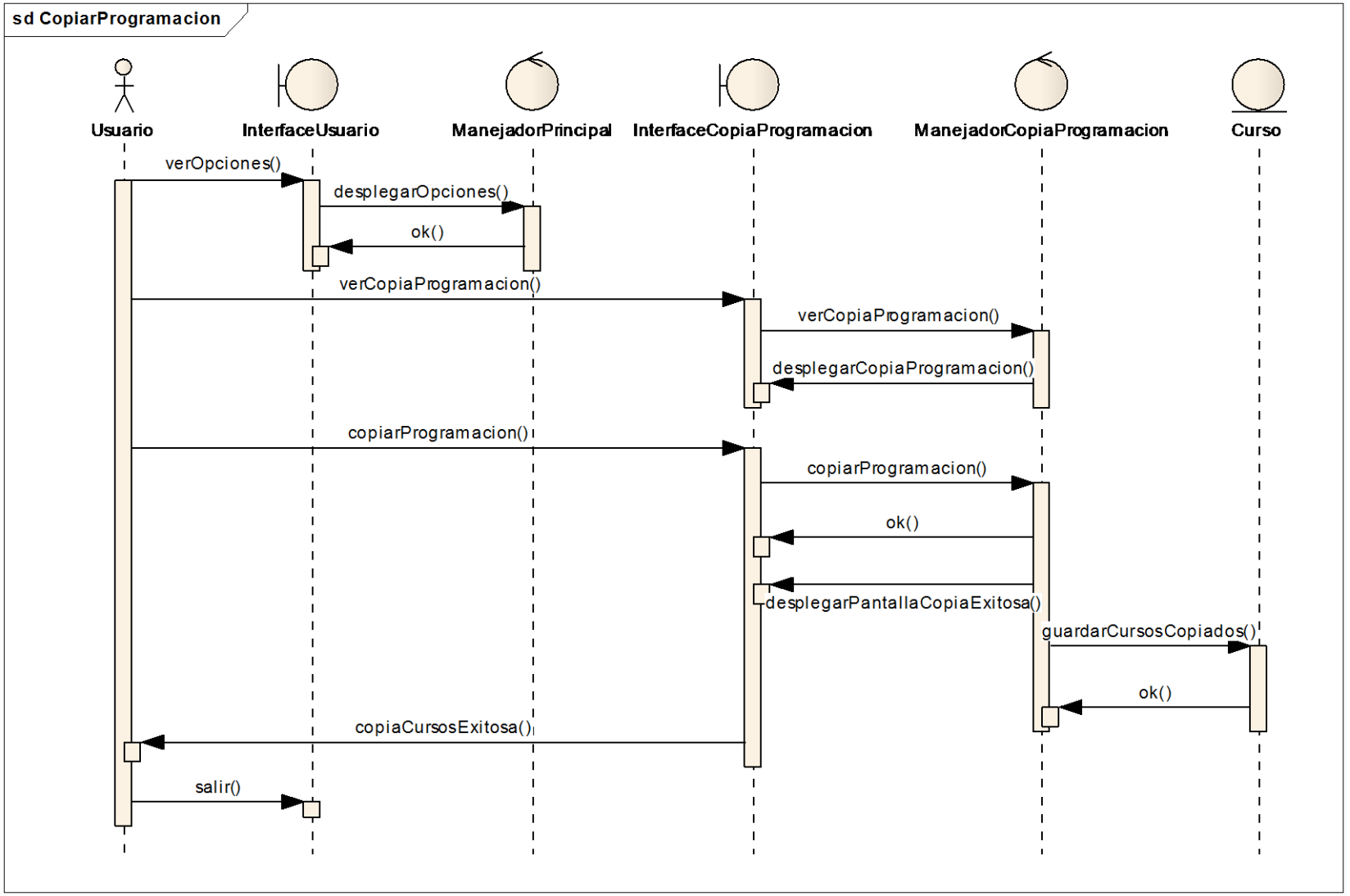


Figura 23. Diagrama de Secuencia para la copia de la programación

3.4 Prototipos

3.4.1 Primer prototipo Para la realización del primer prototipo se realizaron reuniones con los interesados de las escuelas piloto y las demás dependencias que participan activamente del proceso.

En éstos encuentros se escuchó y analizó cada uno de los requerimientos deseados para el sistema y se fue estructurando un prototipo no funcional que serviría como carta de navegación en el transcurso del desarrollo del sistema.

El prototipo no funcional se creó como una página web básica sin ningún tipo de programación de fondo, que mostraría gráficamente el esquema de navegación del sistema, de esta manera se definieron detalles de alta importancia tales como la información que el usuario quiere ver, los datos que el sistema ofrece como base para comenzar su labor y los datos a solicitar a cada uno de los usuarios que interactuarían con el sistema.

Se manejo un esquema cíclico de perfeccionamiento y refinamiento de tal manera que se llegara a un nivel de acercamiento detallado de lo que quiere alcanzar el usuario con el sistema, de esta manera se logra que el proceso de refinamiento del prototipo funcional sea mínimo agilizando de esta manera el ciclo de vida del proyecto.

El tiempo destinado a esta etapa fue bastante amplio, ya que del resultado obtenido dependería la eficiencia del desarrollo y programación del sistema.

Durante el transcurso de la construcción del prototipo se definió todo el diseño integrado por los diagramas de casos de uso, de clases, de datos y de secuencia.

Interfaz resultado del prototipo inicial

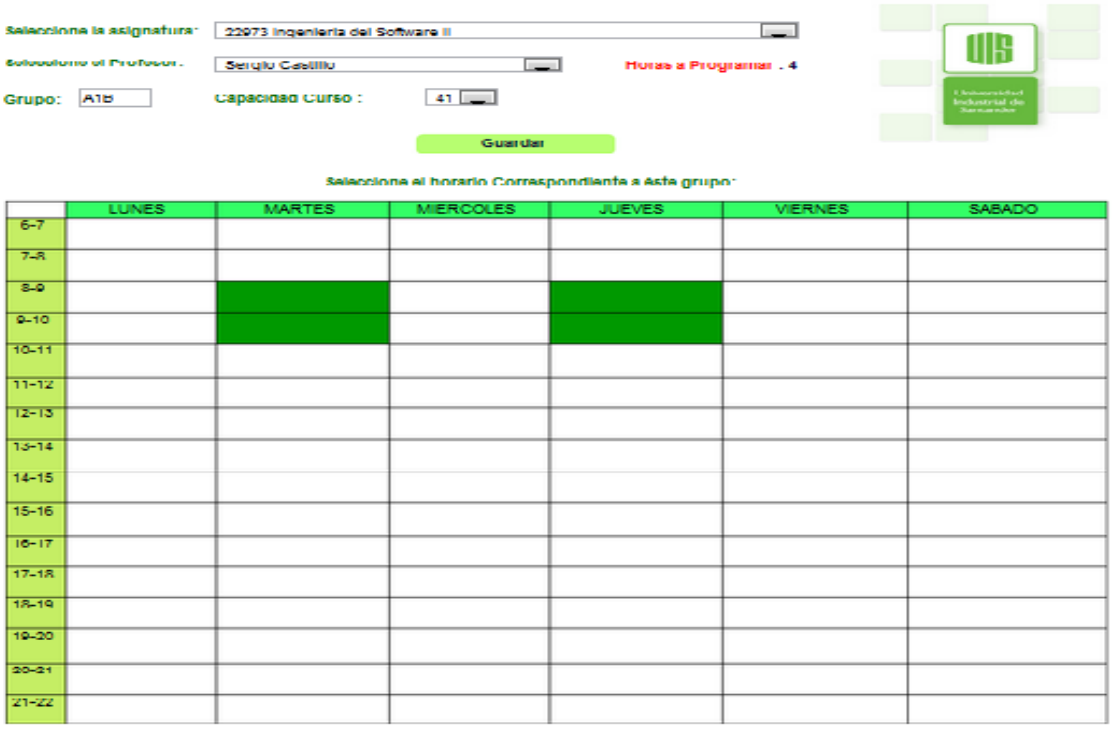
Se presenta a continuación apartes de la interfaz resultante de la construcción del prototipo inicial (no funcional).

Creación de un curso

La interfaz que se observa en la figura muestra la primera aproximación a la creación de un curso en particular.

