UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"SISTEMA DE ADMINISTRACIÓN DEL PERSONAL E INFORMACIÓN DE UNICA (SAPIUN)"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

CLAUDIA IBETH VELASCO RAMÓN

DIRECTOR DE TESIS:
ING. JULIO CÉSAR JUÁREZ SOSA

CODIRECTOR DE TESIS:
ING. MARÍA DEL ROSARIO BARRAGÁN PAZ

Ciudad Universitaria

México, D.F. 2005

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a todas aquellas personas que de manera directa o indirecta han influido en mi vida y en la realización de este trabajo de tesis. En primer lugar a mi familia por su apoyo incondicional y sus enseñanzas.

Al Ing. Julio César Juárez Sosa por su amistad, su apoyo y por lo todo lo que me ha enseñado a lo largo de todo el tiempo que he tenido la fortuna de conocerlo.

A la Ing. María del Rosario Barragán Paz por su apoyo y sus sabios consejos en los momentos que más lo he necesitado.

A todas y cada una de las personas que participaron en el desarrollo del SAPIUN, sin su ayuda esto no hubiera sido posible.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por la oportunidad de formar parte de ellas.

A la Unidad de Servicios de Cómputo Académico (UNICA) por todos los conocimientos que me brindó y a todos mis compañeros que con sus observaciones me ayudaron a crecer como persona.

DEDICATORIAS

A mis padres por todo su amor, su apoyo incondicional y por enseñarme a dar lo mejor de mí siempre. Gracias mamá por estar conmigo siempre que te necesité y por impulsarme para ser una profesionista; gracias papá por darme una carrera y enseñarme a perseguir mis sueños.

A mis hermanos que son mi razón para superarme todos los días.

A mi Avatar, por iluminarme la vida y alentarme a terminar este trabajo de tesis.

A Dios por ponerme en el camino a tanta gente maravillosa.

Índice

OBJETIVOS

Objetivo General

Realizar un sistema que sea capaz de administrar la información académico – administrativa de UNICA, a través de la generación de un sistema principal que integrará a los diversos sistemas que comparten información y que en conjunto agilizarán las actividades administrativas que aquí se realizan.

Objetivo Particular

Para alcanzar el objetivo general se creará un sistema cuyo nombre será, **S**istema de **A**dministración del **P**ersonal e **I**nformación de **UN**ICA (**SAPIUN**) que integrará toda la información académico – administrativa, entre la que podemos destacar:

- ✓ *Información del personal*. Datos Personales, Horario de Trabajo, Capacitación y Nivel que ocupan.
- ✓ *Información de Cursos*. Temario, Antecedentes y Requisitos del Equipo
- ✓ *Información de Salas*. Equipo disponible, Características y Ubicación
- ✓ *Información de Grupos*. Listas de Grupos, Listas de Alumnos y Estado de Inscripción.
- ✓ *Información de Costos*. Costos por período y tipos de becas.
- ✓ *Estadísticas*. Cursos Abiertos, Tipo de Alumnos Inscritos y Remuneraciones Alcanzadas
- ✓ *Integración del módulo de inscripció*n, así como de los diversos módulos que actualmente se encuentran en desarrollo.

Todo esté proceso se realizará a través del modelado que ofrece la metodología OMT y UML, así como con el apoyo del lenguaje de programación de Visual Basic que ofrece una interfaz totalmente amigable para el usuario y que se explicará a lo largo de los siguientes capítulos.

Introducción

Capítulo I

INTRODUCCIÓN

La Unidad de Servicios de Cómputo Académico (UNICA) es una dependencia de la Secretaría General de la Facultad de Ingeniería, cuya finalidad principal es la de proporcionar, en el ámbito institucional, los servicios de apoyo en cómputo que la comunidad de la Facultad requiere, recursos de cómputo comerciales y de alta especialización en el avance de la educación y el desarrollo de la informática, que el ejercicio profesional demanda en la actualidad. Además de formar recursos humanos de calidad, tanto en el área de cómputo como en el desempeño de la vida profesional.

Entre las funciones que desempeña la Unidad de Servicios de Cómputo Académico están:

- ✓ Mantener el liderazgo en cuanto a tópicos en cómputo.
- ✓ Continuar proporcionando recursos de cómputo de calidad a la comunidad de la Facultad.
- ✓ Impulsar a nivel de la Facultad la creación de una política de cómputo definida.
- ✓ Lograr la capacitación cada vez más completa y actualizada para la formación de recursos humanos.
- ✓ Aplicar todos los conocimientos y las herramientas de cómputo con los que cuenta la Unidad para realizar las actividades de forma más eficiente y segura.

UNICA está conformada por diferentes departamentos, como se puede observar en la figura A; entre los que se encuentra el Departamento de Servicio de Cómputo Académico (DSA).

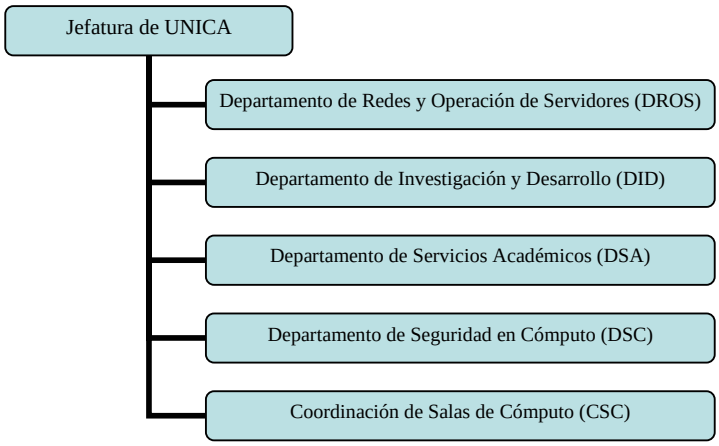


Figura A. Organización de UNICA

El DSA se encarga de la planeación, organización e impartición de los cursos de cómputo que UNICA ofrece semestre a semestre en las modalidades de: semestrales, intersemestrales, especiales, internos, externos y fines de semana dirigidos a la comunidad universitaria y público en general.

Se encarga así mismo de la formación de los recursos humanos que necesita para el cumplimiento de sus funciones; esto es mediante la coordinación del Programa de Formación de Becarios; en el que alumnos de la Facultad, con alto rendimiento académico, que así lo deseen y manifiesten , previa selección, capacitación y aceptación, se integran como personal de apoyo; cabe hacer destacar la importante participación de este recurso humano de alta especialización, para el cumplimiento de las responsabilidades de UNICA.

Por tal motivo el presente trabajo tiene como objetivo realizar un sistema de información, que permita llevar a cabo la administración de las diferentes actividades, entre las que se contemplan:

- ✓ La Información del personal que aquí labora (datos personales y laborales).
- ✓ Generación del calendario de cursos.
- ✓ Salas y equipo de cómputo disponibles.
- ✓ Estadísticas Generales.
- ✓ Reportes generales y detallados para cada uno de los puntos antes mencionados.

1.1. Requerimientos del sistema

1.1.1. Información del personal

UNICA requiere una interfaz que permita llevar una administración eficiente del personal que en ella labora, teniendo en tiempo y forma datos personales, situación académica y laboral, horario de trabajo, experiencia y capacitación en los diferentes cursos que imparte, generando un reporte detallado por persona que contenga dicha información, así como una lista general de todo el personal que indique su RFC, situación laboral, departamento al cual pertenece y número telefónico.

Esta información deberá estar disponible para todos y cada uno de los Jefes de Departamento, misma que tendrán que revisar y emitir una respuesta favorable o desfavorable con respecto al horario de trabajo del personal a su cargo e informar al DSA.

Se deberá crear una interfaz que permita el acopio de la información del personal vía Internet a la cual deberán tener acceso por medio de una clave y contraseña, en dicha interfaz se deberán ingresar los datos personales, horario de trabajo, propuesta de curso y calificaciones de aquel o aquellos cursos que hubiera impartido.

1.1.2. Información de los cursos

Es necesario crear un catálogo de cursos que contenga como dato principal, el nombre del curso así como un temario que describa cada unos de tópicos que en él se tratarán; adicionalmente se deberán dar a conocer los antecedentes necesarios para tomarlo. Generando un reporte detallado con la información recopilada y un reporte general con los nombres y requerimientos de los cursos que se imparten.

Para cada curso es indispensable tomar en cuenta los recursos mínimos de hardware y software que conlleven a su correcta asignación, según las características de cada una de las salas de cómputo con las que cuenta UNICA.

1.1.3. Información de las salas de cómputo

También se necesita tener el control de la información concerniente a los espacios físicos habilitados para el desarrollo de todos y cada uno de los cursos, asignando un lugar y sección a cada sala, para lo cual se recopilarán datos técnicos del equipo, paquetería instalada y número total de equipos disponibles por sección.

Dicha información servirá en la asignación de vacantes para cada curso y lugar donde se impartirá el mismo, así como para la creación de un reporte general de salas e información detallada por sección, donde se incluirán las características antes mencionadas.

1.1.4. Información de los costos

Por cada curso que se imparte se debe asignar un costo tomando como base el nivel del curso (básico, intermedio o avanzado) y el tipo (programación, manejo de paquetería, diseño, etc.). Una vez publicado el calendario de cursos, el costo se verá afectado por un descuento que estará en función del

tipo de persona que desee inscribirse a uno o más cursos, para lo cual es necesario generar una interfaz que administre el catálogo de descuentos, dicho catálogo contendrá la descripción del tipo de persona y el monto del descuento correspondiente. Por lo que se deberá contar con un reporte general de costos por período y un reporte detallado por tipo de modalidad.

1.1.5. Información de estadísticas

Una vez concluido el período de impartición de cursos se generarán las siguientes estadísticas:

- ✓ Cuantos cursos se abrieron y cuantos se cancelaron.
- ✓ Cuantos alumnos se inscribieron por modalidad.
- ✓ El ingreso total de todos los cursos e ingreso de cada uno por modalidad.

1.2. Propuesta de desarrollo

Para el desarrollo del sistema se propone utilizar una metodología orientada a objetos ya que al contrario que la programación estructurada tradicional, la programación orientada objetos (POO) sitúa los datos y las operaciones relacionadas con ellos en una única estructura de datos. En la programación estructurada, los datos y la operaciones sobre ellos están separadas, y las estructuras de datos se envían a los procesos y funciones en las que van a ser manejadas. La POO resuelve muchos de los problemas inherentes a este diseño, porque los atributos y las operaciones forman parte de la misma entidad. Este diseño imita más de cerca al mundo real, en la que todos los objetos poseen tanto atributos como actividades asociadas con ellos; de tal forma que el diseño y desarrollo del sistema puede llevarse a cabo en forma distribuida, lo que representa como ventaja un mayor avance y una mejor sincronización.

Los módulos que conformarán al SAPIUN son:

Módulo 1. Control de la información relacionada con el personal que labora en UNICA; datos personales, horario, experiencia como instructor o alumno (cursos impartidos o adquiridos), considerando que únicamente el personal del DSA y jefes de Departamento manejaran ésta información.

Los reportes que se generarán son los siguientes:

- ✓ Lista de todo el personal indicando nombre completo, RFC, situación laboral, departamento y número telefónico.
- ✓ Detalle por persona considerando:
  - o Datos personales.
  - o Situación académica y laboral.
  - o Horario de trabajo.
- ✓ Listado de cursos impartidos o adquiridos por persona dentro de UNICA.

Módulo 2. Registro de datos personales, que a su vez está dividido en dos casos:

- ✓ Registro en forma personal. El personal acudirá directamente al DSA para darse de alta en el sistema.
- ✓ Registro a través de Internet. El personal solicitará una clave genérica al administrador del sistema en caso de ser su primer registro, en caso contrario lo hará mediante un clave y contraseña proporcionadas después de su primer ingreso. Además tendrá la posibilidad de enviar sus propuestas de cursos y calificar los cursos que hubiera impartido.

Módulo 3. Administración de la información necesaria para la impartición de cursos. Éste estará divido en tres partes:

- ✓ Catálogo de cursos. Contendrá la descripción del curso, antecedentes, temario y recursos de hardware y software necesarios para su impartición.
- ✓ Control de salas. Indicará la ubicación y secciones de cada una de las salas, así como las características técnicas y número total de equipos por sección.
- ✓ Catálogo de costos. Asignará los correspondientes costos por nivel y descuentos por tipo de alumno, en cada una de las modalidades existentes.

Módulo 4. Generación de estadísticas. Una vez terminado el período de cursos el sistema será capaz de generar estadísticas, considerando las siguientes agrupaciones:

Por período

- ✓ El número total de cursos, divididos en abiertos y cancelados indicando la modalidad a la que pertenece.
- ✓ Total de alumnos inscritos por modalidad.
- ✓ Ingresos totales divididos por modalidad.

Por modalidad

- ✓ Nombre de los cursos, estado asociado y total por curso-estado.
- ✓ Detalle del tipo de alumnos inscritos y total por tipo.

  Detalle de ingresos obtenidos por tipo de alumno.

Recursos

Para llevar a cabo el sistema SAPIUN se requiere como mínimo del siguiente hardware y software instalado:

Computadoras personales con procesador Pentium III y tarjeta de red.

Sistema Operativo Windows 2000, XP ó NT Server

Protocolo TCP/IP

Un visualizador WEB (Netscape Communicator o Internet Explorer)

Visual Basic 6.0

ODBC Postgres

Rational Rose

Crystal Reports 8.0

Una computadora con procesador Pentium IV y tarjeta de red que fungirá como servidor.

Sistema Operativo Linux

Protocolo TCP/IP

Servidor WEB (Apache)

Servidor PHP

Servidor Postgres 7.0

1.3. Metodologías de desarrollo

A partir de los años 60 se planteó un nuevo problema debido a la necesidad de racionalizar la producción de software para aumentar la productividad en el campo de la realización y puesta a punto de los programas, ya que con el aumento de la complejidad de los sistemas de información, crecía la dificultad de fabricar de modo correcto y rápido los programas necesarios. Por esta razón se puso tanto interés en el diseño de diferentes técnicas y métodos de realización de sistemas de información.

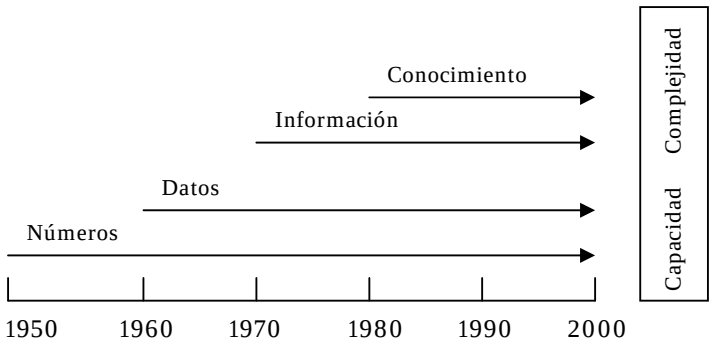


Figura 1.1 Crecimiento de la complejidad del software

Con la evolución de las metodologías de desarrollo, la realización de software se ha enfocando en la comodidad del usuario, dando como resultado la generación de tareas, procedimientos y técnicas, para la automatización en la construcción de sistemas de información. Dando origen a un periodo de Desarrollo Convencional, seguido de un Desarrollo Estructurado y finalmente aparece el paradigma de la Orientación a Objetos como un nuevo enfoque de desarrollo de software.

1.3.1. Características de una metodología

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto de software, esto es, define una serie de pasos a seguir para obtener un software de calidad.

Por lo tanto una metodología representa el camino a seguir para desarrollar software de manera sistemática. Teniendo como objetivo: el desarrollo de mejores aplicaciones bajo un estándar en los procesos de organización, que identifique los productos intermedios de cada fase de forma que se pueda planificar y controlar el proyecto.

Considerando las definiciones anteriores podemos determinar que las características que distinguen a una metodología son:

- ✓ Tareas. Actividades elementales en que se dividen los procesos.
- ✓ Procedimientos. Definición de la forma de ejecutar la tarea, vínculo de comunicación entre usuarios y desarrolladores.
- ✓ Productos. Resultados de un procedimiento, pueden ser intermedios o finales.

- ✓ Técnicas. Herramientas utilizadas para aplicar un procedimiento, pueden ser gráficas y/o textuales, que determinan el formato de los productos resultantes en cada tarea.
- ✓ Herramientas. Proporcionan el soporte a la aplicación de las técnicas, podemos apoyarnos con herramientas de software.

1.3.2. Evolución de las metodologías

En la década de los 50's el desarrollo de los sistemas de información era artesanal debido a la ausencia de metodologías, pues solo se enfocaba en la tarea de programación. Con la aparición de los sistemas de información administrativos en la década de los 60's, el desarrollo de software tomo un nuevo enfoque cuyo objetivo principal era la implementación de normas que permitieran una mayor calidad en los programas. Para la década de los 80's, los grandes avances tecnológicos permitieron el desarrollo de sistemas de información de mayor potencia; ahora los procesos y los datos eran tratados de manera conjunta, dando como resultado la aparición de nuevas metodologías que integran conceptos como abstracción, encapsulamiento de información y modularidad.