Para la realización de la programación de asignaturas se debe llevar a cabo la creación de cursos, en este proceso se selecciona la asignatura que se desea programar, el nombre que se le va a dar al grupo, la capacidad que va a tener el grupo, el docente a cargo y a su vez el usuario selecciona los respectivos horarios en los que va a programar el respectivo curso.

Figura 24. Vista de Creación Curso. Prototipo Inicial



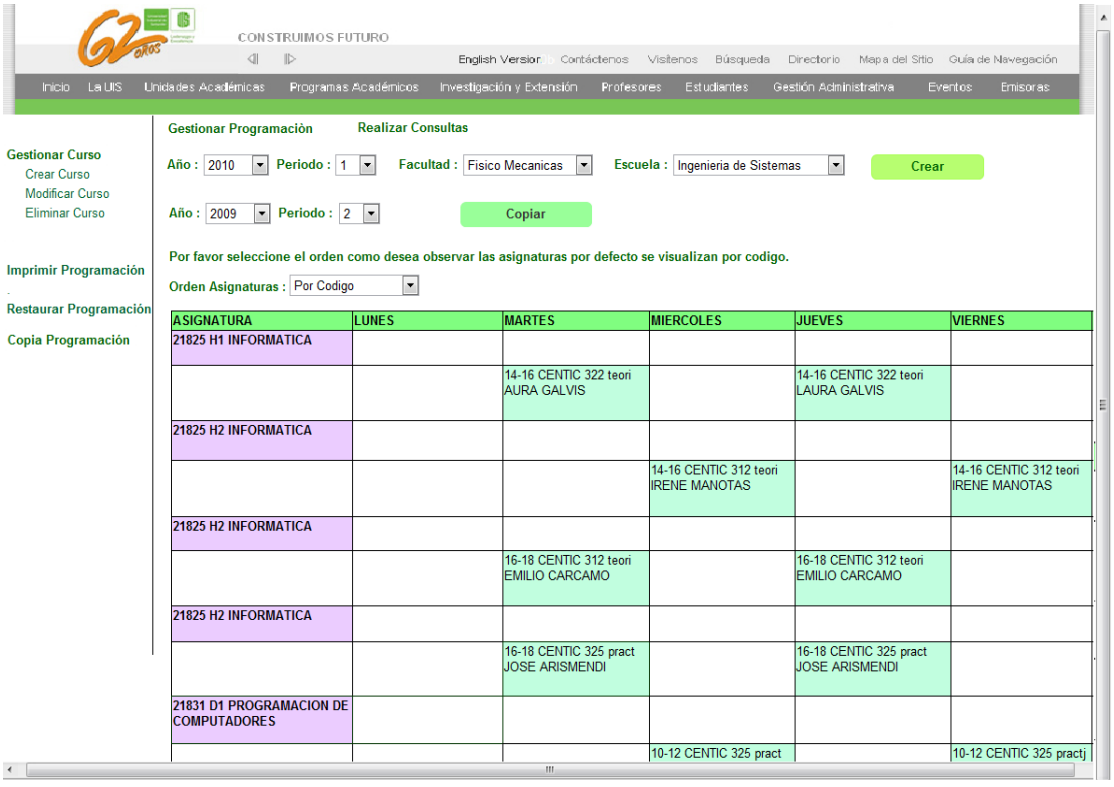
Gestión de la programación

Esta es la vista de la programación de cursos para el semestre que se está programando y a su vez realización de la copia de la programación de semestres anteriores al que se está programando.

En el prototipo inicial se planteó manejar la creación de cursos y la copia de la programación realizada de semestres anteriores en una misma vista, en la versión final ésta interfaz fue modificada para que la creación de la programación y la copia fueran ítems diferentes en el menú, con el propósito hacer más agradable y sencilla la interfaz al usuario final; más adelante podremos apreciar los cambios realizados en las paginas involucradas.

Adicionalmente se pueden realizar filtros por el código de la asignatura.

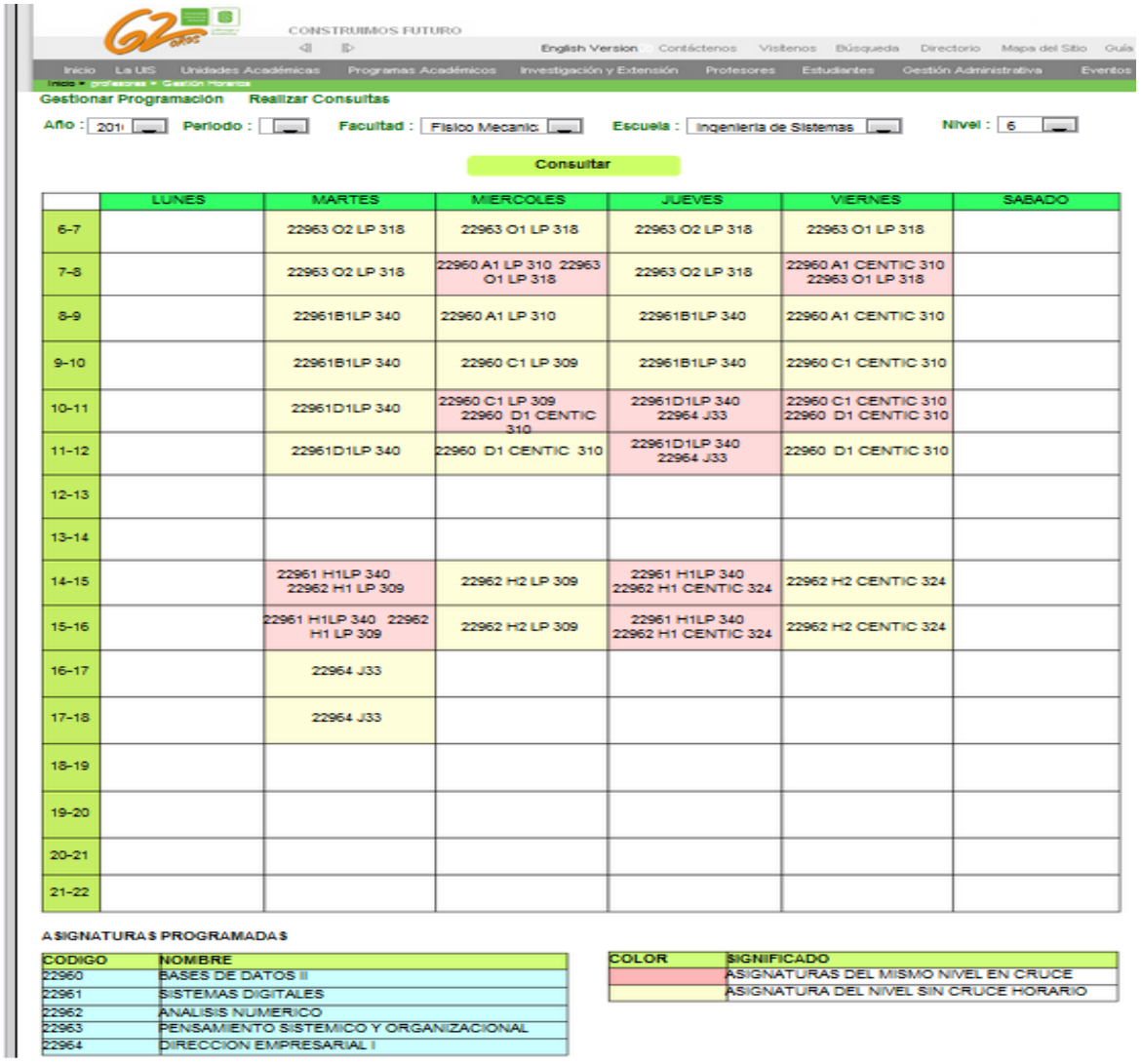
Figura 25. Vista de Creación Programación Cursos y copia. Prototipo Inicial



Consulta programación por nivel

Esta es la vista de la consulta de la programación por nivel para un programa académico y un nivel de un plan de estudio específico, se pueden observar las interferencias que pueden ocurrir en el momento de programar los cursos, estas son identificadas con color rosado para que sea de fácil reconocimiento para el usuario.