A continuación se muestra la evolución histórica de las diferentes metodologías:

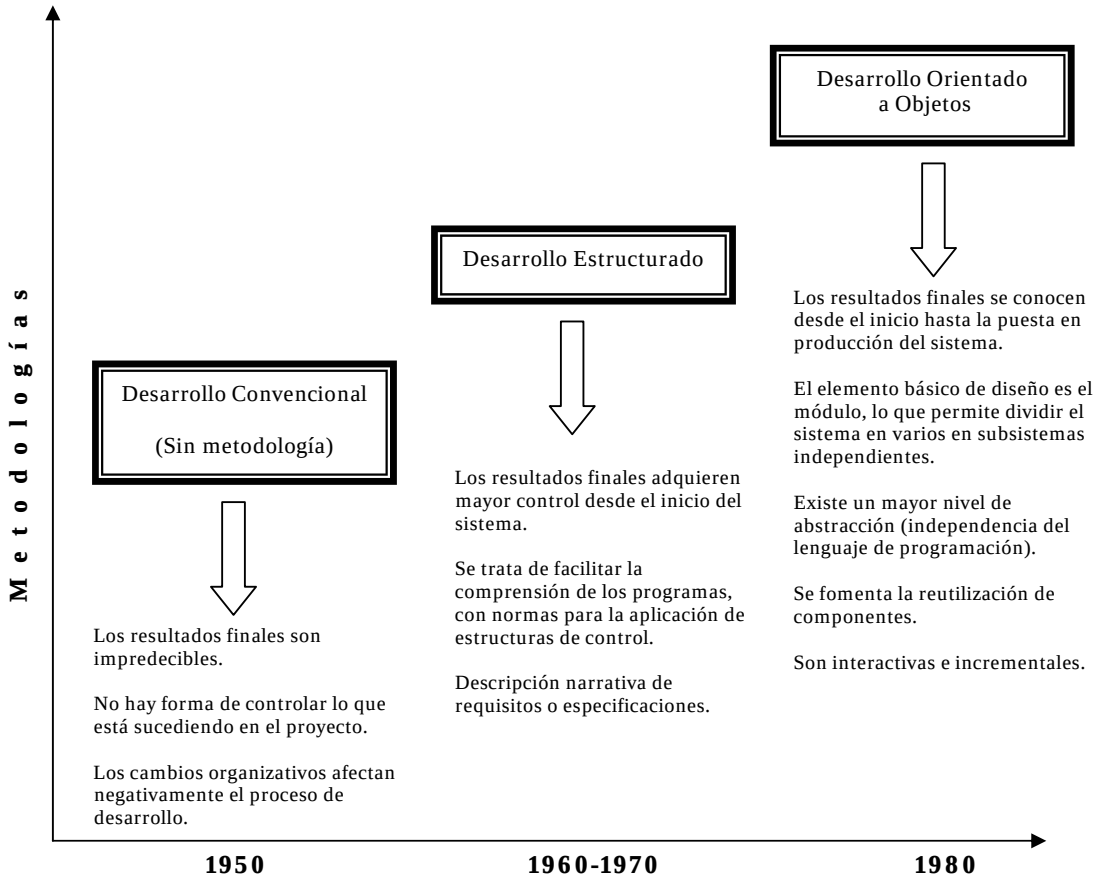


Figura 1.2 Desarrollo histórico de las metodologías

En las metodologías de análisis y diseño estructurado, se examinan los sistemas desde el punto de vista de las funciones o tareas que deben realizar, tareas que se van descomponiendo sucesivamente en otras tareas más pequeñas y que forman los bloques o módulos de las aplicaciones. En la orientación a objeto, por su parte, cobra mucho más importancia el aspecto de "modelado" del sistema, examinando el dominio del problema como un conjunto de objetos que interactúan entres sí.

1.3.3. Metodologías orientadas a objetos

Una metodología Orientada a Objetos enfoca el problema como la solución en los mismos términos: clases, objetos, métodos, atributos y comportamientos. A lo largo del proceso de desarrollo se debe mantener la consistencia de terminología y de perspectiva. Por ejemplo, no importa donde se represente un objeto en el proceso, se debe incluir el nombre del objeto, atributos y comportamientos.

Un proceso Orientado a Objetos utiliza el encapsulamiento de datos y el comportamiento para formar unidades independientes (objetos). Las mismas estructuras semánticas representan el sistema desde la formulación de requerimientos a la implementación y prueba de la aplicación. Así la metodología es una filosofía de representación del problema y solución, y no una representación del ciclo de vida en sí. De esta forma puede aplicarse en cualquiera de los ciclos de vida, desde el modelo en cascada al modelo en espiral.

Una metodología Orientada a Objetos consta de:

- ✓ Conceptos y diagramas.
- ✓ Etapas y definición de entregas en cada una de ellas.
- ✓ Actividades y recomendaciones.

Entre las principales metodologías orientadas a objetos podemos mencionar:

Object Oriented Design (Diseño Orientado a Objeto), por Grady Booch

La metodología de Booch usa los siguientes tipos de diagramas para describir las decisiones de análisis y diseño, que deben ser hechas en la creación de un sistema orientado a objetos.

- ✓ *Diagrama de Clases*. Consisten en un conjunto de clases y relaciones como lo son asociaciones, contenencia, herencia, uso, instanciación y metaclase.
- ✓ *Especificación de Clases*. Es usado para capturar toda la información importante acerca de una clase en formato texto.
- ✓ *Diagrama de Categorías*. Muestra clases agrupadas lógicamente bajo varias categorías.
- ✓ *Diagramas de transición de estados*.
- ✓ *Diagramas de Objetos*. Muestra objetos en el sistema y su relación lógica, donde se muestra cómo colaboran los objetos en cierta operación; o bien la existencia de los objetos y las relaciones estructurales entre ellos.
- ✓ *Diagramas de Tiempo*. Aumenta un diagrama de objetos con información acerca de eventos externos y tiempo de llegada de los mensajes.

- ✓ *Diagramas de módulos*. Muestra la localización de objetos y clases en módulos del diseño físico de un sistema. Un diagrama de módulos representa parte o la totalidad de la arquitectura del sistema.
- ✓ *Subsistemas*. Un subsistema es una agrupación de módulos
- ✓ *Diagramas de procesos*. Muestra la localización de los procesos.

La siguiente tabla muestra las diferentes etapas de esta metodología:

	Actividades	**Definición de entregas**
Análisis de requerimientos	En esta etapa se define qué quiere el usuario del sistema; identifica las funciones principales del sistema, el alcance del modelamiento y documenta los procesos principales y las políticas que el sistema va a soportar.	*Funciones primarias del sistema*: Principales entradas y salidas del sistema, referencias a políticas, sistemas existentes o procedimientos. *Conjunto de mecanismos claves que el sistema debe proveer*: estado de entrada, estado de salida y estados esperados.
Análisis de dominio	Es el proceso de definir de una manera concisa, precisa y orientada a objetos el modelo del sistema.	*Diagrama de clases*: Identificación de las clases y sus relaciones. *Especificación de las clases*. *Vistas de herencia*: Diagramas de clases con relaciones de herencia. *Diagramas de escenarios de objetos.* *Especificación de objetos*: Relación entre objetos y sus clases.
Desarrollo	Es el proceso de determinar una implementación efectiva y eficiente que realice las funciones del análisis de dominio.	*Descripción de arquitectura*: Describe las decisiones más importantes del diseño como son: conjunto de procesos, manejadores de bases de datos, sistemas operativos, lenguajes, etc. *Descripciones de prototipo*: Describen las metas y contenido de las implementaciones sucesivas, su proceso de desarrollo y la forma de probar requerimientos. *Diagramas de Categorías*. *Diagramas de clases en diseño*: Detallan las abstracciones de análisis con características de implementación. *Diagramas de objetos en diseño*: Muestran las operaciones necesarias para desarrollar una operación. *Especificaciones de clases corregidas*: Muestra la especificación completa de los métodos con algoritmos complicados, la implementación de relaciones y el tipo de atributos.

Tabla 1 Etapas de la metodología Booch

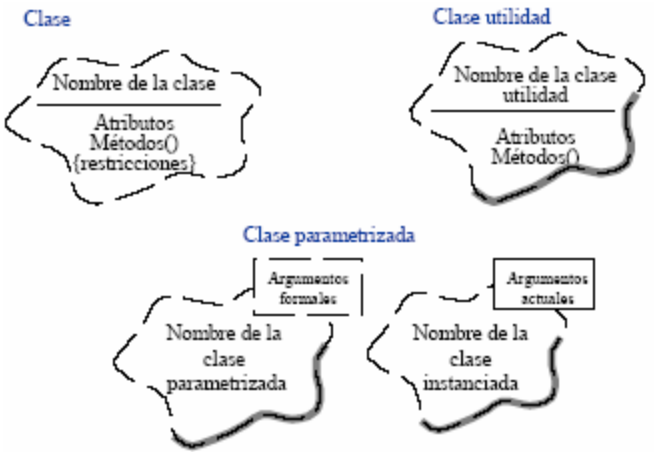


Figura 1.3 Representación de clases en la metodología Booch

Objectory, por Ivar Jacobson

Objectory es un método que se diferencia de otros por la importancia que da al caso de uso, una descripción o escenario que describe como el usuario interactúa con el producto o sistema.

Analiza el sistema en términos de entidades (un modelo de objetos) y casos prácticos de escenarios que abarcan un comportamiento dinámico. Para la implementación, la funcionalidad se divide en servicios, que son grupos de requisitos funcionales relacionados. El diseño consiste en construir una arquitectura del sistema en términos de bloques modulares.

- ✓ *Modelo de Casos de Uso*: Se basa en la descripción de elementos o usuarios externos al sistema (actores) y funcionalidad del sistema (casos de uso).
- ✓ *Modelo de objetos*: Representa la estructura estática de objetos.
- ✓ *Diagrama de interacción*. Muestran la secuencia de eventos entre paquetes u objetos necesarios para realizar un caso de uso.
- ✓ *Diagrama de estado*. Muestra los estados internos de un objeto.

Algunos de los conceptos más importantes de esta metodología son:

- ✓ *Objeto Entidad*. Representa información del sistema que debe sobrevivir cierto período de tiempo.
- ✓ *Objeto de Interfaz*. Modela información y comportamiento que es dependiente de la interfaz actual del sistema.

- ✓ *Objeto de Control*. Modela funcionalidad que no corresponde a ningún objeto en particular y que se presenta en algunos casos de uso. Estos objetos generalmente operan sobre varios objetos entidad, realizan algún algoritmo y retornan algún resultado a un objeto de interfaz.

La siguiente tabla muestra las diferentes etapas de esta metodología:

	Actividades	**Definición de entregas**
Análisis de Requerimientos	Especificación de requerimientos del usuario, en términos de casos de uso. Discusión y validación de requerimientos. Identificación detallada de cada caso de uso, describiendo la funcionalidad por defecto, las posibles variantes y los posibles errores. Definición de un borrador de la interfaz al usuario del sistema, que muestre cómo se verían los distintos casos de uso.	Modelo de Casos de Uso con interfaces de usuario del sistema. Modelo de objetos del dominio.
Modelo de análisis	Definir el modelo de análisis, identificando objetos entidad, de interfaz y de control; independientes del ambiente de implementación. Toda la funcionalidad que es dependiente del entorno del sistema se expresa en objetos de interfaz. Cada objeto de interfaz traduce acciones de los actores en eventos dentro del sistema y traducir los eventos del sistema en algo visible por el actor. Modelar la información (y comportamiento) que el usuario necesitará por largo tiempo en objetos entidad. Solo los objetos que sean justificados por casos de uso que los requieran son incluidos en el modelo. Modelar objetos de control cuando el sistema sea lo suficientemente complicado para tener funcionalidad que no corresponde a ningún objeto de interfaz ni a ningún objeto entidad.	Modelo de objetos con objetos Entidades, de Interfaz y de Control
Modelo de diseño	Refinar las clases de análisis para incluir detalles de implementación. Desarrollar el código de los métodos de los objetos. Realizar diagramas de interacción que muestran como interactúan las distintas clases en el desarrollo de un caso de uso. Desarrollar diagramas de estado para los objetos complejos.	Es un refinamiento y formalización del modelo de análisis. Su principal objetivo es adecuar el modelo de análisis al ambiente de implementación. Definición de módulos en la implementación y detalle de las clases de diseño en cada uno de ellos.
Implementación	Implementar las clases de diseño definidas. Las clases en implementación deben tener las siguientes características: Robustas y altamente reusables. No deben ofrecer funcionalidad similar a menos que estén relacionadas por herencia Altamente cohesivas. Funcionalidad interna altamente relacionada	Código de las clases, organizado por módulos.

	Actividades	**Definición de entregas**
Pruebas	Definir las unidades a probar y las pruebas que cubran la mayor cantidad de código. Utilizar los casos de uso como guía de prueba. Definir las pruebas para cada clase.	Definición de pruebas. Definición de pruebas de protocolo de clases.

Tabla 2 Etapas de la metodología Jacobson

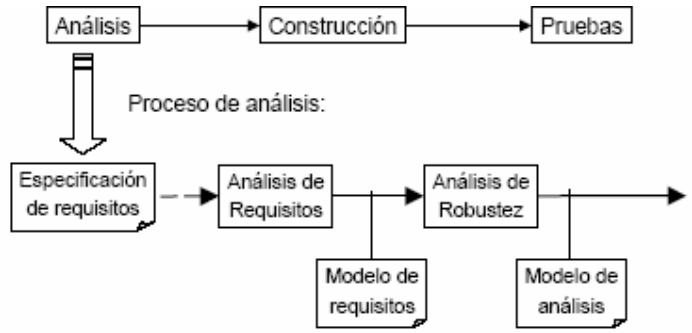


Figura 1.4 Ciclo de vida de la metodología Jacobson

Object Modeling Technique (Técnica de modelado de objetos), por James Rumbaugh

Es una metodología orientada a objeto muy difundida, de hecho es la que actualmente más se está utilizando por encima incluso de la de Booch. Se desarrolló en el "Research and Development Center" de la empresa General Electric a finales de los 80. Se hace cargo de todo el ciclo de vida del software dividiéndolo en 4 fases: análisis de objetos, diseño del sistema, diseño de objetos e implementación, y durante ese tiempo mantiene la misma notación.

- ✓ *Modelo de Objetos*. Se define como un diagrama de objetos además de un diccionario de datos. El diagrama de objetos muestra las clases y sus relaciones (generalización, agregación, asociación, instanciación). El diccionario de datos es el detalle de las clases en el diagrama de objetos.
- ✓ *Modelo dinámico*. Se define como un conjunto de diagramas de estado más un diagrama de Flujo de eventos Global.
- ✓ *Modelo funcional*. Es un diagrama de flujo con restricciones.

	Actividades	**Definición de entregas**
Análisis	Obtener una descripción del problema. Construir el modelo de objetos. Construir el modelo dinámico. Construir el modelo funcional. Verificar, iterar y refinar los tres modelos.	Documento de análisis, que incluye: Descripción del problema. Modelo de Objetos. Modelo dinámico. Modelo funcional.
Diseño del sistema	Organizar el sistema en subsistemas. Identificar concurrencia inherente al problema. Escoger una estrategia para la implementación de almacenamiento de datos. Determinar los mecanismos para controlar el acceso a recursos globales. Escoger la implementación del control del software. Manejar condiciones de frontera. Establecer prioridades.	Definición de subsistemas.
Diseño de objetos	Obtener operaciones de los modelos funcional y dinámico. Diseñar algoritmos para realizar las operaciones. Optimizar los caminos de acceso a los datos. Implementar el control del software. Ajustar la estructura de clases para incrementar herencia. Diseñar implementación de asociaciones. Determinar la representación de los atributos de las clases. Agrupar clases y asociaciones en módulos.	Documento de diseño, que incluye versiones detalladas de los modelos de objetos, dinámico y funcional.

Implementación	Diseñar bases de datos. Codificar.	Diseño de bases de datos, si se requieren. Código.

Tabla 3 Etapas de la metodología Rumbaugh

1.4. Introducción a la programación orientada a objetos

La programación Orientada a Objetos (POO) es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos, intenta simular el mundo real a través del significado de objetos que contiene características y funciones. La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes procedurales tradicionales, en los que los datos y los procedimientos están separados y sin relación. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

1.4.1. Conceptos de programación orientado a objetos (Object Oriented Programming, OOP)

Un lenguaje orientado a objetos puede reconocerse por siete características: identidad, abstracción, clasificación, encapsulamiento, herencia, polimorfismo y persistencia. Algunos lenguajes sólo utilizan un subconjunto compuesto por parte de estas características, no obstante se les otorga la denominación de basados en objetos.

Figura 1.5 Características de la OOP

- ✓ *Identidad*: Esta característica se refiere al hecho de que los datos son organizados en entidades discretas, distinguibles, denominadas objetos. Un objeto simple tiene estado y comportamiento asociados, además de tener un *nombre* (también denominado *referencia* o *handle*) que lo distingue de otros objetos.
- ✓ *Abstracción*: Ayuda a representar los diferentes puntos de vista incorporados en el sistema que se desarrolla y como se relacionan entre sí.
- ✓ *Clasificación*: Se utiliza para agrupar objetos que tienen atributos y comportamientos en común. Se dice que un objeto es una *instancia* de una clase, cada clase tiene sus propios atributos (es decir, tienen formas de describir su propio estado en cualquier instante de tiempo), pero comparte nombre y comportamientos con las otras instancias de la clase. Así una clase describe un conjunto de objetos que comparten una estructura común y tienen comportamientos comunes, pero los valores de los atributos nos ayudan a distinguir cada objeto particular de otro.
- ✓ *Encapsulamiento*: Una clase encapsula los atributos y comportamiento de un objeto, ocultando los detalles de implementación. Sin embargo, no es lo mismo que ocultamiento de información, ya que de lo que se trata es de empaquetar la información que debe ser visible y la que debe ocultarse.
- ✓ *Herencia*: Las clases pueden ser organizadas jerárquicamente de acuerdo con las semejanzas y diferencias entre ellas; para construir la jerarquía se comienza definiendo ampliamente una clase (superclase) para refinarla luego en clases más especializadas (subclases). Una subclase puede heredar la estructura así como el comportamiento y atributos de su superclase.
- ✓ *Polimorfismo*: Un comportamiento es una acción o transformación que un objeto realiza, en ocasiones el mismo comportamiento se manifiesta de manera diferente en diferentes clases o subclases, a esto se le llama polimorfismo. Una implementación específica de una operación para una cierta clase se denomina *método*, en un sistema con polimorfismo una operación puede tener más de un método que la implemente.
- ✓ *Persistencia*: La capacidad del nombre, estados y comportamientos de un objeto para trascender el espacio o el tiempo; en otras palabras el nombre, estado y comportamiento de un objeto se conserva cuando el objeto es transformado.

1.5. Lenguajes orientados a objetos

Numerosos lenguajes de programación han contribuido a la evolución de los lenguajes orientados a objetos, comenzando en la década de los 50's con LISP que es un lenguaje de inteligencia artificial que introdujo el concepto de enlace dinámico y un entorno de desarrollo interactivo. Simula, desarrollado en los 60's es un lenguaje diseñado para hacer simulaciones, contribuyó con el concepto de clase y los mecanismos de herencia. La abstracción de datos fue introducida a principios de los 70's,

primero en le ámbito académico como CLU y más tarde comercialmente en lenguajes tan populares como Ada y Modula-2.

El principal impulsor comercial de los lenguajes orientados a objetos fue Smalltalk desarrollado por Xerox PARC, en él se toma el concepto de clase de Simula-67 y se diseña para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos. Samalltalk fue creciendo a través de varias versiones hasta llegar a Smalltalk- 80 en 1981.