Figura 26. Vista Consulta Programación cursos por nivel. Prototipo Inicial



3.4.2 Prototipo Final Al término del desarrollo del módulo de Horarios, se obtuvo un prototipo que se adapta plenamente a las características que se han definido desde el comienzo del desarrollo del proyecto.

Tomando como punto de partida el prototipo no funcional, se logró consolidar el producto definitivo, y luego de un proceso de refinamiento por parte del cliente, se reemplazará el módulo actual de administración de los horarios por el sistema desarrollado.

El producto cumple con los parámetros que se definieron en el documento de requerimientos que se menciona en el apartado 3.3.1 de éste documento, y después de un proceso de evaluación por parte del cliente se realizaron las respectivas modificaciones o sugerencias al producto final.

A continuación se describirá de forma general las funcionalidades más importantes, propias del sistema de información de Horarios de la Universidad Industrial de Santander clasificadas por módulos.

Creación cursos

La figura 27 muestra la interfaz para la creación de cursos del prototipo final, para la realización de ésta y las siguientes se pueden observar los estándares de desarrollo planteados por la DSI.

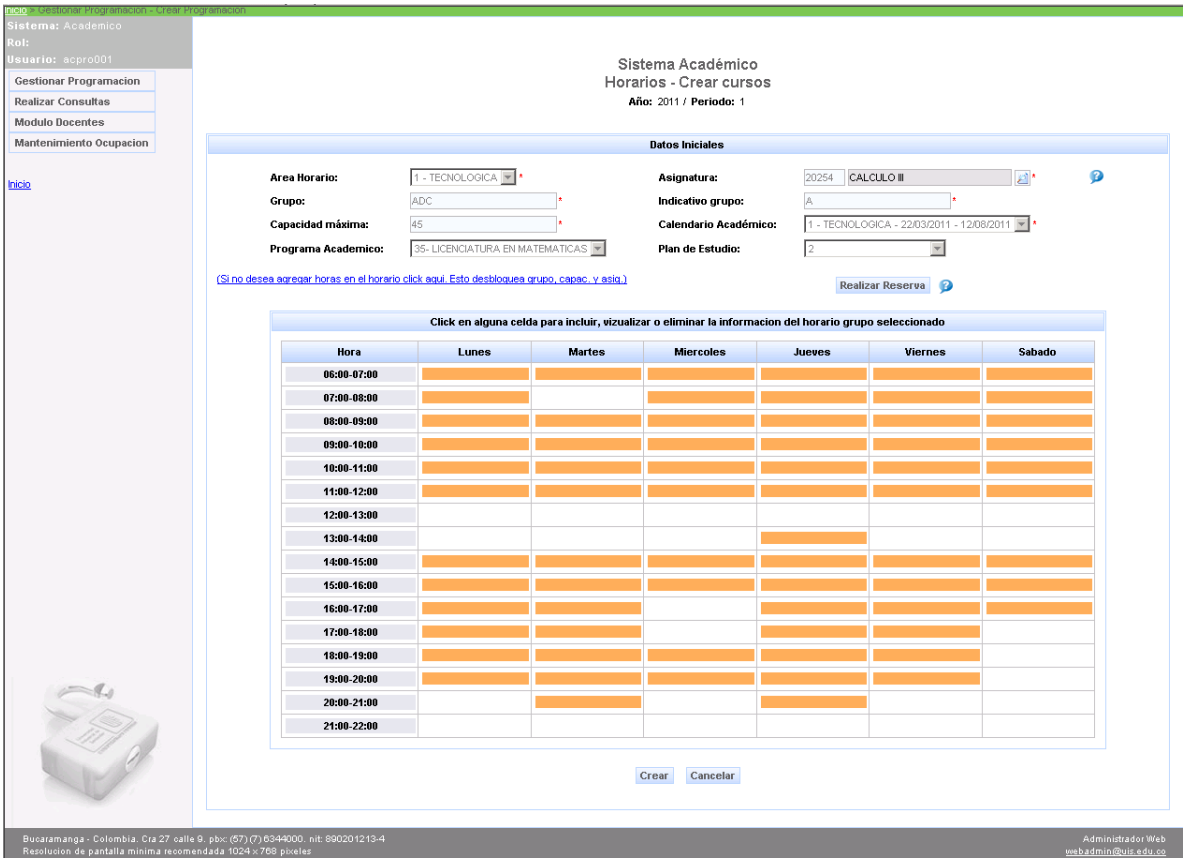
En esta interfaz se puede observar el año y periodo para el cual se realiza la programación de cursos, las distintas áreas horario, la asignatura que se está programando el cual cuenta con un buscador de asignaturas por si el usuario no recuerda el código de dicha asignatura, el nombre que se le va a dar al curso siendo solo permitido tres caracteres, el indicativo de grupo que hace referencia a si es una solicitud de grupo o un grupo activo, la capacidad máxima de estudiantes matriculados, los distintos calendarios académicos según la UAA.

También se cuenta con la opción para el Director de Escuela de seleccionar un programa académico y el plan de estudios para que pueda observar todas las asignaturas

programadas de ese plan de estudios (franjas curuba), éstas franjas están acompañadas de mensajes, cuando se pasa el cursor por una celda se puede observar los grupos que se encuentran programados en ese horario seleccionado con el fin de que no ocurran interferencias entre materias de un mismo nivel.

Como se puede apreciar esta herramienta es de gran utilidad para disminuir la cantidad de cruces entre materias de un plan de estudios situación que en la actualidad es muy común al momento de la programación de cursos, ésta herramienta ayudara a bajar la cantidad de matriculas realizadas por los estudiantes vía web que no pudieron ser satisfechas al cien por ciento (100%).

Figura 27. Vista Creación cursos. Prototipo Final

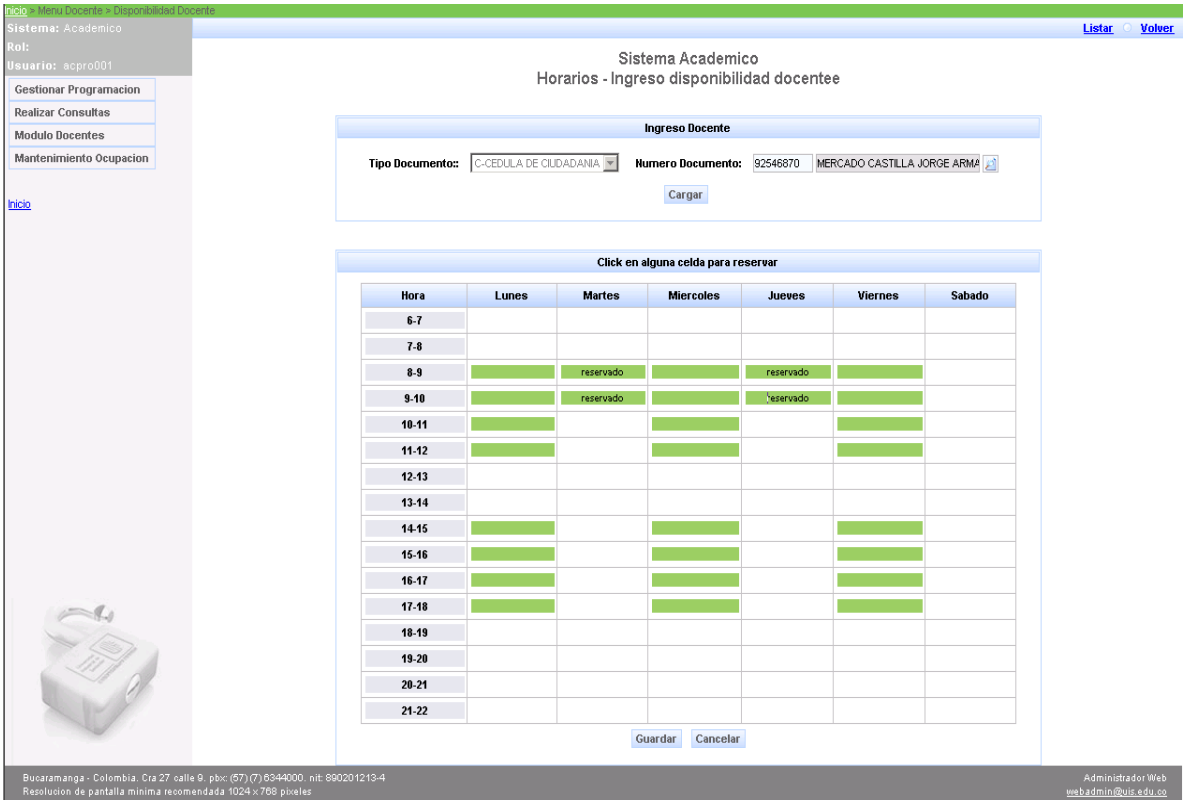


Ingreso disponibilidad docente

Esta interfaz pertenece al módulo de docentes en ella podemos observar la disponibilidad que el docente en otra sesión había ingresado y seguir agregando más disponibilidades en la sesión que tiene abierta.