A partir de ese momento el progreso de los lenguajes orientados a objetos ha sido acelerado, por una lado debido a la disponibilidad de la extensiones orientadas a objetos de lenguajes comercialmente populares como C, Pascal, COBOL y BASIC. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para los cuales la programación orientada a objetos está particularmente bien adaptada.

En la década de los 90's emergen dos de las principales tendencias de los lenguajes orientados a objetos; una de estas es el grupo de los lenguajes orientados a objetos "puros", donde casi todo es un objeto sin embargo muchos de ellos carecían de las características de las cuales muchos programadores habían venido dependiendo. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características procedurales de maneras "seguras", formando de esta manera un grupo híbrido. El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet, para la cual Java tiene características especiales.

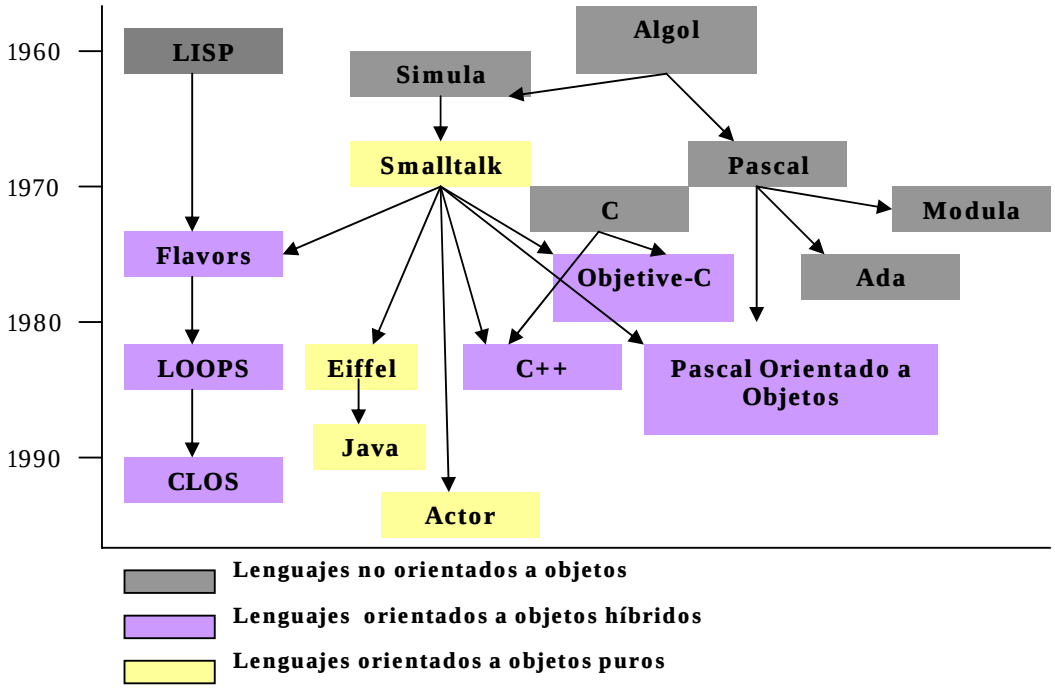


Figura 1.6 Evolución de los lenguajes orientados a objetos

1.5.1. Beneficios de los lenguajes orientados a objetos

Un lenguaje de programación que soporta el paradigma de la programación orientada a objetos beneficia al desarrollador de software en el sentido de que le ofrece la posibilidad de modelar el mundo real, pues se permite el desarrollo modular. Los objetos hacen posible mantener los elementos relacionados juntos y, en particular, mantener datos y métodos que actúan sobre éstos juntos. Conforme se tenga en un solo lugar todas las partes importantes de un programa que se relacionan con una entidad en particular, el mantenimiento y la actualización se simplifica.

En contraste la programación procedimental clásica presenta ciertos problemas, que han ido haciéndose cada vez más graves, a medida que se construían aplicaciones y sistemas informáticos más complejos, entre los que destacan los siguientes:

- ✓ *Modelo mental anómalo*. Nuestra imagen del mundo se apoya en los seres, a los que asignamos nombres sustantivos, mientras la programación clásica se basa en el comportamiento, representado usualmente por verbos.
- ✓ *Es difícil modificar y extender los programas*, pues suele haber datos compartidos por varios subprogramas, que introducen interacciones ocultas entre ellos.
- ✓ *Es difícil mantener los programas*. Casi todos los sistemas informáticos grandes tienen errores ocultos, que no surgen a la luz hasta después de muchas horas de funcionamiento.
- ✓ *Es difícil reutilizar los programas*. Es prácticamente imposible aprovechar en una aplicación nueva las subrutinas que se diseñaron para otra.

La OOP proporciona las siguientes ventajas sobre otros lenguajes de programación:

- ✓ *Uniformidad*. Ya que la representación de los objetos lleva implícita tanto el análisis como el diseño y la codificación de los mismos.
- ✓ *Comprensión*. Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que corresponden con las estructuras de información que el programa trata.
- ✓ *Flexibilidad*. Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.
- ✓ *Estabilidad*. Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.

- ✓ *Reusabilidad*. La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

Metodología y Modelado del Sistema

Capítulo II

METODOLOGÍA Y MODELADO DEL SISTEMA (SAPIUN)

2.2. Metodología OMT

La metodología OMT (Object Modeling Technique, Técnica de Modelado de Objetos) fue creada por James Rumbaugh y Michael Blaha en 1991, mientras James dirigía un equipo de investigación de los laboratorios General Electric.

OMT es una de las metodologías de análisis y diseño orientada a objetos, más maduras y eficientes que existen en la actualidad. Pone énfasis en la importancia del modelo y su uso para lograr una abstracción, en el cual el análisis esta enfocado al mundo real, también pone detalles particulares para modelado de recursos de la computadora. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.).

OMT se basa en tres modelos fundamentales y relacionados entre sí:

- ✓ *Modelo de objetos*. Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones). El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos conceptos del mundo real que sean importantes para la aplicación.
- ✓ *Modelo dinámico*. Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas.
- ✓ *Modelo funcional*. Describe las transformaciones de valores de datos (funciones, correspondencias, restricciones y dependencias funcionales) que ocurren dentro del sistema. Captura lo que hace el sistema, independientemente de cuando se haga o de la forma en que se haga.
- ✓ Una de las grandes virtudes de OMT es su modelo de objetos, que contiene una enorme riqueza semántica, por lo que ha sido adaptado por casi todas las metodologías de segunda generación, y es una de las bases metodológicas de la metodología *Objetory*[*].

[*] Ver capítulo I página 12

OMT divide el ciclo de vida del software en 4 fases: Análisis de objetos, diseño del sistema, diseño de objetos e implementación.

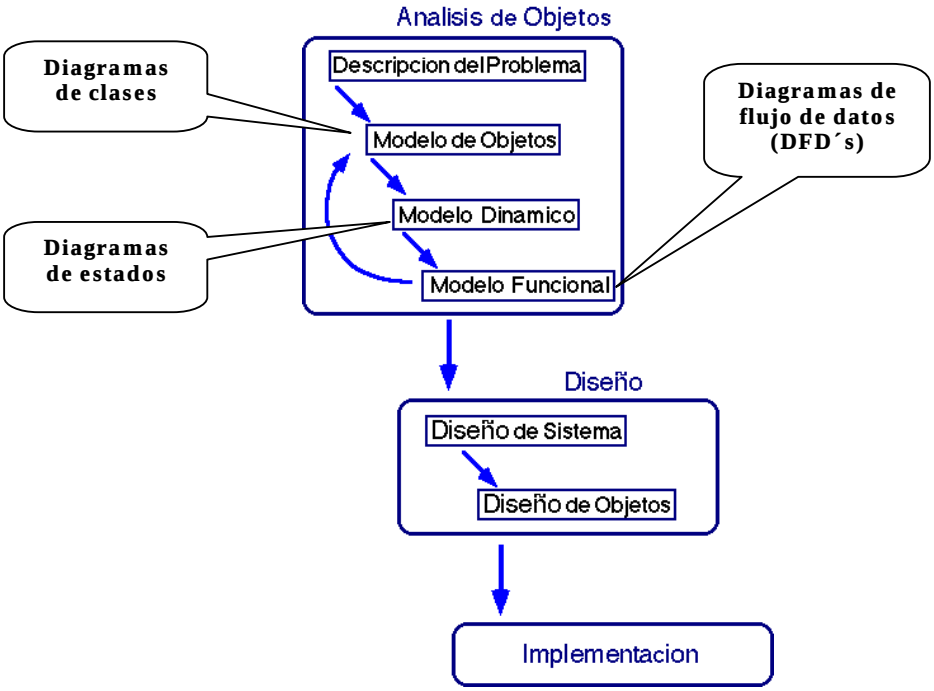


Figura 2.1 Ciclo de vida propuesto en OMT

Análisis de objetos

El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras de datos. Un buen modelo debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos.

En primer lugar, se describe el problema como una solución genérica, más o menos como una plantilla o un resumen tal que no especifica la solución propiamente dicha sino que se describe el comportamiento del sistema como una "caja negra". Para que dicha solución sea la más óptima deberá haber una gran interacción entre el analista, y el grupo de usuarios o grupo del dominio de problema.

En segundo lugar se hacen los diagramas de objetos con su diccionario de datos, en él se define la estructura de los objetos y clases así como las relaciones que los unen. A continuación se crea un modelo dinámico para describir los aspectos de control y evolución del sistema. Después se crea un modelo funcional que describe las principales funciones, los valores de entrada y salida, e imponga las restricciones pertinentes.

Por último se verifican todos los modelos creados y se itera para conseguir un refinamiento de los 3 modelos.

Resumiendo los pasos a seguir son:

- ✓ Obtener y escribir una descripción inicial del problema.
- ✓ Construir un modelo de objetos.
- ✓ Construir un modelo dinámico.
- ✓ Construir un modelo funcional.
- ✓ Verificar, iterar y refinar los tres modelos.

Diseño del sistema

El diseñador del sistema toma decisiones estratégicas sobre la estructura global del mismo. Durante esta fase el sistema se organiza en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta y se selecciona una estrategia para afrontar el problema.

Pasos a seguir:

- ✓ Organizar el sistema en subsistemas.
- ✓ Identificar la concurrencia inherente al problema.
- ✓ Colocar los subsistemas con sus procesos y tareas.
- ✓ Elegir la estrategia básica para almacenar los datos.
- ✓ Identificar los recursos globales y determinar mecanismos de control de acceso a ellos.
- ✓ Elegir un método de implementación del control de software.
- ✓ Considerar las condiciones límite.
- ✓ Establecer las prioridades.

Diseño de objetos

El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. Todo esto ha de hacerse con independencia del lenguaje o entorno en que finalmente se codifique y ejecute la aplicación.

Pasos a seguir:

- ✓ Obtener operaciones para el modelo de objetos a partir de los otros modelos.
- ✓ Diseñar algoritmos para implementar operaciones.
- ✓ Optimizar los accesos a datos.
- ✓ Implementar el control software mediante el sistema elegido durante el diseño del sistema.
- ✓ Ajustar las estructuras de las clases para aumentar la herencia.
- ✓ Diseño de la implementación de las asociaciones.
- ✓ Determinar la representación exacta de los atributos de los objetos.
- ✓ Compactar las clases y asociaciones en módulos, ocultando en la parte privada toda la información que deba estar oculta.

Implementación

Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los siguientes principios:

- ✓ *Fiable.* Un buen software tiene pocos errores.
- ✓ *Flexible.* Todos los cambios que se hacen en el software, después de ser entregado tradicionalmente se llama mantenimiento.
- ✓ *Accesible.* Tanto para la compra como para el mantenimiento, los costos de mano de obra son el elemento más significativo dentro del costo del software, de manera que si éste se reduce quiere decir que es relativamente sencillo, fácil de desarrollar y mantener.
- ✓ *Disponible.* El software tiene que poder ejecutarse en el hardware disponible, con el sistema operativo disponible, etc. Esto implica que el sistema debe ser lo suficientemente portable y de fácil mantenimiento, ya que debe poderse realizar cualquier cambio provocado por cambios en el entorno del software.

2.2.1. Procesos de la metodología OMT

Modelo de objetos

En este modelo se observan y reconocen los objetos y sus clases, cada objeto cumple un rol propio en el dominio del problema.

Los diagramas de objetos proporcionan una notación gráfica formal para el modelado de objetos, clases y sus relaciones mediante dos tipos de diagramas: los diagramas de clases y los diagramas de instancias. En los diagramas que componen este modelo se pueden representar los siguientes elementos del sistema: objetos y clases, atributos, operaciones, y relaciones o asociaciones.

Un objeto es, sencillamente, algo que tiene sentido en el contexto de la aplicación. Una clase describe un grupo de objetos con atributos similares y con relaciones comunes a otros. Los atributos son características que todos los objetos de una clase comparten. Las operaciones son las acciones que los objetos realizan sobre ellos mismos y sobre otros objetos que pertenecen a la misma clase; aunque también pueden añadir otras nuevas que no se definan en su clase a medida que se especializa el objeto en otras subclases. También pueden redefinir las operaciones en estas especializaciones ignorando las definiciones realizadas en las superclases.

Las operaciones pueden llevar implícito el objeto sobre el que se realizan o que realiza la acción, de forma que es posible tener una misma operación que se efectúe de manera distinta según el objeto sobre el que se aplique. La implementación de las operaciones para cada uno de los objetos diferentes (o subclases) se denomina método. Los métodos implementan en cada una de las clases de forma específica para los objetos que representa. Las relaciones entre clases determinan el comportamiento del sistema y constituyen una parte muy importante del mismo ya que mediante las relaciones se define la forma en que los objetos se comunican.

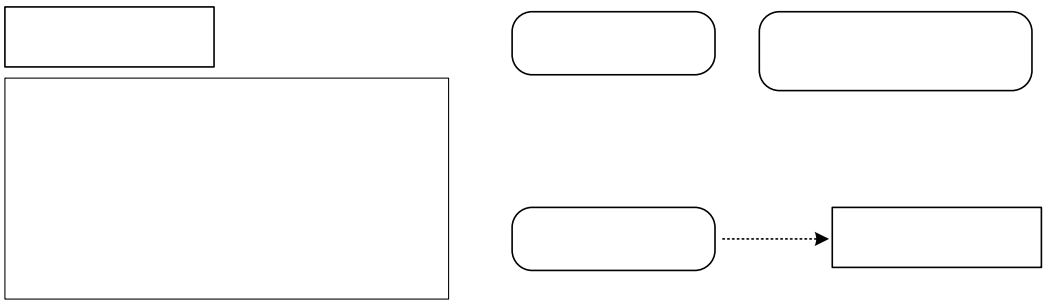


Figura 2.2 Diagrama de clases en OMT

Las relaciones tienen una serie de características que son de interés para el modelado del sistema. En OMT se identifican a través de enlaces: conexiones físicas o conceptuales entre casos concretos de objetos. Una asociación en OMT abstrae un conjunto de enlaces con una estructura y un significado comunes. Desde el punto de vista de la implementación, una asociación es un puntero que apunta desde un objeto a otro, es decir una relación binaria.

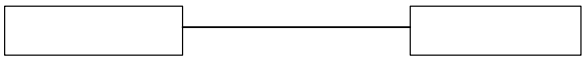


Figura 2.3 Diagrama de asociación en OMT

Las asociaciones muchos a muchos y uno a muchos pueden ser calificadas mediante un elemento calificador que reduce el conjunto de objetos relacionados, indicando un subconjunto de la clase en la asociación.

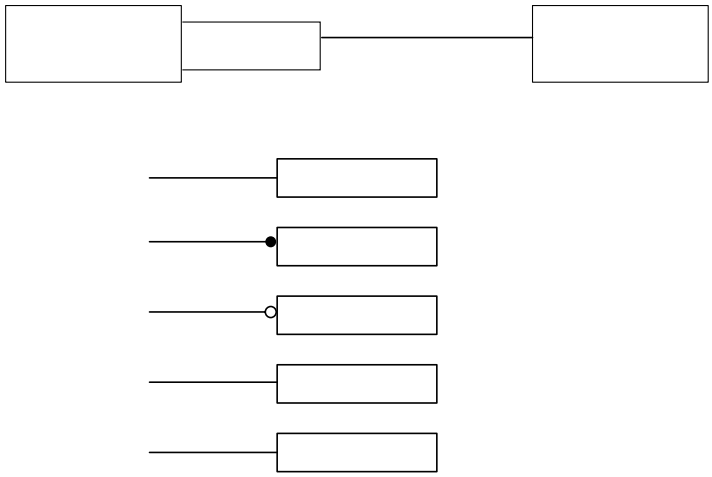


Figura 2.4 Diagrama de asociación cualificada en OMT

Las relaciones de agregación son para OMT formas de asociación del tipo "es parte de", como tales se definen entre una clase agregado y una clase componente y se indican con un rombo en la parte de la clase que actúa como contenedor. Las relaciones de agregación se establecen en los llamados objetos compuestos que contienen otros objetos.

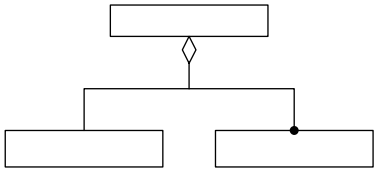


Figura 2.5 Diagrama de agregación en OMT

En resumen para la construcción de un modelo de objetos se requiere:

- ✓ Identificar las clases de objetos.
- ✓ Agregar atributos y asociaciones entre clases.
- ✓ Iniciar un diccionario de datos que contenga descripciones de clases, atributos y asociaciones.
- ✓ Organizar y simplificar las clases de objetos usando herencia.
- ✓ Probar las rutas de acceso usando escenarios e iterar los pasos anteriores según sea necesario.

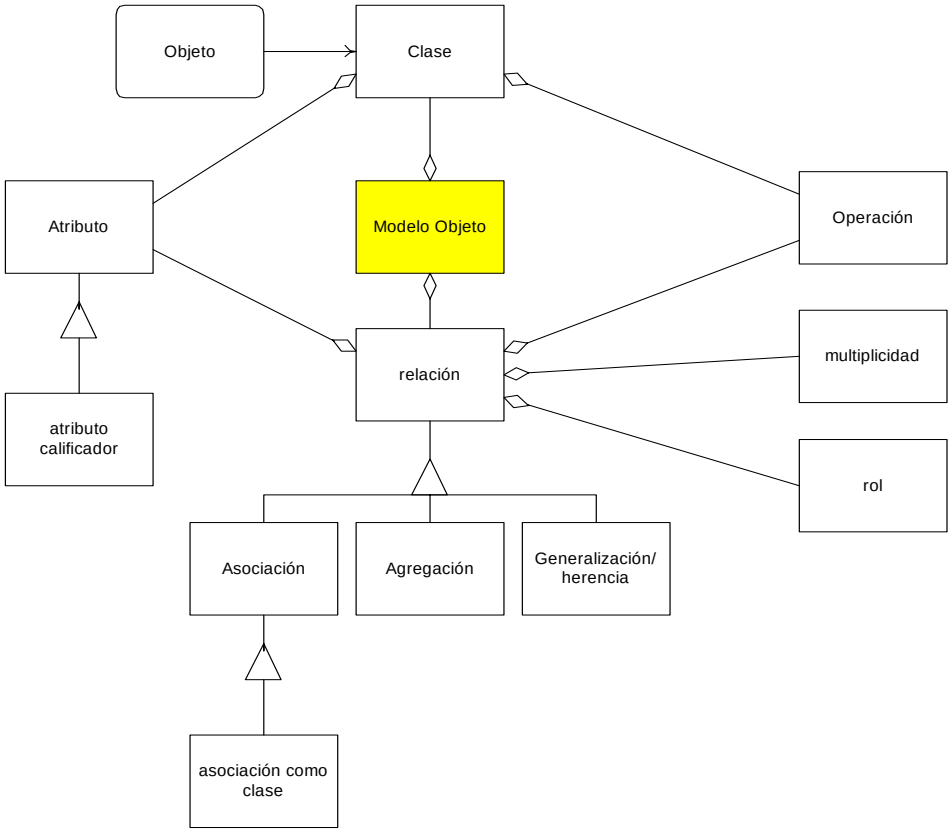


Figura 2.6 Modelo de objetos en OMT

Modelo Dinámico

El modelo dinámico está conformado por los aspectos del sistema que están relacionados con el tiempo y con los cambios. Los conceptos más importantes del modelado dinámico son los sucesos, que representan estímulos externos, y los estados, que representan los valores de los objetos. A lo largo del tiempo, los objetos se estimulan unos a otros, dando lugar a una serie de cambios en sus estados. Un estímulo individual proveniente de un objeto y que llega a otro es un suceso. La respuesta a un suceso depende del estado del objeto que lo recibe, y puede incluir un cambio de estado o el envío de otro suceso al remitente o a un tercer objeto. El diagrama de estados va a representar los sucesos y los estados que se dan en el sistema.

Un suceso, o evento, es una transmisión de información de dirección única entre un objeto y otro que transcurre durante un período de tiempo. Puede preceder o seguir lógicamente a otro, o bien los dos sucesos pueden no estar relacionados. Un escenario es una secuencia de sucesos que se produce durante una ejecución concreta de un sistema. El ámbito de un escenario es variable; puede incluir todos los sucesos del sistema, o que sean generados por ellos. El primer paso para la construcción de escenario es identificar los objetos emisores y receptores de cada suceso. La secuencia de sucesos y los objetos que intercambian sucesos se pueden mostrar ambos en un escenario mejorado que se denomina de seguimiento de sucesos. El tiempo aumenta desde arriba hacia abajo, aunque en el seguimiento de sucesos no importa la temporización exacta, sino que importa la secuencia de los procesos.

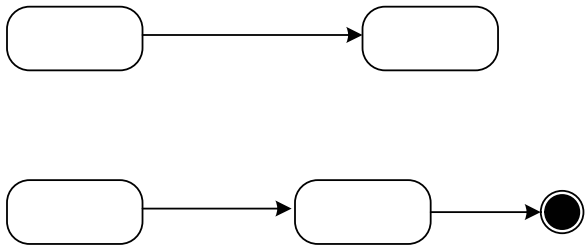


Figura 2.7 Diagrama de sucesos en OMT

Un estado es una abstracción de los valores de los atributos y de los enlaces de un objeto. Los conjuntos de valores se agrupan dentro del estado de acuerdo con aquellas propiedades que afecten al comportamiento del objeto, en respuesta a los sucesos entrantes. La respuesta de un objeto a un suceso puede incluir una acción o un cambio de estado por parte del mismo. Un estado corresponde al intervalo entre dos sucesos recibidos por un objeto. Tanto los sucesos como los estados dependen del nivel de abstracción utilizado. Los estados se pueden

caracterizar de diferentes maneras, pero normalmente cada estado tiene un nombre y una descripción en la que se indica en la situación en la que se encuentra el sistema.

Los estados pueden poseer subestados que hereden las transiciones de sus superestados, del mismo modo que las clases poseen subclases que heredan los atributos y operaciones de sus superclases.

Figura 2.8 Diagrama de estados en OMT

Una actividad es una operación cuya realización requiere un cierto tiempo y se encuentra asociada a un estado. Entre las actividades se cuentan las operaciones continuas, así como las operaciones secuenciales que terminan por sí mismas después de un cierto intervalo de tiempo. Un estado puede controlar una actividad continua o una actividad secuencial que va avanzando hasta que termina o hasta que se produce un suceso que la hace finalizar prematuramente. Las acciones también pueden representar operaciones internas de control, tales como dar valores a atributos o generar otros sucesos. Estas acciones no tienen afectación en el mundo real, sino que son mecanismos para estructurar el control dentro de una implementación.

Una actividad de un estado se puede expandir en forma de diagrama de estados de nivel inferior, en el cual cada uno representará un paso de la actividad. Las actividades anidadas son diagramas de estados de un solo uso, con transiciones de entrada y de salida parecidas a subrutinas.

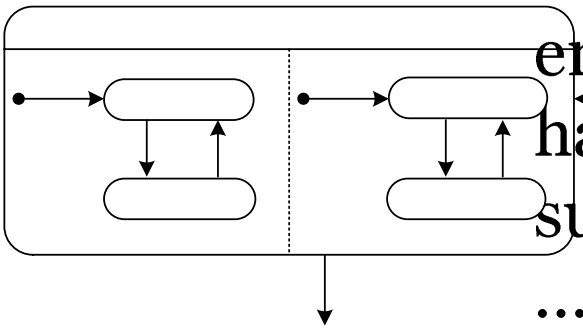


Figura 2.9 Diagrama de subestados en OMT

Finalmente para la construcción de un modelo dinámico es necesario:

- ✓ Preparar escenarios para las secuencias de interacción típicas.
- ✓ Identificar eventos entre objetos y preparar trazos de eventos para cada escenario.
- ✓ Preparar un diagrama de flujo de eventos para el sistema.
- ✓ Desarrollar un diagrama de estados para cada clase que tenga un comportamiento dinámico importante.
- ✓ Verificar que los eventos compartidos entre diagramas de estado sean consistentes y correctos.

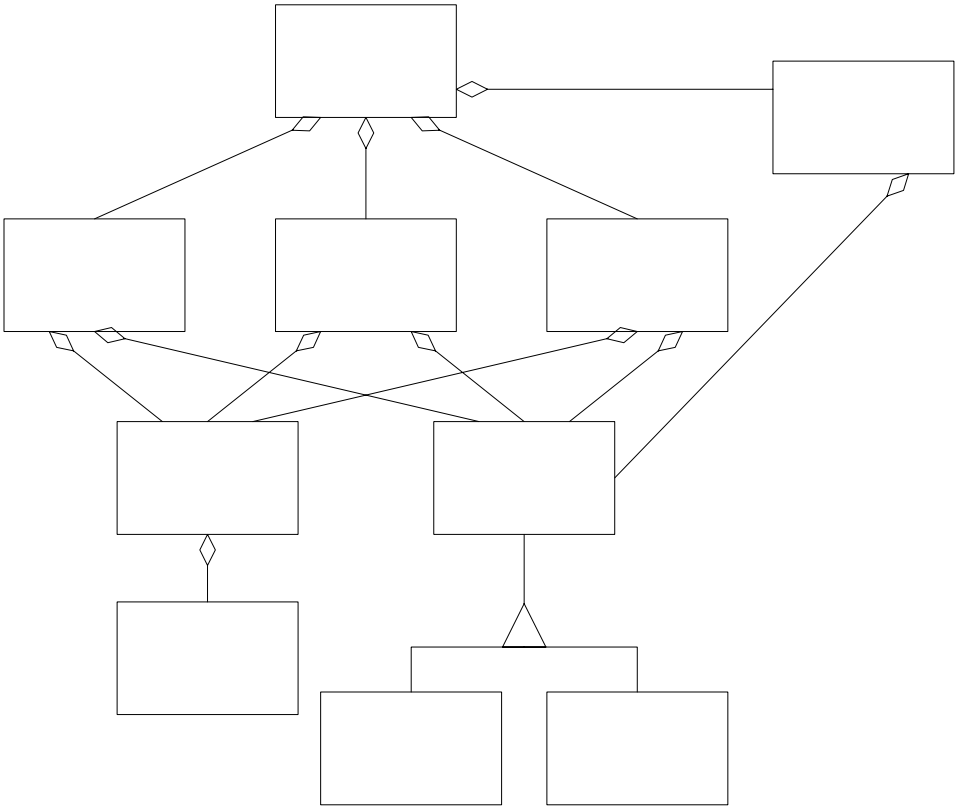


Figura 2.10 Modelo dinámico en OMT

Modelo Funcional

El modelo funcional contiene diagramas de flujo de datos (DFD). Un DFD muestra las relaciones funcionales entre los valores calculados por un sistema, incluyendo los valores introducidos, los obtenidos, y los almacenes internos de datos. Es un grafo que muestra el flujo de valores de datos desde sus fuentes en los objetos mediante procesos que los transforman, hasta sus destinos en otros objetos sin mostrar información de control. Las funciones se invocan como acciones en el modelo dinámico y se muestran como operaciones que afectan a objetos en el modelo de objetos.

Un DFD contiene procesos que transforman datos, flujos de datos que los trasladan, objetos actores que producen y consumen datos y almacenes de datos.

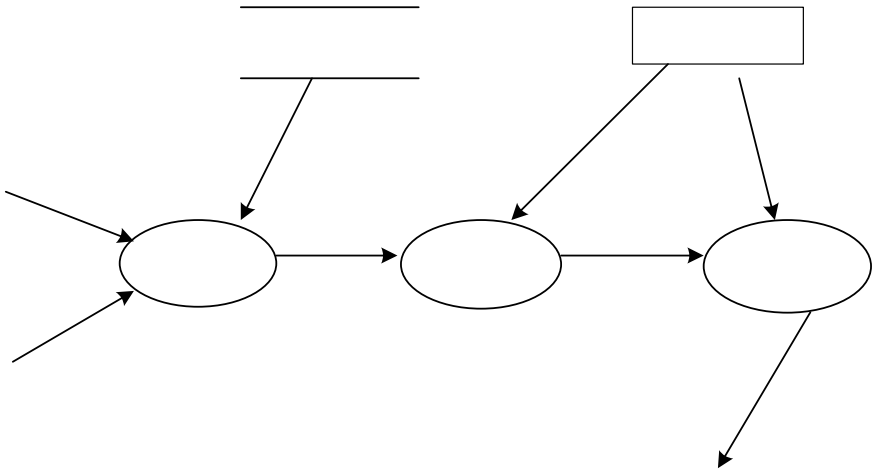


Figura 2.11 Diagrama de flujo de datos en OMT

Un proceso transforma valores de datos. Los procesos del más bajo nivel son funciones puras, sin efectos laterales. Un grafo completo de flujo de datos es un proceso de alto nivel. Los procesos pueden tener efectos laterales si contienen componentes no funcionales, tales como almacenes de datos u objetos externos. Los resultados de estos procesos dependen del comportamiento del sistema, según se especifica en el modelo dinámico Los procesos se dibujan en forma de elipse que contienen una descripción de la transformación, normalmente su nombre. Cada proceso tiene un número fijo de flechas de entrada y salida de datos, cada una de las cuales lleva un valor de un tipo dado. Las entradas y salidas se pueden rotular, para mostrar su papel en el cálculo, pero es frecuente que baste el tipo de valor asociado al flujo de datos.

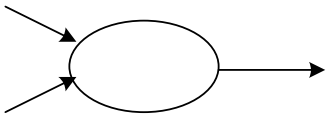


Figura 2.12 Diagrama de proceso en OMT

Un actor es un objeto activo que controla el grafo de flujo de datos produciendo o consumiendo valores. Los actores están asociados a las entradas y salidas del grafo de flujo de datos. En cierto sentido, los actores yacen en la frontera del grafo, pero hacen que concluya el flujo de datos como consumidores (sumideros) y productores (fuentes) de datos, así que en algunos casos se denominan terminadores. Las flechas entre el actor y el diagrama son las entradas y salidas del diagrama.

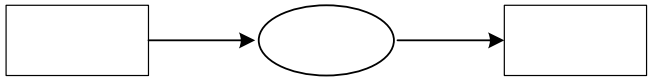


Figura 2.13 Diagrama de actor en OMT

Un almacén de datos es un objeto pasivo dentro de un diagrama de ‡ujo de datos que almacena datos para su posterior utilización. Di...eren con los atores en que no generan ninguna operación por si mismos, sino que se limitan a responder a solicitudes de almacenamiento y acceso a datos. Los almacenes se dibujan en forma de un par de líneas paralelas que contienen el nombre del almacén. Las ‡echas de entradas indican información u operaciones que modifican los datos almacenados; esto incluye la adición, modificación o el borrado de elementos. Las flechas de salidas indican información que se ha extraído del almacén.

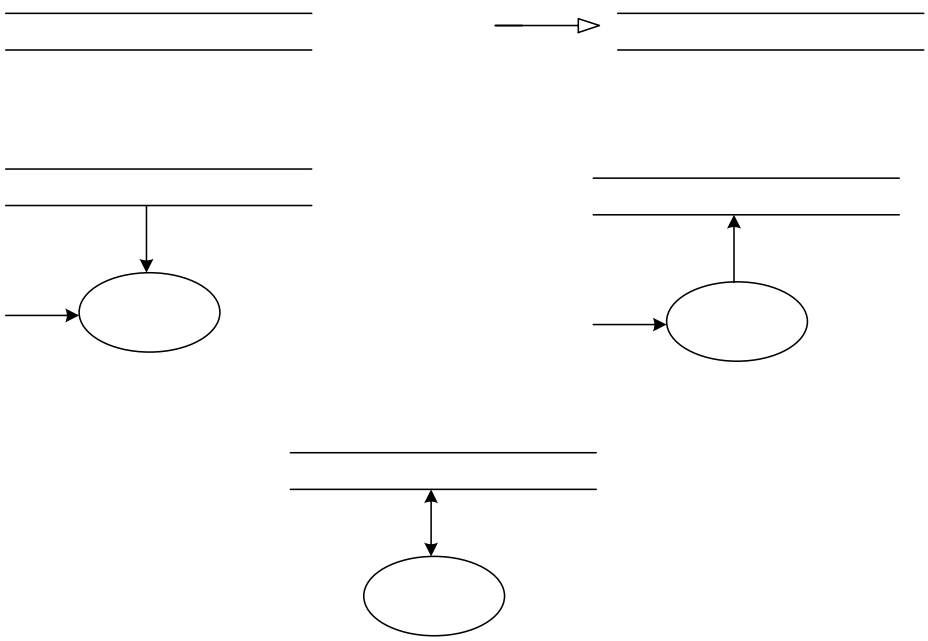


Figura 2.14 Diagrama de almacén de datos en OMT

Tanto los actores como los almacenes de datos son objetos. Se diferencian en que su comportamiento y utilización suelen ser distintos, aun cuando en un lenguaje orientado a objetos ambos pudieran ser implementados como objetos. También el almacén puede ser implementado como un archivo y el terminador como un dispositivo externo.

Un DFD resulta especialmente útil para mostrar funcionalidad de alto nivel de un sistema, y su descomposición en unidades funcionales más pequeñas. Un proceso se puede expandir en otro diagrama de flujo de datos. Todas las entradas y salidas del proceso lo serán también del nuevo diagrama, también pueden tener almacenes de datos que no se muestren en el diagrama de nivel superior.

Un diagrama de flujo de control muestra todas las posibles vías de computación para los valores. No muestra cuales son las vías que se ejecutan ni en que orden. Las decisiones y las secuencias son problemas de control, que forman parte del modelo dinámico. Una decisión afecta a sí una o más funciones llegan incluso a ejecutarse, en lugar de proporcionarles un valor. Aun cuando las funciones no poseen valores de entrada procedentes de estas funciones de decisión, a veces resulta útil incluirlas en este modelo funcional, para que no se olviden, y para que sea posible mostrar sus dependencias de datos. Esto se hace incluyendo flujos de control en el diagrama de flujo de datos. Un flujo de control es un valor booleano que afecta a sí un proceso es o no evaluado. Los flujos de control se muestran en los diagramas con una línea discontinua que va desde un proceso que produce un valor booleano hasta el que se esta controlando.

Los procesos de los diagramas de flujo deben ser implementados eventualmente como operaciones que se aplican a objetos. Toda operación se podrá especificar de diferentes maneras, entre las que se cuentan las siguientes:

- ✓ Funciones matemáticas, tales como las funciones trigonométricas.
- ✓ Tablas de valores de entrada y salida para pequeños conjuntos finitos.
- ✓ Ecuaciones que especifican la salida en términos de la entrada.
- ✓ Condiciones previas y posteriores.
- ✓ Tablas de decisión.
- ✓ Pseudocodigo.
- ✓ Lenguaje natural.
- ✓ Diagrama de flujos.
- ✓ Diagrama de árboles.
- ✓ etc.

Una restricción muestra la relación entre dos objetos al mismo tiempo o bien entre distintos valores del mismo objeto en instantes diferentes. Las restricciones se pueden expresar como una función total (un valor que es especificado completamente por otro) o como una función parcial

(un valor que está restringido, pero no completamente especificado por otro valor). Las restricciones pueden aparecer en todas las clases del modelo.

Las restricciones de objetos especifican que algunos objetos dependen entera o parcialmente de otros objetos. Las restricciones dinámicas especifican relaciones entre los estados o sucesos de distintos objetos. Las restricciones funcionales especifican limitaciones aplicables a operaciones. Una restricción entre valores de un objeto a lo largo del tiempo es lo que suele denominarse un invariante.