Esta interfaz está planteada para la utilización de un usuario tipo administrador por lo tanto se puede ingresar el número de la cédula del docente o utilizando el buscador de personas para que sea cargada la disponibilidad que había sido ingresada previamente. También se puede generar un listado a través del hipervínculo Listar para que el docente guarde este documento como constancia del ingreso de su disponibilidad.

Figura 28. Vista Ingreso disponibilidad docente. Prototipo Final



Creación de la programación de asignaturas

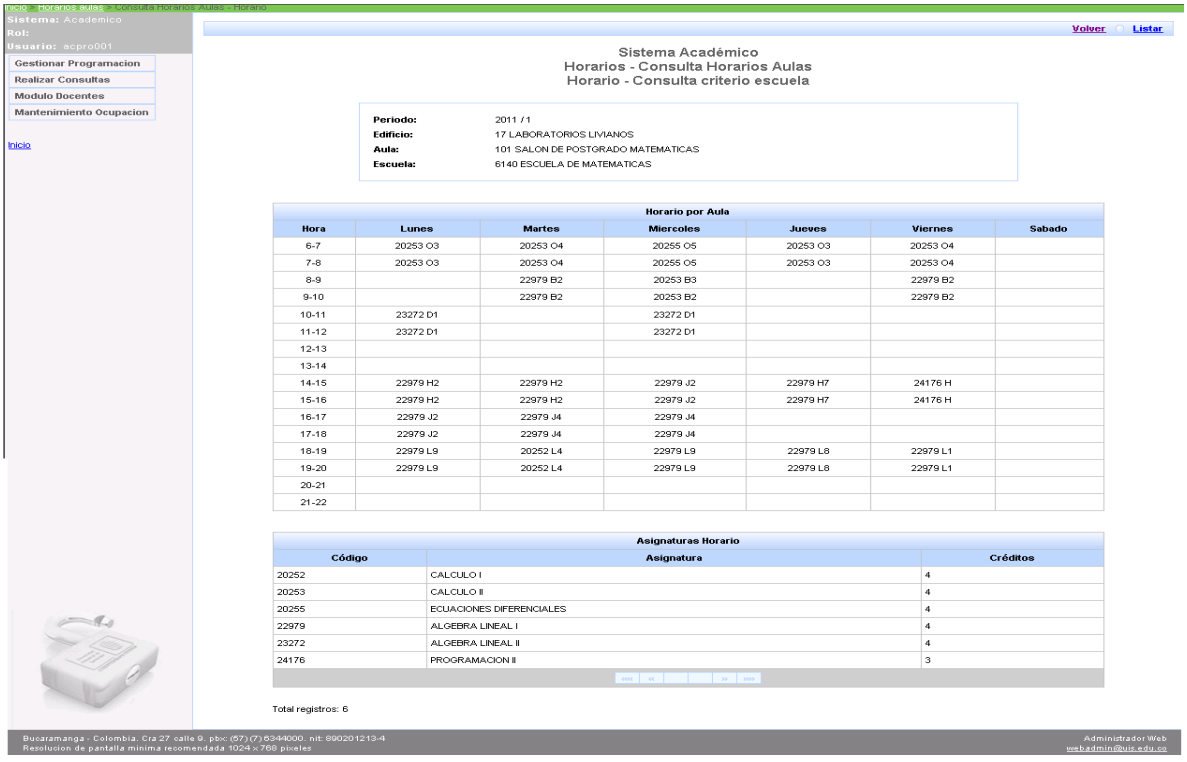
Figura 29. Vista programación de cursos. Prototipo Final

En esta interfaz podemos apreciar los grupos creados y se da la posibilidad de ver más detalles, modificar o eliminar la información ingresada para un grupo, esta modificación está condicionada a varios factores como: si el docente es cátedra y ya está nombrado solo se le puede cambiar las horas en que va a dictar pero no los días, si tiene contrato en trámite o no tiene contrato no se puede modificar, también es necesario tener en cuenta las fechas establecidas para la modificación de la programación de cursos.

Se podrá eliminar si cumple con las condiciones para este procedimiento como por ejemplo: no se puede eliminar un grupo con un docente cátedra que tenga ya contrato firmado y por ultimo también cuenta con un hipervínculo para la creación de cursos y un panel de filtrado para localizar fácilmente las asignaturas que se han programado por código o por nombre, de esta manera el Director de Escuela puede ver lo que ha programado en una menor cantidad de datos para tener un mejor control del proceso realizado.