En conclusión los pasos a seguir para la construcción de un modelo funcional son:

- ✓ Identificar valores de entrada y salida.
- ✓ Usar diagramas de flujo de datos para mostrar dependencias funcionales.
- ✓ Describir las funciones.
- ✓ Identificar restricciones.

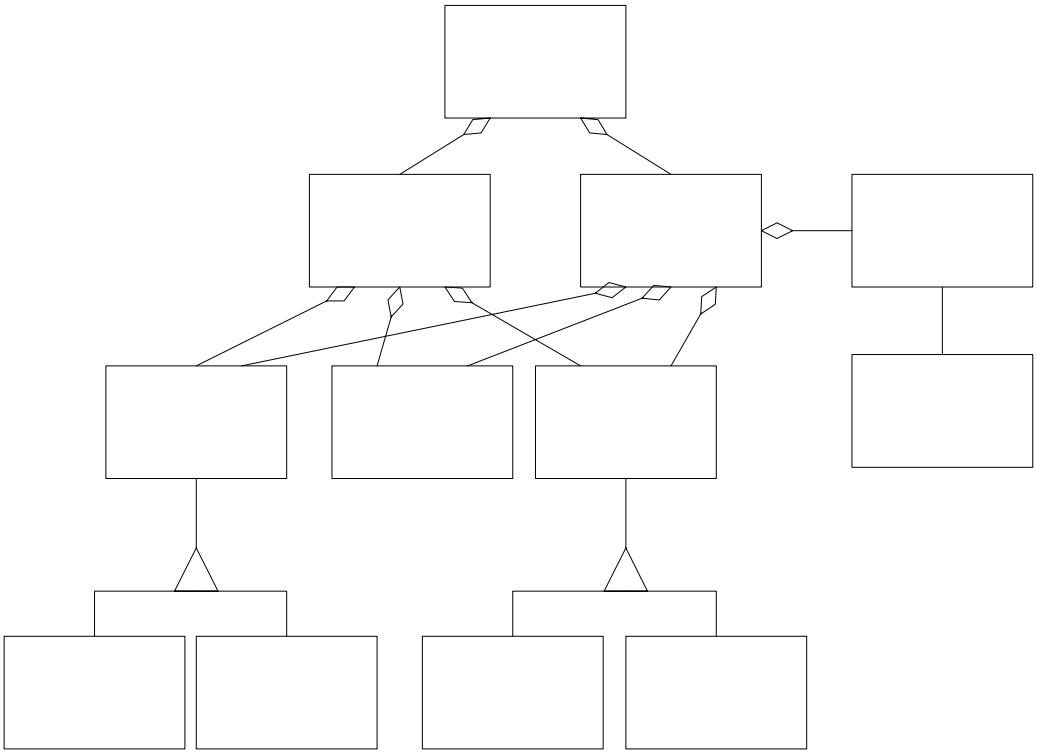


Figura 2.15 Modelo Fucional en OMT

2.2 Lenguaje de modelado unificado (UML)

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Puede ser adaptado para encajar en diferentes situaciones de desarrollo y ciclos de vida de software. De hecho, se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y James Rumbaugh.

Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE*. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle e IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa: Booch, Objectory y OMT. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

El objetivo principal cuando se empezó a gestar UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común.

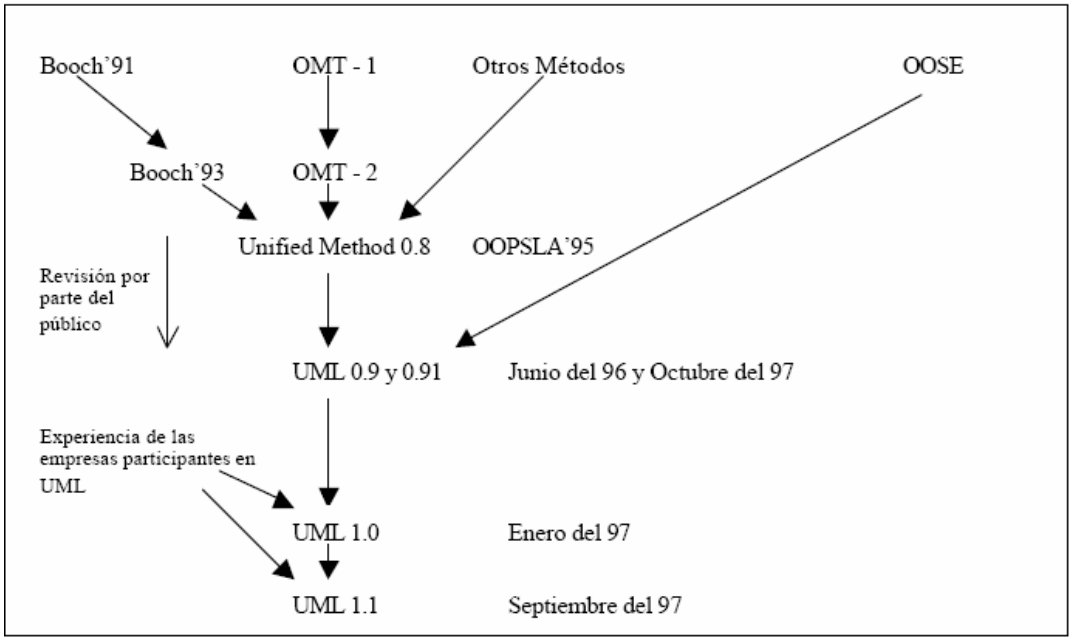


Figura 2.16 Evolución de UML

*Son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación.

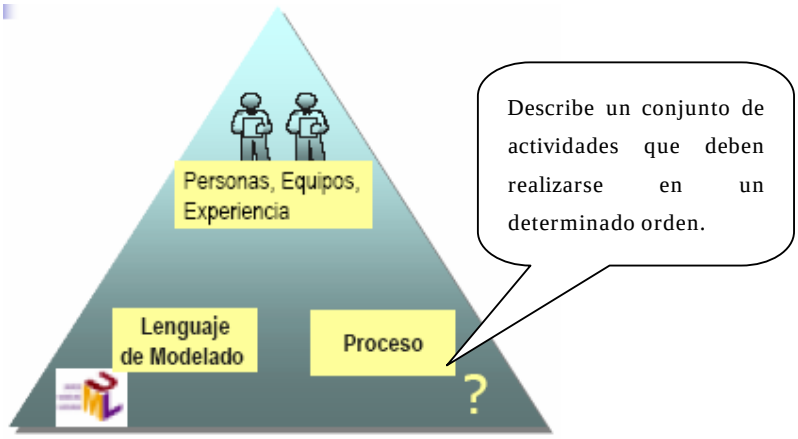


Figura 2.16 Evolución de UML

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

- ✓ *Diagramas de comportamiento*: Especifican las partes dinámicas de un sistema tales como estados del sistema, flujo de control de actividades, secuencia de mensajes, etc.
- ✓ *Diagramas de clases:* Conjunto de clases, interfaces y colaboraciones, y las relaciones entre ellas.
- ✓ *Diagramas de objetos*: Instantáneas de las instancias de los elementos encontrados en los diagramas de clases.
- ✓ *Diagramas de componentes*: Conjunto de componentes y sus relaciones.
- ✓ *Diagramas de despliegue*: Conjunto de nodos y sus relaciones.
- ✓ *Diagramas de casos de uso*: Conjunto de casos de uso y actores y sus relaciones. Son importantes para organizar y modelar el sistema.
- ✓ Diagramas de interacción:
    - o *Diagramas de secuencia*: Conjunto de objetos y los mensajes enviados y recibidos por ellos. Resalta ordenación temporal de los mensajes.
    - o *Diagramas de colaboración*: Resalta organización estructural de objetos que envían y reciben mensajes.
- ✓ *Diagramas de estados*: Representan máquinas de estados, construida por estados, transiciones, eventos y actividades. Útiles para modelar sistemas reactivos.

- ✓ *Diagramas de actividades*: Muestran el flujo de actividades de un sistema. Importantes para modelar la función de un sistema, así como para resaltar el flujo de control entre objetos.

Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas. Las vistas existentes en UML son:

- ✓ *Vista casos de uso*: Muestra el comportamiento del sistema tal y como es percibido por usuarios, analistas y encargados de pruebas. Se forma con los diagramas de casos de uso, colaboración, estados y actividades.
- ✓ *Vista de diseño*: Comprende el vocabulario del problema y su solución, y soporta los requisitos funcionales del sistema (servicios que el sistema debería proporcionar a los usuarios finales). Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.
- ✓ *Vista de procesos*: Hilos y procesos que forman mecanismos de sincronización y concurrencia del sistema. Se forma con los diagramas de la vista de diseño; recalcando las clases y objetos referentes a procesos.
- ✓ *Vista de implementación*: Componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. Se forma con los diagramas de componentes, colaboración, estados y actividades.
- ✓ *Vista de despliegue*: Nodos que forman la topología hardware sobre la que se ejecuta el sistema. Distribución, entrega e instalación de las partes. Se forma con los diagramas de despliegue, interacción, estados y actividades.

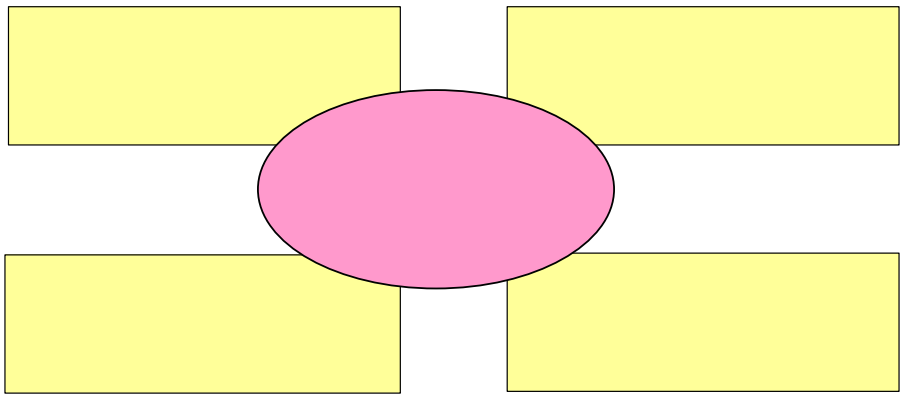


Figura 2.17 Relación entre vistas de UML

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

UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica.

2.2.1. Relación entre UML y OMT

Tanto OMT como UML son metodologías abiertas, no propietarias, intentan solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos con el cuál se crea una documentación también común, que cualquier desarrollador con conocimientos de OMT/UML será capaz de entender, independientemente del lenguaje, arquitectura y sistema operativo utilizado para el proyecto.

UML define el lenguaje formal mediante el cuál se representa el modelado a objetos pero los principales conceptos de análisis y diseño fueron aportados por OMT, ella se concentra en la abstracción del problema. Por tal motivo resulta imprescindible entender los fundamentos aportados por OMT.

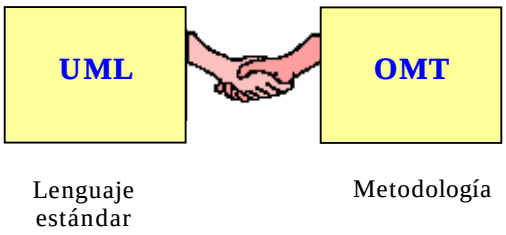


Figura 2.18 Relación de UML y OMT

La utilización de OMT y UML nos permite realizar todos los procesos mentales para comprender el problema a resolver mucho antes de empezar a lidiar con problemas específicos de implementación. Otra ventaja es una exhaustiva documentación que si fue actualizada en forma permanente durante el proyecto tiene una correlatividad precisa con el código orientado a objetos que tenemos como resultado final de todo este proceso.

2.2.2. Diagramas de UML

Con el nombre de Diagramas de Estructura Estática se engloba tanto al Modelo Conceptual de la fase de Análisis como al Diagrama de Clases de la fase de Diseño. Ambos son distintos conceptualmente, mientras el primero modela elementos del dominio el segundo presenta los elementos de la solución software. Sin embargo, ambos comparten la misma

notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos (asociaciones).

A continuación se muestran los elementos comunes que integran los diagramas de UML:

Una ***clase*** se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática (plegada), con los detalles como atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase.

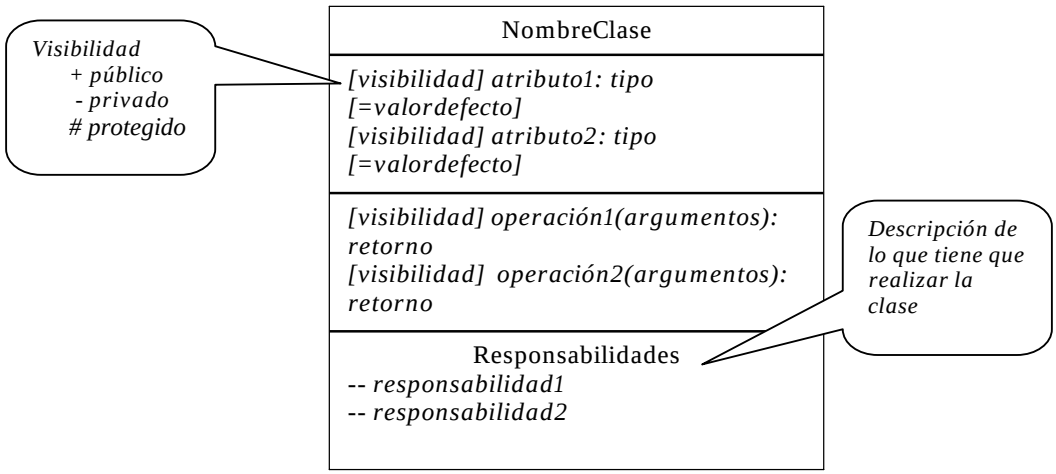


Figura 2.19 Representación de clase en UML

Un ***objeto*** se representa de la misma forma que una clase. En el compartimento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis: *nombre_del_objeto: nombre_de_la_clase*

Puede representarse un objeto sin un nombre específico, entonces sólo aparece el nombre de la clase.

Las ***relaciones*** entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. El nombre de la relación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la relación.

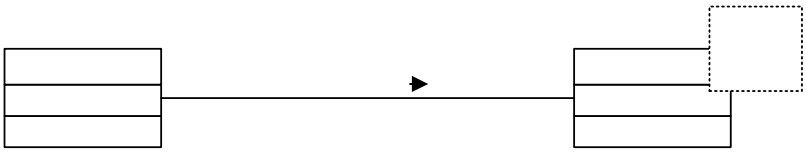


Figura 2.20 Diagrama de relaciones en UML

Existen diferentes tipos de relaciones, como las que se muestran a continuación:

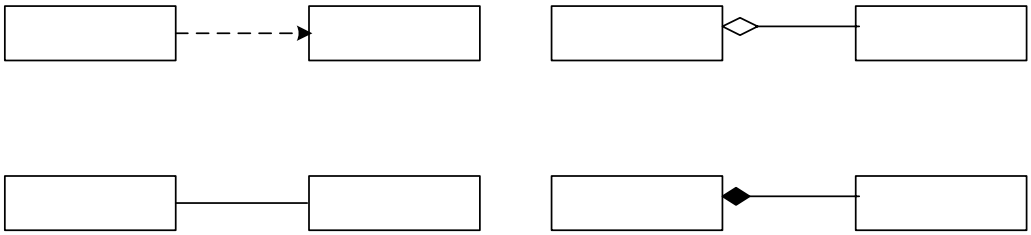


Figura 2.21 Tipos de relaciones en UML

La ***multiplicidad*** es una restricción que se pone a una relación, que limita el número de instancias de una clase que pueden tener esa relación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

- ✓ Con un número fijo: 1.
- ✓ Con un intervalo de valores: 2...5.
- ✓ Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, 2...* significa 2 o más.
- ✓ Con una combinación de elementos como los anteriores separados por comas: 1, 3...5, 7, 15...*.
- ✓ Con un asterisco: En este caso indica que puede tomar cualquier valor (cero o más).

Relación de depe

Clase 1

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de ***rol***. Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

Una ***interfaz*** específica ciertas operaciones de algunos elementos del modelo, tales como una clase, que son visibles fuera del mismo. No necesita especificar todas las operaciones que soporta el elemento, por lo que el mismo elemento podría incluir varias interfaces diferentes. Una interfaz se define en un diagrama de clases utilizando un rectángulo como el de icono de clase con las operaciones listadas en un compartimiento como si fuera una clase. El icono se marca como <<interface>>, y no tiene un compartimiento de atributo, porque una interfaz no puede tener atributos.

Relación de asoc

Clase 1

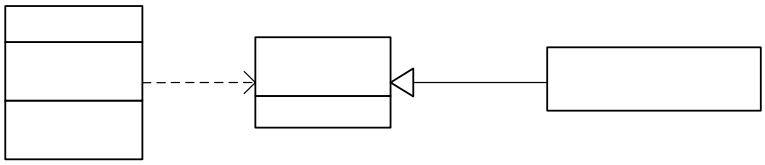


Figura 2.22 Diagrama de interfaces en UML

Para modelar el aspecto estático del sistema se utilizan los siguientes diagramas:

- ✓ Diagramas de casos de uso.
- ✓ Diagramas de interacción.
- ✓ Diagramas de colaboración.
- ✓ Diagramas de estados.

Para representar el aspecto dinámico de un sistema se utilizan los *diagramas de actividades* que sirven fundamentalmente para modelar el flujo de control entre actividades.

2.2.3. Diagramas estáticos

Diagrama de Casos de Uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

Un actor es una entidad externa al sistema que realiza algún tipo de interacción con el mismo. Se representa mediante una figura humana. Esta representación sirve tanto para actores que son personas como para otro tipo de actores (otros sistemas, sensores, etc.).

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

Entre dos casos de uso puede haber las siguientes relaciones:

- ✓ *Extiende*: Cuando un caso de uso especializa a otro extendiendo su funcionalidad.
- ✓ *Usa*: Cuando un caso de uso utiliza a otro.

Se representan como una línea que une a los dos casos de uso relacionados, con una flecha en forma de triángulo y con una etiqueta <<extiende>> o <<usa>> según sea el tipo de relación.

En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea.