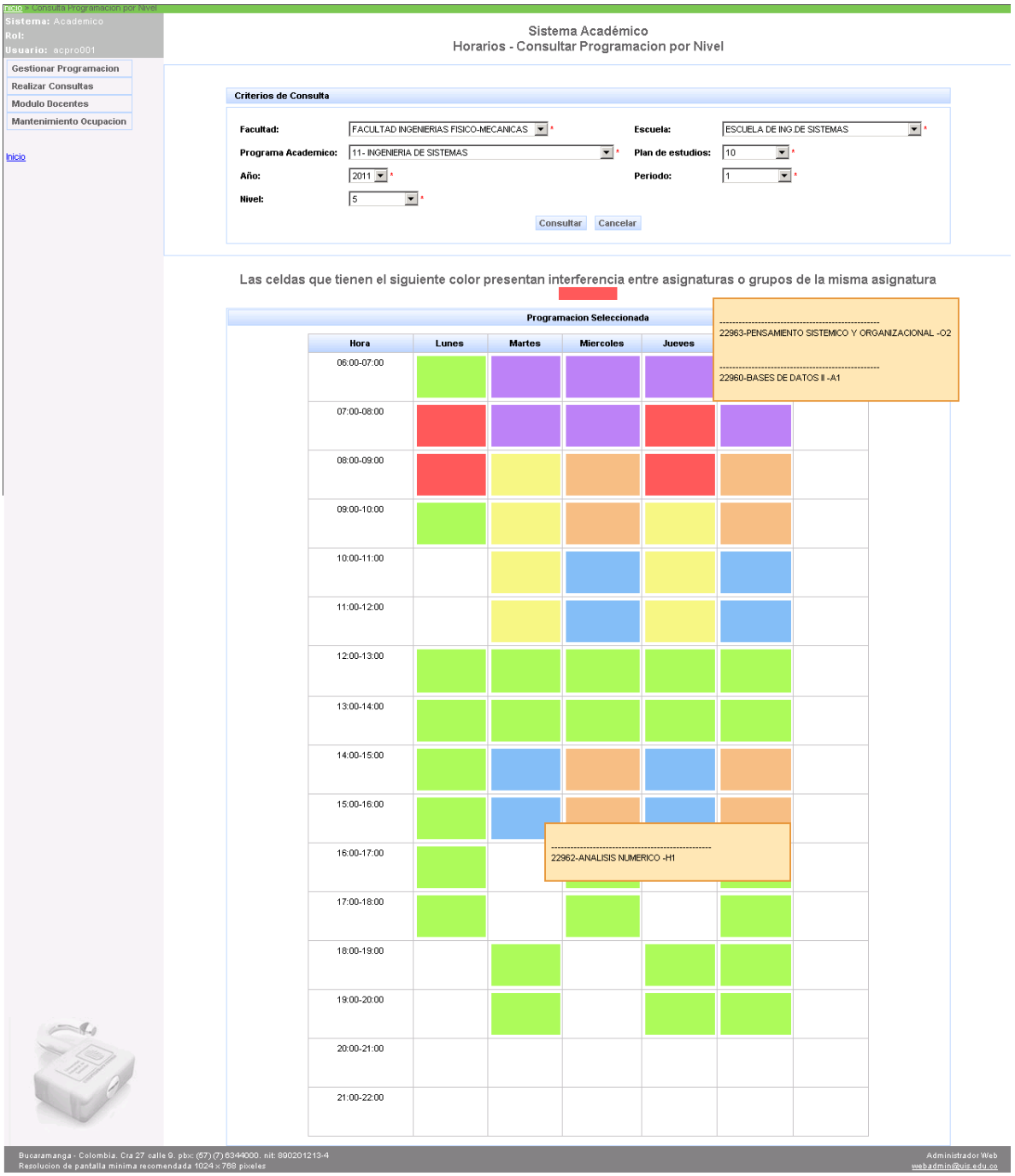
Consulta horarios aula

Figura 30. Vista consulta horarios aulas. Prototipo Final



Consulta programación por nivel

Figura 31. Vista consulta programación por nivel. Prototipo Final



En esta interfaz se puede apreciar por medio de colores los cursos de las distintas asignaturas pertenecientes a un plan de estudios de un programa académico por nivel, para los cursos que se encuentren a la misma hora y en el mismo día, la celda respectiva se puede observar en un color particular para mostrar interferencias. Al situarse en una celda se puede apreciar por medio de un tooltip el curso que se encuentra en ese horario y para cada asignatura se asigna un color que la caracteriza, por ejemplo, todos las celdas celestes son cursos de la asignatura análisis numérico.

3.4.3 Estructura del sistema El sistema de programación de horarios está compuesto por los módulos: gestionar programación, realizar consultas, modulo docentes, mantenimiento. Todos estos módulos permiten a los distintos usuarios gestionar la creación de la programación de forma eficiente y eficaz.

A continuación se describe la funcionalidad de cada módulo:

- **Gestionar programación**: este módulo está encargado de la gestión de los cursos creados para las distintas asignaturas para ello se hace necesario tener dos procesos descritos a continuación.

    - **Copiar programación**: en esta página se realiza la copia de la programación de semestres anteriores, al hacerse el proceso se podrán ver estadísticas de la información copia como los cursos que fueron copiados sin aula, la totalidad de los cursos copiados, la totalidad de los cursos.

    - **Crear programación**: aquí se puede observar cómo va la programación que se está planteando, también se brinda la posibilidad que el usuario cree cursos en donde se puede utilizar un buscador de personas, un buscador de aulas y un buscador de asignaturas para facilitar la labor de ingreso de datos al usuario.

- **Realizar consultas**: este módulo es una herramienta importante para el usuario al brindar información usando ayudas visuales e interfaces agradables y acordes a sus necesidades.

- **Consultar programación:** permite visualizar la programación actual de las asignaturas.

- **Consultar asignatura:** muestra una asignatura en particular permitiendo que el usuario revise si dos cursos de una misma asignatura se encuentran programados para la misma hora en un mismo día.

- **Consultar cursos:** en este módulo se pueden visualizar las siguientes consultas.

  - **Cursos sin aula:** son los cursos que han sido programados y aún no tienes aula asignada.

  - **Consultar un curso:** permite visualizar los horarios de un curso en particular.

- **Consultar docentes:** este módulo lista los docentes por escuela y permite observar la programación de cursos para cada docente.

- **Programación por nivel:** es una ayuda significativa para el usuario debido a que permite visualizar de una forma agradable e intuitiva los cursos programados para un nivel de un plan de estudios y si existe cruces entre asignaturas de un mismo nivel el usuario pueda identificar cada cruce y los grupos que lo ocasionaron.

- **Programación por aula**: permite visualizar la ocupación de un aula.

- **Modulo docentes**: este módulo será usado por los docentes permitiéndoles consultar su horario, ingresar su disponibilidad y las asignaturas que desean dictar.

- **Consultar horario:** el docente puede visualizar como quedo programado su horario.

- **Ingreso disponibilidad:** el docente puede ingresar su disponibilidad para dictar clases a unas horas en unos días en específico.

- **Asignaturas a dictar:** el docente ingresa las asignaturas que se encuentra en capacidad de dictar.

- **Mantenimiento:** este modulo es usado por los administradores del sistema para realizar mantenimientos a las tablas.

  - **Ocupación docente**: en este modulo se permite consultar, crear, modificar o eliminar ocupaciones de un docente.

3.5 PROYECTO ENMARCADO EN EL ESQUEMA DE SEGURIDAD DE LA UIS

Para este proyecto se utiliza el esquema de seguridad definido por la División de Servicios de Información para los diferentes sistemas de información que apoyan la gestión de la Universidad Industrial de Santander, el cual está basado en la estructura de roles – usuarios.

Los roles se establecen en cada una de las unidades académico administrativas, UAA, responsables de cada sistema, de acuerdo a las actividades que realizan. A cada uno de los roles definidos se le asocian los usuarios de acuerdo a las funciones que desempeñen.

Para el sistema de horarios de la UIS existen tres roles o tipos de usuarios principales:

- **Administrador del Modulo Horarios:** persona encargada de administración del modulo y a su vez tener una visión global de los procesos realizados dentro del mismo.

- **Director del Escuela:** persona encargada de la realización de la programación de cursos para el semestre que se está proyectando en una Escuela en específico.

- **Secretarias:** al igual que los Directores de Escuela las secretarias también son parte activa del proceso de programación de las asignaturas para una Escuela.

Estructura de la Base de Datos soporte

La base de datos que soporta el esquema de seguridad contempla básicamente las siguientes tablas:

Sistema: Contiene información de los sistemas de información de la universidad. Para cada sistema se específica: Nombre, descripción del sistema, fecha y hora de creación en la base de datos, fecha y hora de inicio de vigencia del sistema, fecha y hora de cierre de vigencia del sistema.

Rol: contiene información de los diferentes roles definidos para cada sistema de información, como: Nombre asignado al rol, descripción del rol, fecha y hora de creación, fecha y hora de inicio de vigencia del rol, fecha y hora de cierre de vigencia del rol.

Usuario: Contiene información de los posibles usuarios de los sistemas de información. Entre esta información está: tipo y número de documento de identidad del usuario, fecha y hora de creación del usuario, fecha y hora de inicio de vigencia del usuario, fecha y hora de cierre de vigencia del usuario.

Sistema–rol: Contiene los roles definidos para cada uno de los sistemas de información, indicando: rol, sistema, fecha y hora de creación del rol – sistema, fecha y hora de inicio de vigencia del rol en el sistema, fecha y hora de cierre de vigencia del rol en el sistema.

Rol–usuario: Contempla los usuarios asociados a cada uno de los roles definidos, considerando: Rol, usuario, fecha y hora de creación del rol – usuario, fecha y hora de inicio de vigencia del usuario en el rol, fecha y hora de cierre de vigencia del usuario en el rol.

Menú–rol–sistema: Contiene los menús asociados a los roles en los distintos sistema de información, contemplando: Sistema de información, nombre del menú, descripción del menú, fecha y hora de creación del menú, fecha y hora de inicio de vigencia del menú asociado al rol, fecha y hora de cierre de vigencia del menú asociado al rol.

Opción–menú–rol: Contempla las opciones definidas para cada una de los posibles menús establecidos para cada sistema de información. Contiene: Nombre de la opción, descripción de la opción, nombre del menú superior, nombre del menú que contiene la opción, nombre del programa a ejecutar cuando la opción es la de más bajo nivel, fecha y hora de creación de la opción del menú, fecha y hora de inicio de vigencia de la opción, fecha y hora de cierre de la opción.

Tabla–sistema: Contiene información de las tablas que conforman la base de datos que soporta cada uno de los sistemas de información. Considera: Sistema de información, nombre de la tabla, descripción de la tabla.

Tipo–permiso: Establece para cada tabla de un sistema de información, los roles que tienen permisos para incluir registros, para modificar registros o para eliminar registros en ella. Contiene: Sistema de información, nombre de la tabla, clase de permiso (inclusión, modificación, eliminación de registros), fecha y hora de creación del permiso, fecha y hora de inicio de vigencia del permiso, fecha y hora de fin de vigencia del permiso.

Acceso–tabla: Define para las tablas de un sistema de información si un rol tiene permiso sobre toda la información de la tabla o sobre una parte de esta. Considera: Sistema,

nombre de la tabla, clase de acceso (total, parcial), fecha y hora de creación del permiso, fecha y hora de inicio de vigencia del permiso, fecha y hora de fin de vigencia del permiso.

Atributo–tabla: Establece los atributos sobre los cuales se debe controlar el acceso a una tabla, cuando a un rol se le concede permiso para hacer uso parcial de la información existente en una tabla. Contiene: Sistema de información, nombre de la tabla, nombre del atributo sobre el cual se controla el acceso a la tabla, descripción del atributo, fecha y hora de creación del atributo, fecha y hora de inicio de vigencia del atributo, fecha y hora de fin de vigencia del atributo.

Valor–atributo-proceso: Contiene los valores que deben tener los atributos definidos en cada tabla en la tabla atributo – tabla que permiten el acceso a la información asociada a estos valores. Específica: Sistema de información, nombre de la tabla, nombre del atributo, valor del atributo, descripción, fecha y hora de creación del valor del atributo, fecha y hora de inicio de vigencia del valor del atributo, fecha y hora de fin de vigencia del valor del atributo.

Acceso-sistema: Contempla el histórico de acceso que un usuario ha realizado a un sistema, identificando las opciones que ha seleccionado. Contiene: Login de usuario, rol, identificación de la sesión, sistema, opción seleccionada, fecha y hora de ingreso, fecha y hora de salida.

Entorno de Navegación

Para cada sistema de información, la UAA responsable define los roles necesarios para el adecuado uso del sistema de información de acuerdo a las funciones que realice y establece los usuarios asociados a cada uno de ellos.

Para cada rol se define el menú de inicio, el cual le permite a cada usuario que hace parte de este rol, empezar la navegación por las distintas opciones que le ofrece el sistema, hasta llegar al nivel más bajo en el cual se ejecuta el proceso que soporta la actividad que desea realizar.

Este entorno está soportado por las siguientes tablas de las base de datos del esquema de seguridad: Rol, usuario, sistema, sistema-rol, usuario-rol, menú-rol, opción-menú rol, descritas en “Estructura de la Base de Datos Soporte”.

Entorno de Control de Datos

Para los roles definidos en cada uno de los sistemas de información se especifican las tablas a las cuales puede accesar, el tipo de transacción que puede realizar sobre estas tablas (inclusión, modificación o eliminación de registros), si tiene acceso total o parcial a la información que contiene la tabla.

Para el acceso a la información de la tabla de manera parcial, se debe establecer el atributo o atributos seleccionados, los valores que estos atributos deben tener para autorizar el acceso solicitado.

Este entorno está soportado por las siguientes tablas de las base de datos del esquema de seguridad: Rol, usuario, sistema, sistema-rol, usuario-rol, tabla–sistema, tipo-permiso, acceso-tabla, atributo-tabla, valor atributo proceso, descritas en “ESTRUCTURA DE LA BASE DE DATOS SOPORTE”.

Auditoría

Todas las tablas que conforman la base de datos soporte del esquema de seguridad tienen el historial de las transacciones realizadas sobre cada una de ellas.

El historial de las transacciones de cada tabla contiene información de los registros incluidos en la tabla, de los registros modificados y de los registros eliminados. Adicionalmente, en cada transacción se especifica: Fecha de la transacción, hora de la transacción, tipo de transacción (I/U/D), tipo y número de documento de identidad del usuario que realizó la transacción, login, rol asociado, dirección IP y MAC del equipo desde el cual llevó a cabo la transacción.

3.6 DOCUMENTACIÓN DE PROGRAMAS FUENTE[10].

Nombre de archivos, variables, constantes, atributos, métodos y parámetros.

Los nombres dentro del código fuente siguen los parámetros establecidos en el estándar general de nombres, con las siguientes particularidades.

- Los nombre de los parámetros de los métodos empiezan con guión de piso, el resto del nombre sigue el estándar para nombres de variables. Una vez dentro del código fuente, se deben asignar a una nueva variable con el mismo nombre pero sin el guión de piso.
- Los nombres de constantes o variables finally, se escriben en mayúscula sostenida separando las palabras por guiones de piso.

Comentarios Java:

Los programas en Java pueden tener dos tipos de comentarios: comentarios de implementación y comentarios de documentación. Los comentarios de implementación están delimitados por /*.. */ cuando se trata de varias líneas y // cuando se trata de una línea. Los comentarios de documentación (conocidos como "comentarios JavaDoc") son específicos de Java y están delimitados por:

/** ... */. Los comentarios de documentación se pueden extraer a ficheros HTML usando la herramienta javadoc.

Los comentarios de implementación están destinados a comentar el código o para comentarios sobre la implementación en particular, buscan dar una orientación y hacer

[10] Fuente: Estándares de la División de Servicios de Información de la Universidad Industrial de Santander

aclaraciones sobre la implementación a quien observa el código fuente. Los comentarios de documentación están destinados a describir la especificación del código, desde una perspectiva independiente de la implementación, están hechos para ser leídos por desarrolladores que pueden no tener necesariamente el código fuente a mano.

Los comentarios se deben usar para dar una visión general del código y para proporcionar información adicional que no esté disponible fácilmente en el propio código.

3.7 CAPA DE PRESENTACIÓN

Los colores para la interfaz deben ser blanco y verde ya que estos son los colores que posee la plataforma de la Universidad y son los colores insignia de la misma, la interfaz para el usuario debe ser amigable e intuitiva, donde se emplean colores para diferenciar los distintos cursos y otras tantas características serán diferenciadas de esta manera.

Plantilla principal

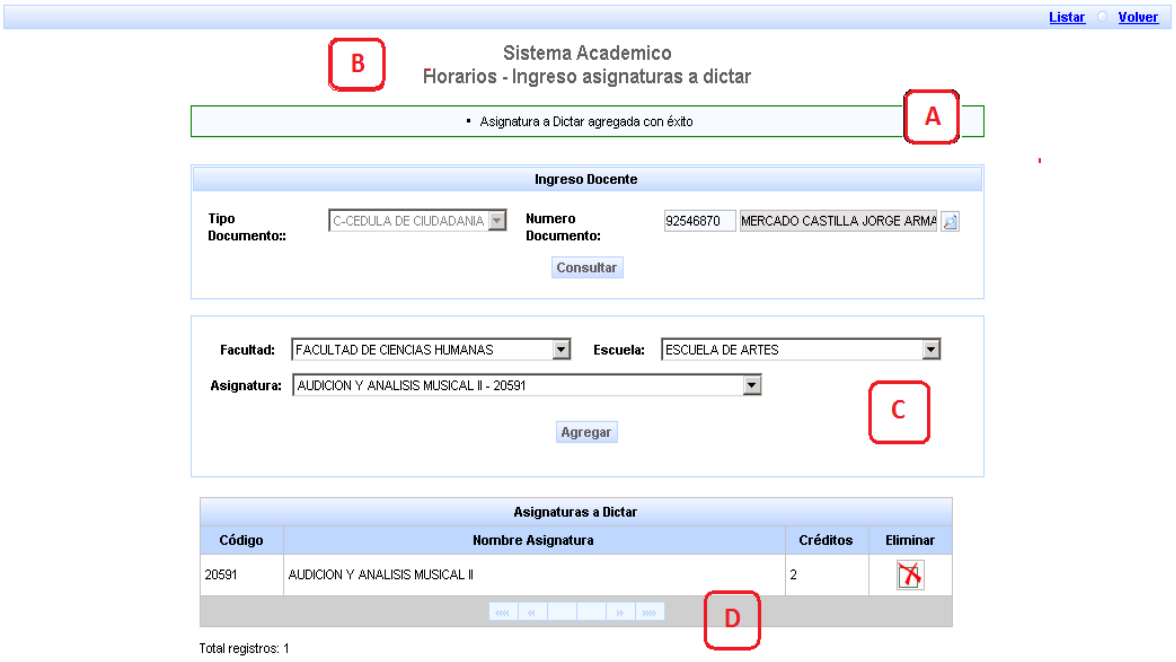
La plantilla principal de una página es la siguiente:

Figura 32. Plantilla Principal División de Servicios de Información

Contenido

Esta sección se refiere al caso de uso implementado. La estructura de esta sección es:

Figura 33. Plantilla Principal División de Servicios de Información realizada con Java



A. Mensajes: En la parte superior se visualizan los mensajes asociados a la creación de un registro. Los mensajes producto de las transacciones de edición y eliminación se visualizan en un modal panel en el centro de la página.