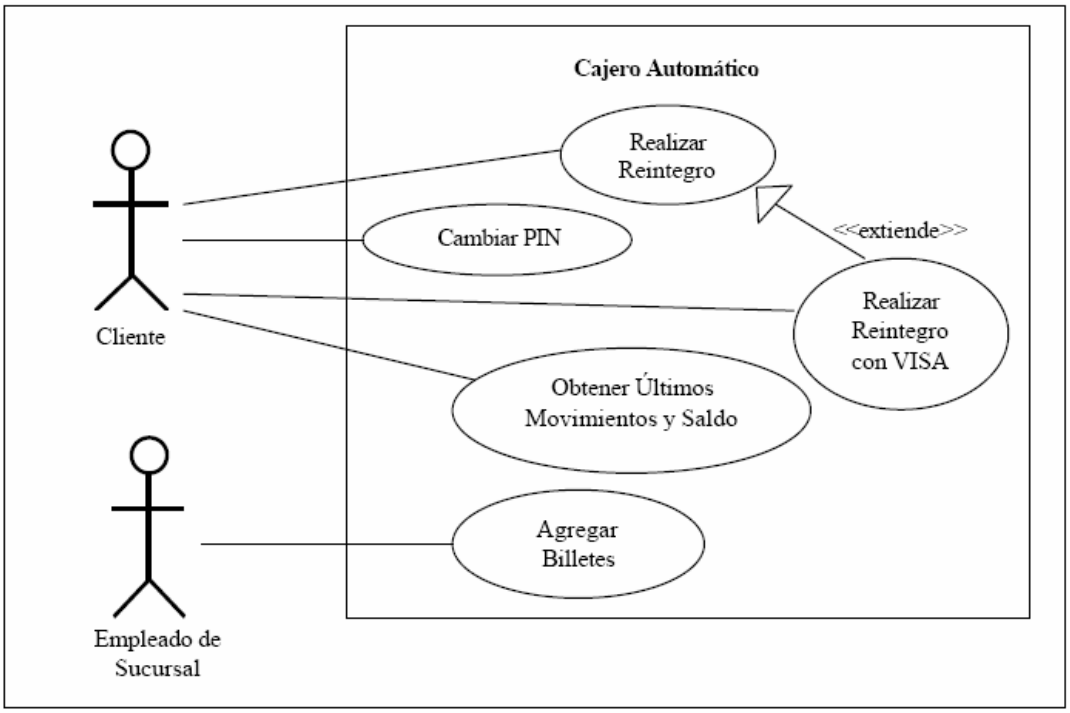


Figura 2.23 Diagrama de casos de uso

Diagramas de Interacción

En los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren.

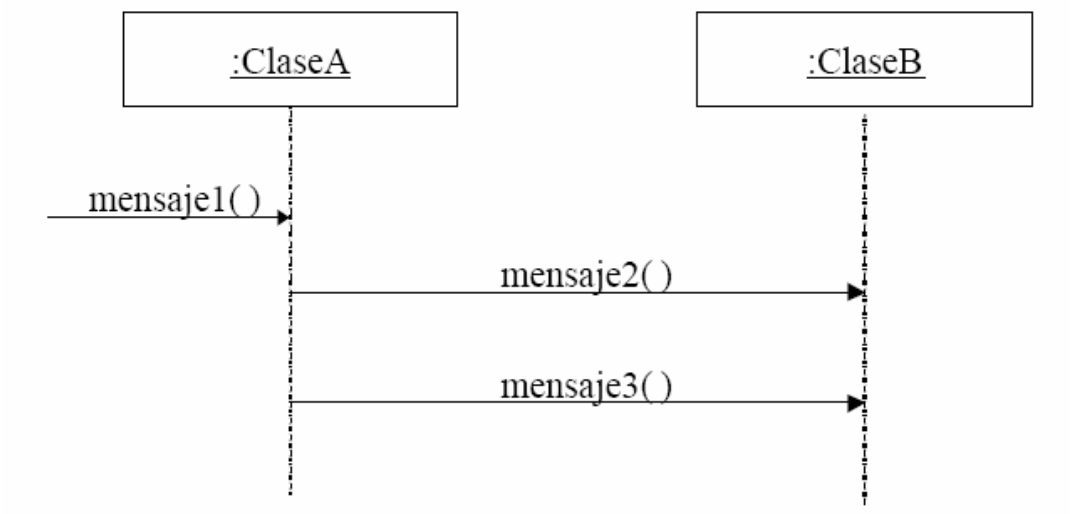


Figura 2.24 Diagrama de interacción

Diagrama de Colaboración

Un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

En cuanto a la representación, un Diagrama de Colaboración muestra a una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que "circulan", y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis. Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia.

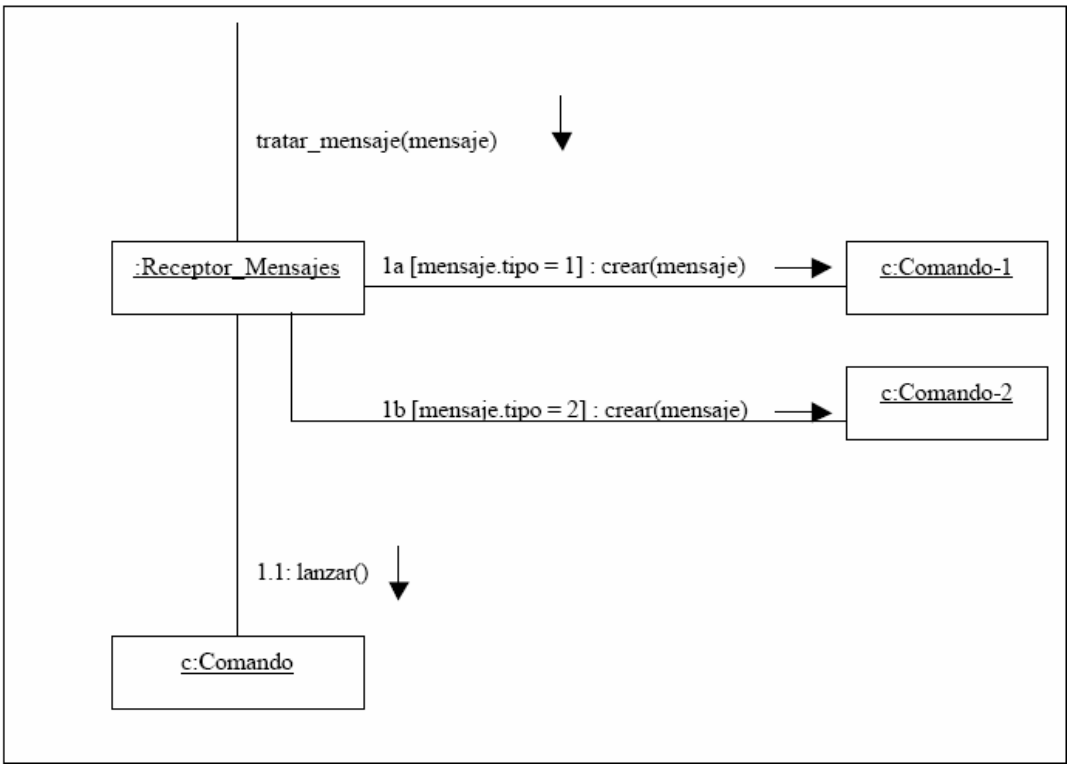


Figura 2.25 Diagrama de colaboración

Diagrama de Estados

Un Diagrama de Estados muestra la secuencia de estados por los que pasa un caso de uso o un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos. Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino. La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimiento aparece el nombre del estado. El segundo compartimiento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.

Una acción de entrada aparece en la forma entrada/acción_asociada donde acción_asociada es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta.

Una acción de salida aparece en la forma salida/acción_asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta.

Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma nombre_de_evento/acción_asociada.

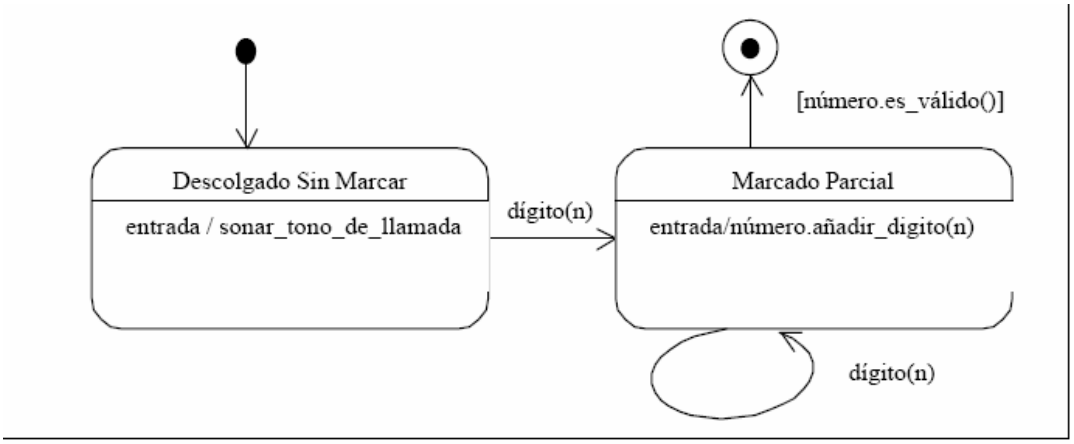


Figura 2.26 Diagrama de estados

Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (del caso de uso o del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudoestados, pues un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final(es).

2.2.4. Diagramas dinámicos

Diagrama de actividades

El diagrama de actividades sirve para representar el sistema desde otra perspectiva, y de este modo complementa a los anteriores diagramas vistos. Desde un punto de vista conceptual, el diagrama de actividades muestra cómo fluye el control de unas clases a otras con la finalidad de culminar con un flujo de control total que se corresponde con la consecución de un proceso más complejo. Por este motivo, en un diagrama de actividades aparecerán acciones y actividades correspondientes a distintas clases. Colaborando todas ellas para conseguir un mismo fin.

Básicamente un diagrama de actividades contiene:

- ✓ Estados de actividad.
- ✓ Estados de acción.
- ✓ Transiciones.

La representación de los ***estados de actividad*** y los ***estados de acción*** es un rectángulo con las puntas redondeadas, en cuyo interior se representa bien una actividad o bien una acción. La forma de expresar tanto una actividad como una acción, no queda impuesta por UML, se podría utilizar lenguaje natural, una especificación formal de expresiones, un metalenguaje, etc.

Las ***transiciones*** reflejan el paso de un estado a otro, bien sea de actividad o de acción. Esta transición se produce como resultado de la finalización del estado del que parte el arco dirigido que marca la transición. Como todo flujo de control debe empezar y terminar en algún momento, podemos indicar esto utilizando dos disparadores de inicio y fin.

Un flujo de control no tiene porqué ser siempre secuencial, puede presentar caminos alternativos. Para poder representar dichos caminos alternativos o ***bifurcación*** se utilizará como símbolo el rombo. Dicha bifurcación tendrá una transición de entrada y dos o más de salida. En cada transición de salida se colocará una expresión booleana que será evaluada una vez al llegar a la bifurcación, las guardas de la bifurcación han de ser excluyentes y contemplar todos los casos ya que de otro modo la ejecución del flujo de control quedaría interrumpida. Para poder cubrir todas las posibilidades se puede utilizar la palabra ELSE, para indicar una transición obligada a un determinado estado cuando el resto de guardas han fallado.

No sólo existe el flujo secuencial y la bifurcación, también hay algunos casos en los que se requieren tareas concurrentes. UML representa gráficamente el proceso de ***división***, que representa la concurrencia, y el momento de la ***unión*** de nuevo al flujo de control secuencial, por una línea horizontal ancha.

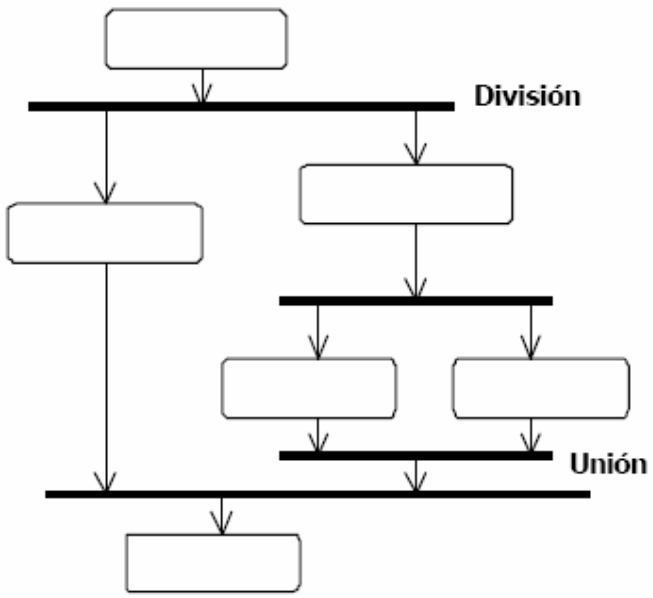


Figura 2.27 Proceso de unión y división en UML

Cuando se modelan flujos de trabajo de organizaciones, es especialmente útil dividir los estados de actividades en grupos, cada grupo tiene un nombre concreto y se denominan ***calles***.

Cada calle representa a la parte de la organización responsable de las actividades que aparecen en esa calle.

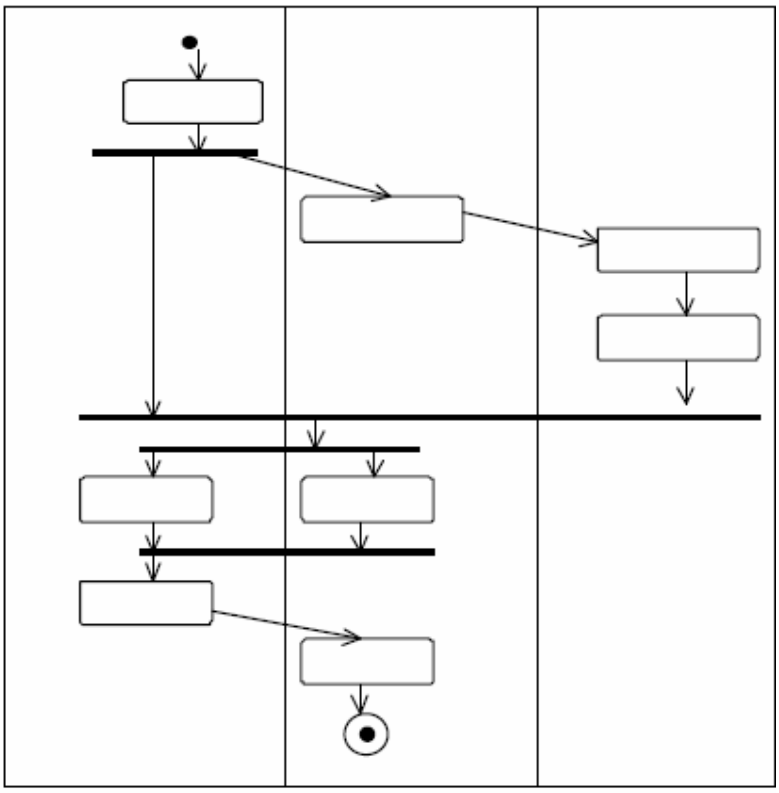


Figura 2.27 Calles en UML

2.2.5. Diagramas de aspectos físicos

Existen dos tipos de diagramas que sirven para modelar los aspectos físicos de un sistema orientado a objetos:

- ✓ Diagramas de componentes.
- ✓ Diagramas de despliegue.

Diagrama de componentes.

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos.

Normalmente los diagramas de componentes contienen:

- ✓ Componentes
- ✓ Interfaces
- ✓ Relaciones de dependencia, generalización, asociaciones y realización.

- ✓ Paquetes o subsistemas
- ✓ Instancias de algunas clases

Visto de otro modo un diagrama de componentes puede ser un tipo especial de diagrama de clases que se centra en los componentes físicos del sistema. Dentro de los usos más comunes tenemos:

Modelado de Código Fuente. Para modelar el código fuente de un sistema:

- ✓ Hay que identificar el conjunto de archivos de código fuente de interés y modelarlos como componentes estereotipados como archivos.
- ✓ Si el sistema es muy grande es necesario utilizar los paquetes para agrupar los archivos de código fuente.
- ✓ Es necesario identificar la versión del componente.

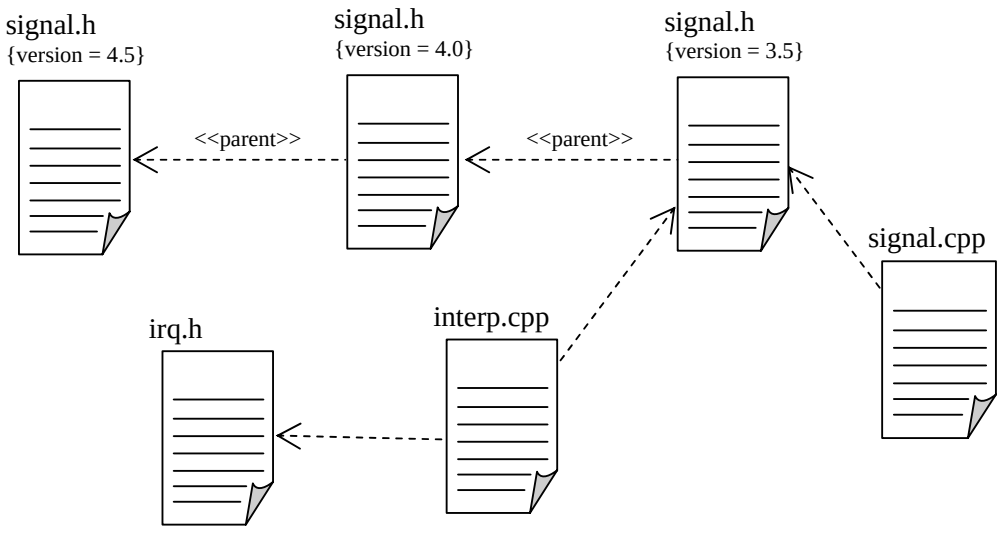


Figura 2.28 Modelado de código fuente en UML

Modelado de una versión ejecutable y bibliotecas. La utilización de los componentes para modelar versiones ejecutables se centra en la definición de todos los elementos que componen lo que se conoce como versión ejecutable, es decir la documentación, los archivos que se entregan etc. Para modelar una versión ejecutable es preciso:

- ✓ Identificar el conjunto de componentes que se pretende modelar.
- ✓ Identificar el estereotipo de cada componente del conjunto seleccionado.
- ✓ Para cada componente de este conjunto hay que considerar las relaciones con los vecinos. Esto implica definir las interfaces importadas por ciertos componentes y las exportadas por otros.

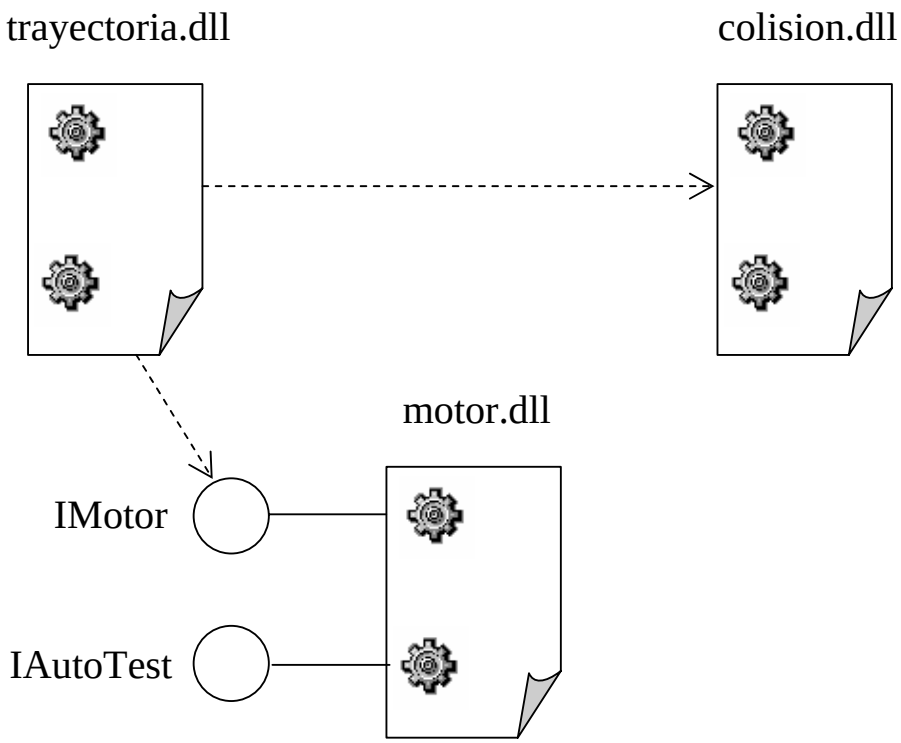


Figura 2.29 Modelado de una versión ejecutable en UML

Modelado de una base de datos física. Para modelar una base de datos física es necesario:

- ✓ Identificar las clases del modelo que representan el esquema lógico de la base de datos.
- ✓ Seleccionar una estrategia para hacer corresponder las clases con tablas. Así como la distribución física de la/s base/s de datos.
- ✓ Para poder visualizar, especificar, construir y documentar dicha correspondencia es necesario crear un diagrama de componentes que tenga componentes estereotipados como tablas.
- ✓ Donde sea posible es aconsejable utilizar herramientas que ayuden a transformar diseño lógico en físico.

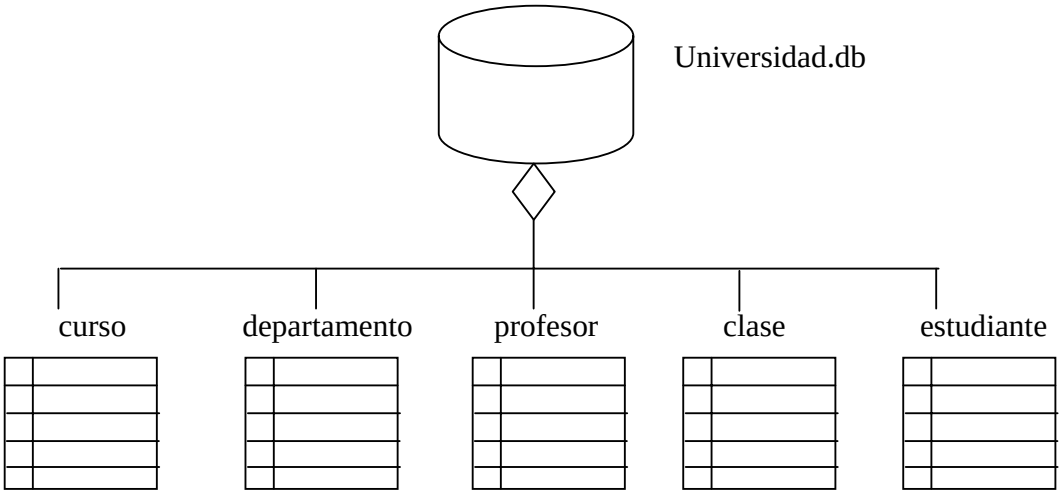


Figura 2.30 Modelado de una base de datos física en UML

Diagramas de Despliegue

Muestran la configuración de nodos que participan en la ejecución y de los componentes que residen en ellos. Se utilizan para:

- ✓ Modelar sistemas empotrados.
- ✓ Modelar sistemas cliente/servidor.

Modelado de un sistema empotrado. El desarrollo de un sistema empotrado es más que el desarrollo de un sistema software. Hay que manejar el mundo físico. Los diagramas de despliegue son útiles para facilitar la comunicación entre los ingenieros de hardware y los de software.

Para modelar un sistema empotrado es necesario:

- ✓ Identificar los dispositivos y nodos propios del sistema.
- ✓ Proporcionar señales visuales, sobre todo para los dispositivos poco usuales.
- ✓ Modelar las relaciones entre esos procesadores y dispositivos en un diagrama de despliegue.

Modelado de un sistema cliente servidor. La división entre cliente y servidor en un sistema es complicada ya que implica tomar algunas decisiones sobre dónde colocar físicamente sus componentes, qué cantidad de software debe residir en el cliente, etc. Para modelar un sistema cliente/servidor hay que hacer lo siguiente:

- ✓ Identificar los nodos que representan los procesadores cliente y servidor del sistema.
- ✓ Destacar los dispositivos relacionados con el comportamiento del sistema.

- ✓ Proporcionar señales visuales para esos procesadores y dispositivos a través de estereotipos.
- ✓ Modelar la tipología de esos nodos mediante un diagrama de despliegue.

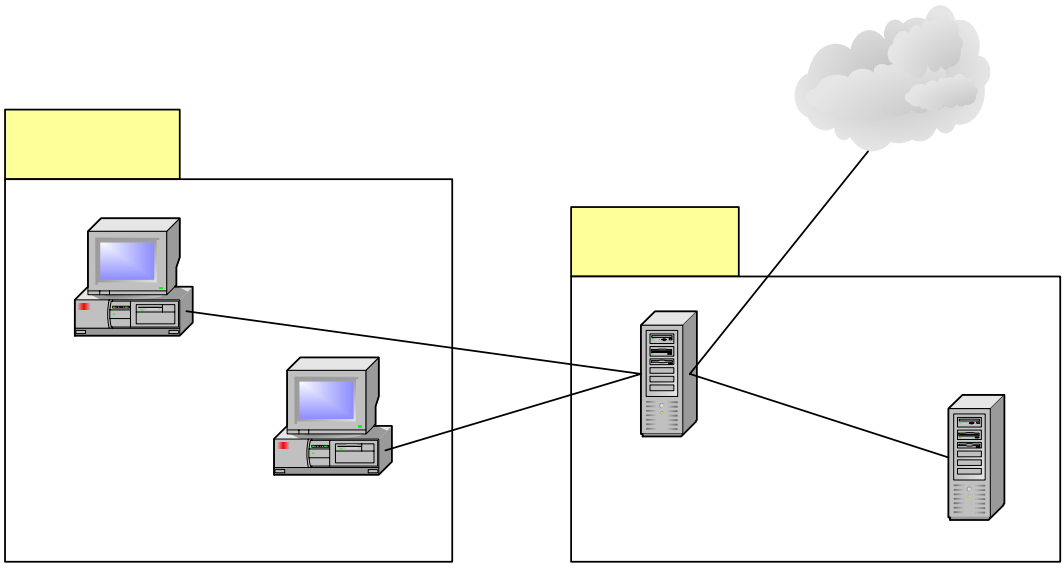


Figura 2.31 Modelo de un sistema cliente servidor

2.2.6. Implementación de OMT y UML en el SAPIUN

OMT y UML son metodologías de ingeniería de software. Cuando se habla de metodologías tradicionales, se hace hincapié en especificar y descomponer la funcionalidad del sistema, lo que parece ser un enfoque más directo para obtener un objetivo determinado; sin embargo el sistema en sí resulta frágil cuando se intentan cambiar los requisitos en caso de ser necesaria una reestructuración. En contraste, el enfoque orientado a objetos, en este caso OMT, se centra primordialmente en identificar objetos procedentes de la aplicación ajustándose después la funcionalidad, aunque a simple vista este enfoque puede parecer más indirecto, soporta mejor las evoluciones dado que el entorno de la aplicación siempre es más estable que sus requisitos funcionales.

clientes

UML define el lenguaje formal mediante el cuál se representa el modelado a objetos pero los principales conceptos de análisis y diseño fueron aportados por OMT, pues se concentra en la abstracción del problema. Por su parte UML los traslada a un lenguaje común entre usuarios y desarrolladores.

Por lo expuesto anteriormente se decidió utilizar OMT, ya que por los requisitos que presenta el SAPIUN, resulta ser más eficiente su uso. Puesto que no sólo nos permitirá realizar

la descomposición del sistema, sino que identificará todas y cada una de las fallas que se pudieran tener al momento de establecer los requerimientos, generando nuevas alternativas. Como parte de las nuevas tecnologías es importante utilizar un lenguaje común, mismo que será proporcionado por UML, ya que todos los diagramas que utiliza hacen entendible el desarrollo del sistema.

Al utilizar OMT y UML de manera conjunta podemos garantizar un sistema con un ciclo de vida de mayor al que resultaría sí se diseñara con las metodologías tradicionales. Entre los beneficios aportados por esta conjunción tenemos:

El uso de fases claras, cada una de las cuales tiene un producto final que se traduce en documentación.

- ✓ Encuentra un conjunto claro de requisitos, cuidadosamente definidos tan pronto como sea posible.
- ✓ Considera las pruebas como algo esencial para la construcción del sistema.
- ✓ Permite reutilizar componentes mediante una arquitectura modular.

Diagramas de Modelado (OMT Y UML)

Capítulo III

DIAGRAMAS DE MODELADO (OMT Y UML) EN EL SAPIUN

3.1. Las herramientas CASE en el SAPIUN

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE (*Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora*), se pueden definir como: "el conjunto de herramientas y metodologías que soportan un enfoque de ingeniería para las distintas fases del desarrollo de software", fueron desarrolladas para automatizar el proceso de desarrollo de un sistema y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de vida del software. La parte más importante de cualquier herramienta CASE es su metodología de desarrollo. Si los desarrolladores no están siguiendo estrictamente una metodología, las herramientas automatizadas no ayudarán mucho. Una vez que los desarrolladores adoptan una metodología, el uso de la herramienta CASE con dicha metodología ayuda enormemente.

La tecnología CASE supone la "informatización de la informática", es decir "la automatización del desarrollo del software", contribuyendo así a elevar la productividad y la calidad en el desarrollo de sistemas de información, de forma análoga a lo que suponen las técnicas CAD (*Computer Aided Design, Diseño Asistido por Computadora*)/CAM (*Computer Aided Manufacturing, Fabricación Asistida por Computadora*) en el área de fabricación.

Este nuevo enfoque persigue los siguientes objetivos:

- ✓ Permitir la aplicación práctica de metodologías estructuradas, lo que resulta muy difícil sin emplear herramientas.
- ✓ Mejorar la calidad del software.
- ✓ Facilitar la realización de prototipos, y el desarrollo conjunto de aplicaciones.
- ✓ Simplificar el mantenimiento de los programas.
- ✓ Estandarizar la documentación.
- ✓ Aumentar la portabilidad de las aplicaciones.
- ✓ Facilitar la reutilización de componentes software.
- ✓ Permitir un desarrollo visual de las aplicaciones, mediante la utilización de gráficos.

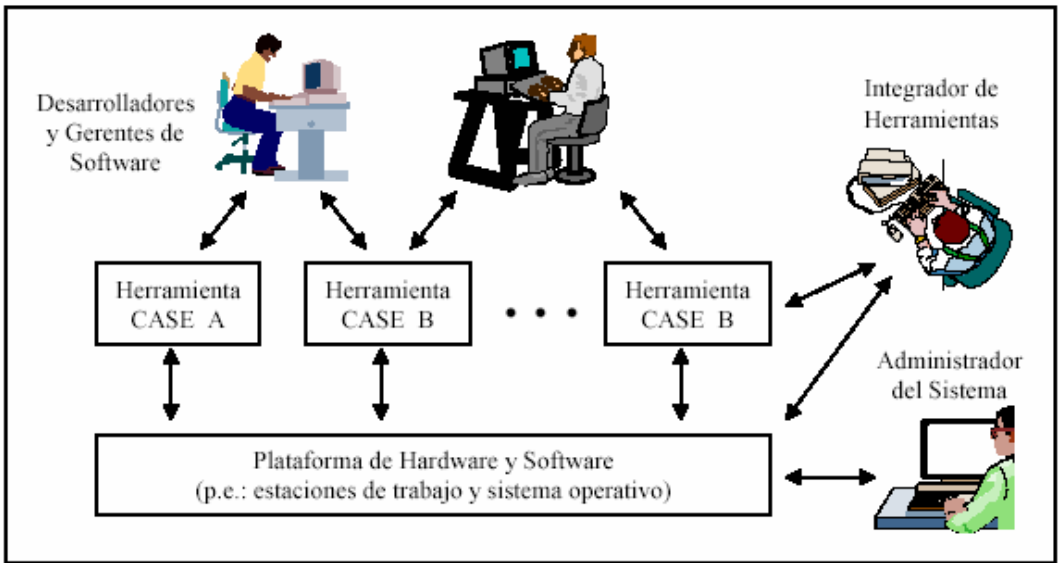


Figura 3.1 Ambiente CASE del SAPIUN

* Un ambiente CASE es "un conjunto de herramientas CASE y otros elementos (plataforma de hardware/software), junto con un enfoque de integración, que soporta la mayoría o todas las interacciones entre las componentes del ambiente y entre sus usuarios"

3.1.1. Evolución histórica de las herramientas CASE

Esta tecnología surge a mediados de los años setenta, cuando aparecen las primeras herramientas de documentación y diagramación automática, mejorando la calidad de los diseños de software. Los diccionarios de datos, un documento muy usado que mantiene los detalles de cada tipo de dato y los procesos dentro de un sistema, son el resultado directo de la llegada del diseño de flujo de datos y análisis estructural, hecho posible a través de las mejoras en las Herramientas CASE. Como un paso final, la verificación de errores y generadores de casos de pruebas fueron incluidos para validar el diseño del software. No fue sino hasta 1985 en que las herramientas CASE se volvieron realmente importantes en el proceso de desarrollo de software; sin embargo, ésta primera generación de herramientas fracasa debido principalmente a tres factores:

- ✓ limitaciones de los productos.
- ✓ Falsas expectativas sobre sus posibilidades.
- ✓ Incorrecta implementación.

A mediados de los noventa esta tecnología entró en su fase de madurez en la que surgió una "segunda generación" de herramientas, que superan gran parte de las limitaciones existentes en la primera generación.

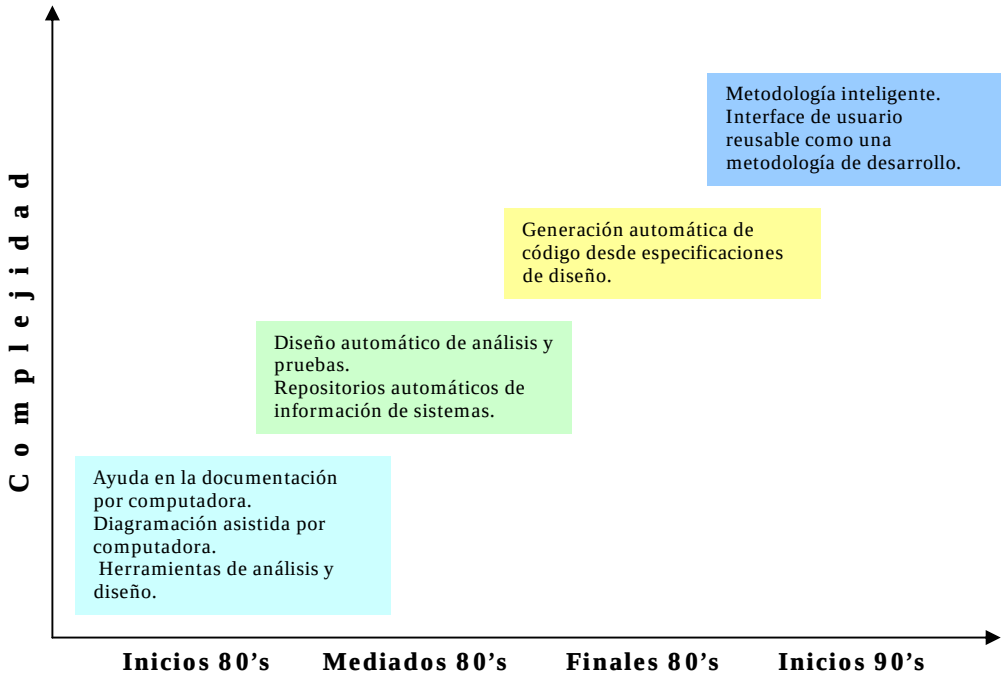


Figura 3.2 Evolución de las herramientas CASE

En definitiva, la tecnología CASE ha experimentado la clásica evolución que sufren aquellos paradigmas (como técnicas estructuradas, inteligencia artificial, lenguajes de cuarta generación y, en estos momentos, la orientación al objeto), que se ofrecen como la panacea universal capaz de resolver todos los problemas del desarrollo de sistemas de información.

3.1.2. Elementos de una herramienta CASE

De manera específica una herramienta CASE se compone de los siguientes elementos:

- ✓ *Repositorio*, base de datos central de una herramienta CASE. El repositorio amplía el concepto de diccionario de datos para incluir toda la información que se va generando a lo largo del ciclo de vida del sistema. La mayoría de herramientas CASE poseen un repositorio propio o bien trabajan sobre un repositorio suministrado por otro fabricante o vendedor. Apoyándose en la existencia del repositorio se efectúan comprobaciones de integridad y consistencia:
  - o Que no existan datos no definidos.
  - o Que no existan datos autodefinidos (datos que se emplean en una definición pero que no han sido definidos previamente).
  - o Que todos los alias (referencias a un mismo dato empleando nombres distintos) sean correctos y estén actualizados.