B. Título: Nombre del módulo seguido de el nombre del submenú seleccionado.

C. Cuerpo del formulario:
   a. Nombre del formulario: Acción que se está realizando las cuales pueden ser crear, modificar, eliminar entre otras.
   b. Campos: Información a diligenciar por el usuario. La información requerida se encuentra con un * al derecho del campo.
   c. Ayuda: El usuario dispone ayuda asociadas a cada campo en caso de no saber qué información colocar en el mismo.

D. Listados: Cada transacción que se realice como guardar, modificar o eliminar un registro se podrá visualizar en la tabla que aparece en la parte inferior del formulario. Asociado a cada elemento se encuentran las acciones disponibles para este.

Paginación

Para la paginación se debe tener en cuenta el tipo de consulta que se va a realizar y la memoria consumida por ésta en el servidor de base de datos. De acuerdo a esto la aplicación decide el número de registros máximos que debe retornar la consulta.

Por ejemplo, se quiere consultar los estudiantes de la sede Bucaramanga. De acuerdo al análisis realizado, no hay un consumo alto de memoria, por lo que se decide retornar 100 estudiantes máximos. Esto indica que cuando se implemente el método de consulta en JPQL, el parámetro setMaxResults debe tener el valor de 100.

Texto

Existen cinco tipos de plantillas para mostrar texto en la página. Éstas se encuentran dentro de la carpeta Plantillas.

- etiqueta.xhtml: Texto o datos.
- etiquetaColumna.xhtml: El título de la columna en una tabla
- titulo.xhtml: Titulos
- etiquetaNavegacion.xhtml: Utilizado en la barra de navegación
- mostrar.xhtml: Permite visualizar dos etiquetas (etiqueta, dato) al mismo tiempo. Su objetivo es la de visualizar datos como si fuera un formulario.

La sintaxis para utilizar las plantillas son:

```
<s:decorate template="../Plantillas/[nombrePlantilla]">
        <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
        </ui:define>
</s:decorate>
```

Donde nombrePlantilla puede ser etiqueta.xhtml, etiquetaColumna.xhtml, etiquetaTitulo.xhtml o etiquetaNavegacion.xhtml.

Para la plantilla mostrar.xhtml:

```
<s:decorate template="../Plantillas/mostrar.xhtml">
        <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
        </ui:define>
        <h:outputText   value="#{formatoWeb.usuario.primerNombre}"/>
</s:decorate>
```

Edición

Para capturar cualquier tipo de dato en una caja de texto se debe utilizar la plantilla edicion.xhtml de la siguiente manera:

```
<s:decorate id="dcr[identificador]"   template="/Plantillas/edicion.xhtml">
   <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
   </ui:define>

   <h:inputText id="txt[nombreCajaDeTexto]" value="[valor]"
                required="true">
```

```
        <a:support event="onblur"    reRender="dcr[identificador]"/>
    </h:inputText>
</s:decorate>
```

dcr[identificador] es el nombre del elemento decorate. Por ejemplo: dcrPrimerNombre.
txt[nombreCajaDeTexto] es el nombre de la caja de texto. Por ejemplo: txtPrimerNombre.

Tablas estáticas

Se debe usar el control panelGrid de Java Server Faces.
Tablas dinámicas

Se debe usar el control dataTable de Rich Faces, agregando las siguientes líneas de programación en su definición:
onRowMouseOver="this.style.backgroundColor='#E1E1E1'"
onRowMouseOut="this.style.backgroundColor='#{a4jSkin.tableBackgroundColor}'
Las cuales indican el color de la fila cuando el puntero del mouse esta sobre ésta.

Listas desplegables

Para cualquier tipo de lista se debe utilizar el control de Java Server Faces.

Mensajes del sistema

Los mensajes del sistema son textos enviados por los EJB de sesión, ya sea de confirmación de una acción o errores en el procedimiento. Dichos errores se deben mostrar en la etiqueta FacesMessages la cual debe estar definida al comienzo de la página después del menú si existiera éste. El código es el siguiente:

<h:messages globalOnly="true" styleClass="message"/>

Para mostrar errores de validación en los formularios, éstos deben aparecer al frente de cada control y no globalmente.

Orden dentro de los archivos de código fuente:

Cada archivo de código fuente Java debe contener una única clase o interfaz pública y deben seguir el siguiente orden:

- Sentencias package.
- Sentencias import.
- Comentario de documentación de la clase/interfaz.
- Declaraciones de clase e interfaz.
- Comentario de la implementación de la clase/interfaz si fuera necesario.
- Declaración de constantes (solo se declaran dentro de interfaces).
- Declaración de atributos. (la declaración se debe hacer uno por línea y al frente debe ir un comentario de implementación de ser necesario.)
- Declaración de variables. (Según los estándares no se deben declarar variables públicas, para ello se definen los métodos gets() y sets()). Las variables aquí declaradas serán privadas y se utilizarán como indicadores de estado de la clase. Las variables se deben declarar en el siguiente orden:
    - Variables estáticas: organizadas por ámbito o accesibilidad de la siguiente manera: públicas, protegidas, de paquete (sin modificador) privadas.
    - Variables de instancia: organizadas de la misma manera que las estáticas.
    - Al igual que los atributos se debe declarar una por línea para facilitar los comentarios de implementación frente a ellas en caso de necesitarse.

- Comentario de documentación sobre cada uno de los constructores (no aplica a entidades).
- Declaración de constructores. (Siempre se declarará el constructor sin parámetros, se requiera o no una acción dentro de este. No aplica a entidades).
- Comentario de documentación sobre cada uno de los métodos que lo requieran.
- Declaración de métodos del la lógica del negocio: estos deben ir agrupados por funcionalidad en lugar de por ámbito, siendo el objetivo de esta organización el hacer el código de más fácil lectura y compresión.
- Declaración de métodos gets() , sets() e is().
- Sobre escritura del método equals().
- Sobre escritura del método hashCode().
- Sobre escritura del método compare().
- Sobre escritura del método compareTo().

Ejemplo de documentación básica de un EJB de Sesión.

```
import java.util.List;

/**
 * @author Jorge Mercado y Viviana Jaimes Este EJB permite que se visualize la
 *       consulta de los cursos que no tienen un aula asignada para una escuela
 *       en particular
 *
 */
@Stateful
@Scope(ScopeType.CONVERSATION)
@Name("consultarCursoSinAula")
public class ConsultarCursoSinAulaEJB implements ConsultarCursoSinAula {

    @In
        public EntityManager          em;

        @In(required = false)
```

```
        StatusMessage                       statusMessage;

        private DetalleHorario              detalleHorario;
        private Facultad                    facultad;
        private EscuelaDepartamento         escuela;
        private HorarioGrupo                horarioGrupo;
        private EntornoHorario              consultaEntornoHorario;
        private List<DetalleHorario>        consultaDetalleHorario;
        private List<Facultad>              consultaFacultad;
        private List<EscuelaDepartamento>   consultaEscuelaDepartamento;
        private List<HorarioGrupo>          consultaHorariosGrupos;
        private List<EntornoHorario>        entornoHorario;
        private List<SelectItem>            itemsFacultad;
        private List<SelectItem>            itemsEscuela;

        /**
         * Este método se encarga de realizar la consulta
         * de entorno horario
         *
        */
        @SuppressWarnings("unchecked")
        public void consultarEntornoHorario() throws DSIException {
         try{
                 try {
                         String jpql = "SELECT p " +
                                         "FROM EntornoHorario p";
                         Query query = em.createQuery(jpql);
                         entornoHorario = query.getResultList();
          } catch (Exception e) {
    LOG.error(e.toString());
    throw new DSIException(e);
   }
  } catch (DSIException ee) {
   ee.printStackTrace();
   }
        }

        /**
         * Este método se encarga de consultar las facultades
         * de la universidad
         *
        */
```

```
        @SuppressWarnings("unchecked")
        public void consultarFacultad() throws DSIException {
         try{
                try {
                        String jpql = "SELECT p " +
                                        "FROM Facultad p";
                        Query query = em.createQuery(jpql);
                        this.setConsultaFacultad(query.getResultList());
         } catch (Exception e) {
   LOG.error(e.toString());
   throw new DSIException(e);
  }
 } catch (DSIException ee) {
  ee.printStackTrace();
  }
        }

        /**
         * Este método se encarga de la consulta de las escuelas
         * para una facultad en particular
         *
        */
        @SuppressWarnings("unchecked")
        public void consultarEscuela() throws DSIException {
         try{
                try {
                        String jpql = "SELECT p " +
                                "FROM EscuelaDepartamento p " +
                                "WHERE p.facultad.codigo= codigoFacultad";
                        Query query = em.createQuery(jpql);

                        query.setParameter("codigoFacultad",this.facultad.getCodigo());
                        this.setConsultaEscuelaDepartamento(query.getResultList());

                } catch (Exception e) {
   LOG.error(e.toString());
   throw new DSIException(e);
  }
 } catch (DSIException ee) {
  ee.printStackTrace();
  }
        }
```

```
	/**
	 * Este método se encarga de la inicializacion de los datos
	 * necesarios para la consulta de cursos sin aula asignada
	 *
	 */
	public void inicioCursoSinAula() {
	 try{
		try {
			if (facultad == null) {
				consultarFacultad();
			}

			if (this.horarioGrupo == null) {
				this.horarioGrupo = new HorarioGrupo();
				this.horarioGrupo.setHorarioGrupoId(new
HorarioGrupoId());
				this.horarioGrupo.getHorarioGrupoId().setGrupoAsignatura(
						new GrupoAsignatura());
				this.horarioGrupo.getHorarioGrupoId().getGrupoAsignatura()
						.setGrupoAsignaturaId(new
GrupoAsignaturaId());
				this.horarioGrupo.getHorarioGrupoId().getGrupoAsignatura()
						.getGrupoAsignaturaId().setCodigoAsignatura(
								new Asignatura());
				this.horarioGrupo.getHorarioGrupoId().getGrupoAsignatura()
						.getGrupoAsignaturaId().getCodigoAsignatura()
						.setEscuelaDepartamento(new
EscuelaDepartamento());

			}
		 } catch (Exception e) {
	    LOG.error(e.toString());
	    throw new DSIException(e);
	   }
	  } catch (DSIException ee) {
	   ee.printStackTrace();
	   }
		}
		/**
		 * Este método se encarga dela consulta de los cursos sin aula
		 * aun asignada para el año y el periodo de entornoHorario para
		 * una escuela en particular   */
```

```
        @SuppressWarnings("unchecked")
        public void consultarCursosSinAula() throws DSIException {

                if (this.horarioGrupo != null) {
 try{
  try{
                String jpql = "SELECT h " +
                                "FROM HorarioGrupo h " +
                                "WHERE 1=1";
                        jpql = jpql
                                        + " and
h.horarioGrupoId.grupoAsignatura.grupoAsignaturaId.ano= :ano" ;
                        jpql = jpql
                                        + " and
h.horarioGrupoId.grupoAsignatura.grupoAsignaturaId.periodo= :periodo" ;
                        jpql = jpql
                                        + " and
h.horarioGrupoId.grupoAsignatura.grupoAsignaturaId.codigoAsignatura.escuelaDepartame
nto.codigoUnidadEscuela= :codigoEscuela";
                        jpql = jpql + " and h.edificio IS NULL";

                        Query query = em.createQuery(jpql).setMaxResults(100);

                        query.setParameter("ano",entornoHorario.get(0).getAnoHorario());
    query.setParameter("periodo",entornoHorario.get(0).getPeriodoHorario());
    query.setParameter("codigoEscuela",escuela.getCodigoUnidadEscuela());

                        this.consultaHorariosGrupos = query.getResultList();
  } catch (Exception e) {
   LOG.error(e.toString());
   throw new DSIException(e);
  }
 } catch (DSIException ee) {
  ee.printStackTrace();
  }
 }

        /**
         * Este método obtiene los datos de entorno horario
         * @return  consultaEntornoHorario
         */
        public EntornoHorario getConsultaEntornoHorario() {
```

```
                if (this.consultaEntornoHorario == null)
                        this.consultaEntornoHorario = new EntornoHorario();
                return consultaEntornoHorario;
        }

        /**
         * Este método asigna los datos de la consulta de
         * entorno horario
         *
         * @param  consultaEntornoHorario
         */
        public void setConsultaEntornoHorario(EntornoHorario    consultaEntornoHorario)
{
                this.consultaEntornoHorario = consultaEntornoHorario;
        }


    @Remove
        @Destroy
        public void destroy() {
        }
```

CAPITULO 4

4. CONCLUSIONES

- El desarrollo de este software permite reducir en gran medida las inconsistencias que se presentan por las interferencias entre grupos de asignaturas de un mismo nivel, brindándole a los estudiantes más posibilidades de satisfacer su matrícula.

- La herramienta de programación de horarios permite reemplazar la antigua metodología manual que se ha estado utilizando para la programación de cursos en la Universidad Industrial De Santander.

- El sistema permite una mejor gestión de los docentes y sus tiempos dado que éstos pueden ingresar vía web su disponibilidad horaria para dictar las asignaturas.

- La solución planteada es una herramienta de gran utilidad para los directores de escuela, ya que facilita el proceso de creación de cursos al indicar aspectos relevantes como interferencia entre docentes, disponibilidad horaria de éstos y las asignaturas que el docente está en capacidad de dictar.

- El uso de la herramienta permite realizar un control de las aulas con las que cuenta cada escuela, debido que al momento de crear cursos muestra la disponibilidad de aulas en la Escuela y en la Universidad para el horario seleccionado.

- Al ser una herramienta para la web aumenta su facilidad de uso y permite desarrollar un sistema con una interfaz más agradable al usuario final.
- La utilización de herramientas libres como Java, aparte de reducir costos, permite la realización de aplicaciones seguras, robustas y de tipo empresarial, debido al

permanente aporte que hacen usuarios de todo el mundo y la facilidad de capacitación online en foros de discusión.

- El proceso de construcción de prototipos no funcionales, permite un ahorro de tiempo debido a que el usuario interactúa con las entradas y salidas que va a tener el sistema y su esquema de navegación y hace las correcciones y sugerencias, sin desgastes de tiempo de programación.

CAPITULO 5

5. RECOMENDACIONES

- Seguir aprovechando este tipo de espacios con la División de Servicios de Información pues se convierte en un espacio donde los estudiantes pueden tener contacto con una dependencia que maneja procesos, estándares y últimas tecnologías de desarrollo para beneficio de la comunidad universitaria.

- Evolucionar esta primera versión que se planteó del modulo de horarios para que cada día facilite más la labor realizada por los directores de escuela y personas involucradas en el proceso de programación de cursos para un semestre.

- La Universidad debe seguir apoyando este tipo de iniciativas (desarrollo de Software para la creación de la programación de cursos) que ayudan a optimizar los procesos realizados en la misma.

- Dar a conocer ampliamente a las unidades académico administrativas y a la comunidad universitaria involucrada en estos procesos los nuevos servicios que ofrece este sistema.

- Las Escuelas deben realizar todos los esfuerzos posibles para que éste sistema de información entre en funcionamiento teniendo en cuenta los beneficios que ofrece.

- Integrar el sistema desarrollado con el sistema de asignación automática de aulas para ofrecer una solución más robusta al problema de programación de cursos y los recursos con los que cuenta la universidad para tal fin.

CAPITULO 6

6. BIBLIOGRAFÍA

[1] PRESSMAN, Roger. Ingeniería del Software, un enfoque práctico. Mc GraW Hill, 2005.

[2] COBOS, Carlos Alberto; MENDOZA, Martha Eliana. Manual De Informix – SQL . Universidad Industrial de Santander, 1998.

[3] FERRÉ GRAU, Xavier - SÁNCHEZ S. María Isabel. Desarrollo Orientado a Objetos con UML. Facultad de Informática - UPM.

[4] Estándares de desarrollo. División de Servicios de Información. Bucaramanga, UIS.

SITIOS WEB

- http://www.sparxsystems.com.ar/
- http://download.oracle.com/docs/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/pdldocs/facelets/index.html
- http://livedemo.exadel.com/richfaces-demo/richfaces/comboBox.jsf?tab=usage&cid=170149
- www.querix.com
- http://download.oracle.com/javaee/5/api/