Las características más importantes de un repositorio son:

- ✓ *Tipo de información*. Que contiene alguna metodología concreta, datos, gráficos, procesos, informes, modelos o reglas.
- ✓ *Tipo de controles*. Si incorpora algún módulo de gestión de cambios, de mantenimiento de versiones, de acceso por clave, de redundancia de la información.
- ✓ *Metamodelo* (no siempre visible), que define las técnicas y metodologías soportadas por la herramienta, y que es conveniente que pueda ser extensible por parte del usuario.
- ✓ *Generador de informes*, que permite obtener toda la documentación que describe el sistema de información desarrollado; documentación que está asociada a las técnicas y metodologías.
- ✓ *Herramienta de carga/descarga de datos*, que permite cargar el repositorio de la herramienta CASE con datos provenientes de otros sistemas, o generar a partir de la propia herramienta esquemas de bases de datos, programas, etc.
- ✓ *Interfaz de usuario*, que constará de editores de texto y herramientas de diseño gráficos, que permitan mediante la utilización de un sistema de ventanas, iconos

y menús, con la ayuda del ratón, definir los diagramas, matrices, etc. que incluyen las distintas metodologías. Lo que se conoce usualmente por las siglas inglesas WIMP (Windows, Icons, Mouse y Pull-down menus).

- ✓ *Comprobación de errores*, facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.

3.1.3. Categorías de herramientas CASE

Las herramientas CASE, en función de las fases del ciclo de vida abarcadas, se pueden agrupar de la forma siguiente:

- ✓ Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.
- ✓ Herramienta(s) que comprende(n) alguna(s) fase(s) del ciclo de vida de desarrollo de software:
  - o Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o front-end, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.
  - o Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.
  - o Juegos de herramientas o toolkits, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

Tipo de Case	Ventajas	Desventajas
Upper Case	Se utiliza en plataforma PC, es aplicable a diferentes entornos. Menor costo.	Permite mejorar la calidad de los sistemas, pero no mejora la productividad. No permite la integración del ciclo de vida.

Lower Case	Permite lograr importantes mejoras de productividad a corto plazo. Permite un eficiente soporte al mantenimiento de sistemas.	No garantiza la consistencia de los resultados a nivel corporativo. No garantiza la eficiencia del Análisis y Diseño. No permite la integración del ciclo de vida.
I-Case	Integra el ciclo de vida. Permite lograr importantes mejoras de productividad a mediano plazo. Permite un eficiente soporte al mantenimiento de sistemas. Mantiene la consistencia de los sistemas a nivel corporativo.	No es tan eficiente para soluciones simples, sino para soluciones complejas. Depende del hardware y del software. Es costoso.

Tabla 1 Tipos de herramientas CASE

Otra posible clasificación, utilizando la funcionalidad como criterio principal, es la siguiente:

- ✓ *Herramientas de planificación de sistemas de gestión*. Su objetivo principal es ayudar a comprender mejor cómo se mueve la información. Proporcionando un "metamodelo" del cual se pueden obtener sistemas de información específicos.
- ✓ *Herramientas de análisis y diseño*. Proporcionan un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación ya que permite al desarrollador crear un modelo del sistema que se va a construir y también la evaluación de la validez y consistencia de este modelo.
- ✓ *Herramientas de programación*. Se engloban aquí los compiladores, los editores y los depuradores de los lenguajes de programación convencionales.
- ✓ *Herramientas de integración y prueba*: Sirven de ayuda a la adquisición, medición, simulación y prueba de los equipos lógicos desarrollados.
- ✓ *Herramientas de gestión de prototipos*. Los prototipos son utilizados ampliamente en el desarrollo de aplicaciones, para la evaluación de especificaciones de un sistema de información, o para un mejor entendimiento de cómo los requisitos de un sistema de información se ajustan a los objetivos perseguidos.

- ✓ *Herramientas de mantenimiento*: Sirven en los procesos de ingeniería inversa, reingeniería, reestructuración y análisis de código.
- ✓ *Herramientas de gestión de proyectos*. La mayoría de las herramientas CASE de gestión de proyectos, se centran en un elemento específico de la gestión del proyecto, en lugar de proporcionar un soporte global para la actividad de gestión. Utilizando un conjunto seleccionado de las mismas se puede: realizar estimaciones de esfuerzo, coste y duración, hacer un seguimiento continuo del proyecto, estimar la productividad y la calidad, etc.
- ✓ *Herramientas de soporte*. Se engloban en esta categoría las herramientas que recogen las actividades aplicables en todo el proceso de desarrollo como son: documentación, control de calidad y bases de datos.

3.2. Planeación, análisis y diseño del SAPIUN

En el capítulo uno, se definieron los requerimientos del sistema, mismos que trasladaremos a la metodología OMT como se justifico en el capitulo dos; para tal efecto haremos uso de la herramienta CASE *Rational Rose* (desarrollada por los creadores de UML y que cubre todo el ciclo de vida de un sistema de información), pues gracias a su fácil utilización nos permitirá llevar a cabo una administración completa del SAPIUN en su fase de análisis y diseño, obteniendo los diagramas necesarios para su concepción. Pues facilita el desarrollo de un proceso cooperativo en la que todos los agentes tienen sus vistas de información (Vistas de Casos de Uso, Vista Lógica, Vista de Componentes), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad del sistema en construcción.

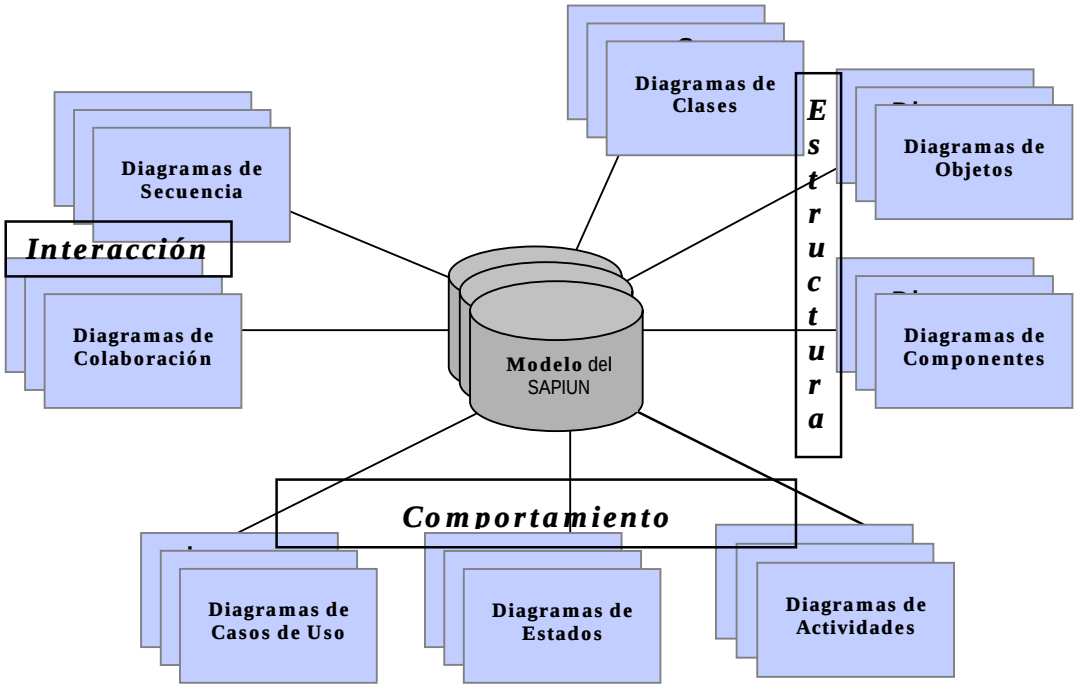


Figura 3.3 Administración del SAPIUN mediante Rational Rose

Como parte de la fase de análisis de OMT es necesario definir el diagrama de despliegue, mismo que se construirá tomando como base los recursos disponibles de software y hardware dentro de UNICA, es decir, actualmente se cuenta con un servidor de base de datos que administra las bases de datos utilizadas, cuyo manejador es Postgress, además de tener un servidor Web con Apache que se encarga de administrar los recursos que ofrece en Internet, los equipos que utiliza el personal administrativo tienen como base el sistema operativo Windows en sus diversas versiones por tanto se tiene la siguiente arquitectura del trabajo para el SAPIUN:

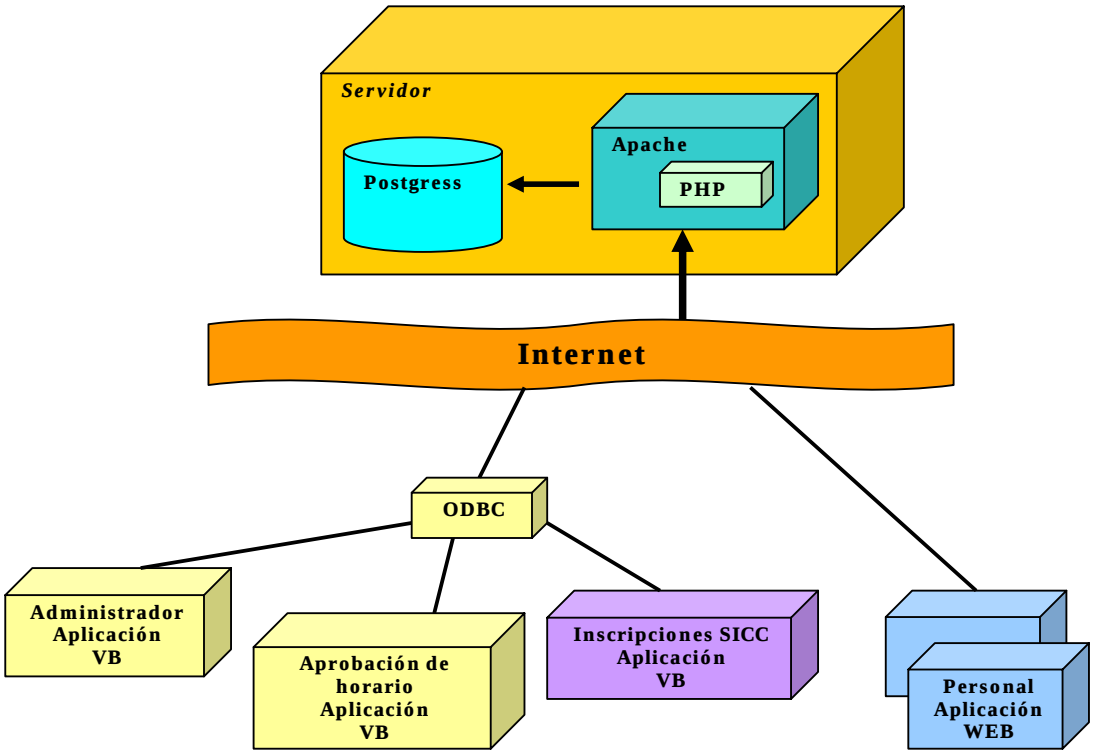


Figura 3.4 Arquitectura del SAPIUN

La tecnología utilizada para acceder a los servicios de la base de datos es ODBC (*Open Data Base Connectivity, Conectividad Abierta de Bases de Datos*). Esta tecnología proporciona una interfaz común para tener acceso a bases de datos de distintos tipos, por estar basado en SQL (*Structured Query Language*). Esta interfaz proporciona máxima interoperatividad: una aplicación simple puede tener acceso a un DBMS (*Data Base Manager System, Sistema Manejador de Bases de Datos*) a través de un conjunto de código común. Esto permite al programador, crear y distribuir aplicaciones cliente--servidor sin limitarse a un DBMS específico. El controlador utilizado para vincular el SAPIUN con el DBMS del usuario es PostgreSQL permitiendo la realización de consultas y actualizaciones.

EL ODBC será administrado por el API OLE DB (*Object Linking and Embedding for Data Base*) en un modelo objeto simple llamado ADO (*ActiveX Data Objects*) que reduce el desarrollo, mantenimiento y costo de la aplicación. Ofrece las siguientes ventajas a los programadores: puede accesar datos desde cualquier recurso OLE DB y puede mostrar propiedades específicas de los datos.

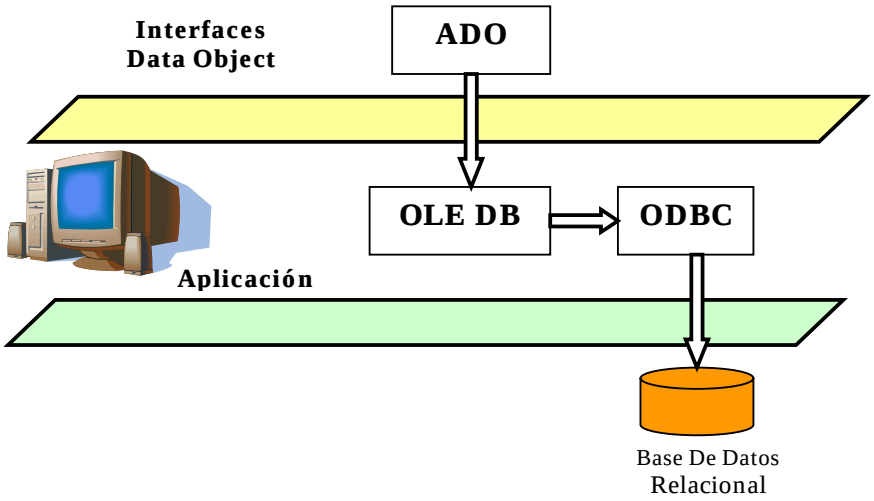


Figura 3.5 Relación entre ADO y ODBC

ADO constituye la interfaz de alto nivel que emplea un lenguaje neutral proporciona un acceso a los datos coherente y de alto rendimiento, tanto si se desea crear un cliente de base de datos de usuario como un objeto de empresa de nivel intermedio mediante una aplicación, herramienta, lenguaje o incluso un explorador de internet. ADO es la única interfaz necesaria para el desarrollo de soluciones controladas por datos para Web y cliente-servidor a diferentes niveles.

ADO está compuesto de siete objetos los cuales se presentan en la siguiente figura:

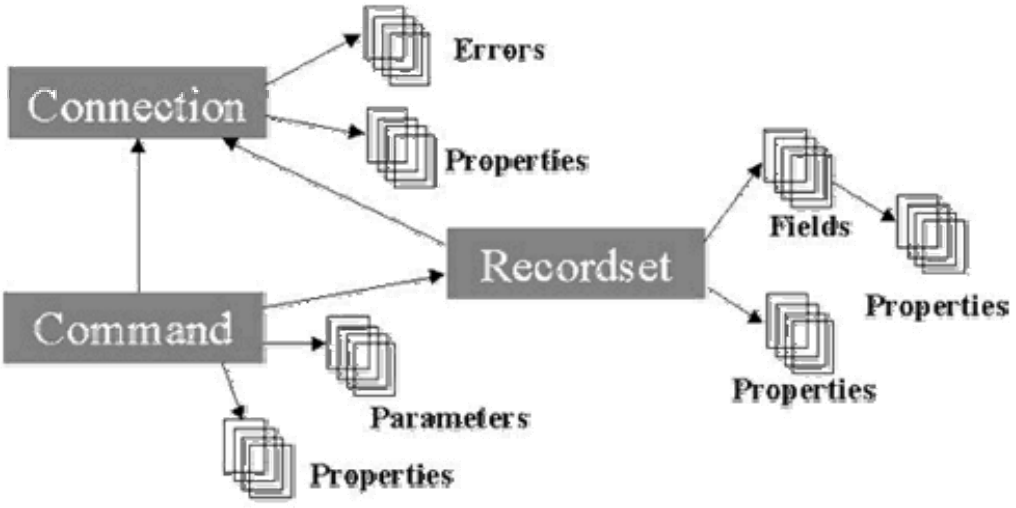


Figura 3.6 Objetos de ADO

El objeto Connection encapsula a los objetos OLE DB Data Source y Session. Define propiedades de la conexión, asigna las transacciones, provee una localidad central para recuperar errores, y provee un punto para ejecutar los esquemas de consulta.

El objeto Command encapsula el objeto OLE DB Command. Especifica las sentencias data-definition y data-manipulation para ser ejecutadas. Además permite especificar parámetros y personalizar el funcionamiento de la sentencia a ser ejecutada.

El objeto Recordset encapsula el objeto OLE DB Rowset. Representa la interface actual a los datos, ya sea un resultado de una consulta (query) o que haya sido generado de otra forma. Dicho objeto provee el control sobre cualquier mecanismo usado, el tipo de cursor, el número de filas a accesar en un tiempo, etc. El objeto Recordset muestra la colección de objetos Field que contienen metadatos acerca de las columnas en un conjunto de registros, obtenidos de la base de datos, tales como nombre, tipo, longitud, así como el valor actual.

Los objetos Command y Connection pueden ser creados una vez y compartirse a través de múltiples Recordsets. Esta ventaja es aprovechada ya que no solo se pueden compartir conexiones a través de múltiples consultas, sino también a través de múltiples páginas Web, para el mismo cliente o para muchos clientes.

Cada uno de los objetos descritos anteriormente contienen una colección de objetos Property. El objeto Property permite a ADO mostrar dinámicamente las capacidades de un objeto específico.

3.3. Modelo del SAPIUN en OMT y UML

3.3.1. Diagrama de Casos de Uso

Los casos de uso documentan el comportamiento del sistema desde el punto de vista del usuario. El diagrama muestra, no un único caso de uso, sino todos los casos de uso del sistema y puede verse como un resumen conciso de la información contenida en todas las descripciones. Un caso de uso individual, que aparece como un óvalo con un nombre, representa un tipo de tarea que tiene que soportar el sistema en el desarrollo. Un actor, que normalmente aparece con el símbolo de un muñeco, representa un tipo de usuario del sistema. Hay una línea que conecta un actor con un caso de uso, si el actor puede interactuar con el sistema para realizar parte de la tarea. Opcionalmente, puede haber una caja en un diagrama de casos de uso, alrededor de los casos de uso etiquetada con el nombre del sistema, la cual representa el límite del sistema. En el caso del SAPIUN se tiene el siguiente diagrama de casos de uso:

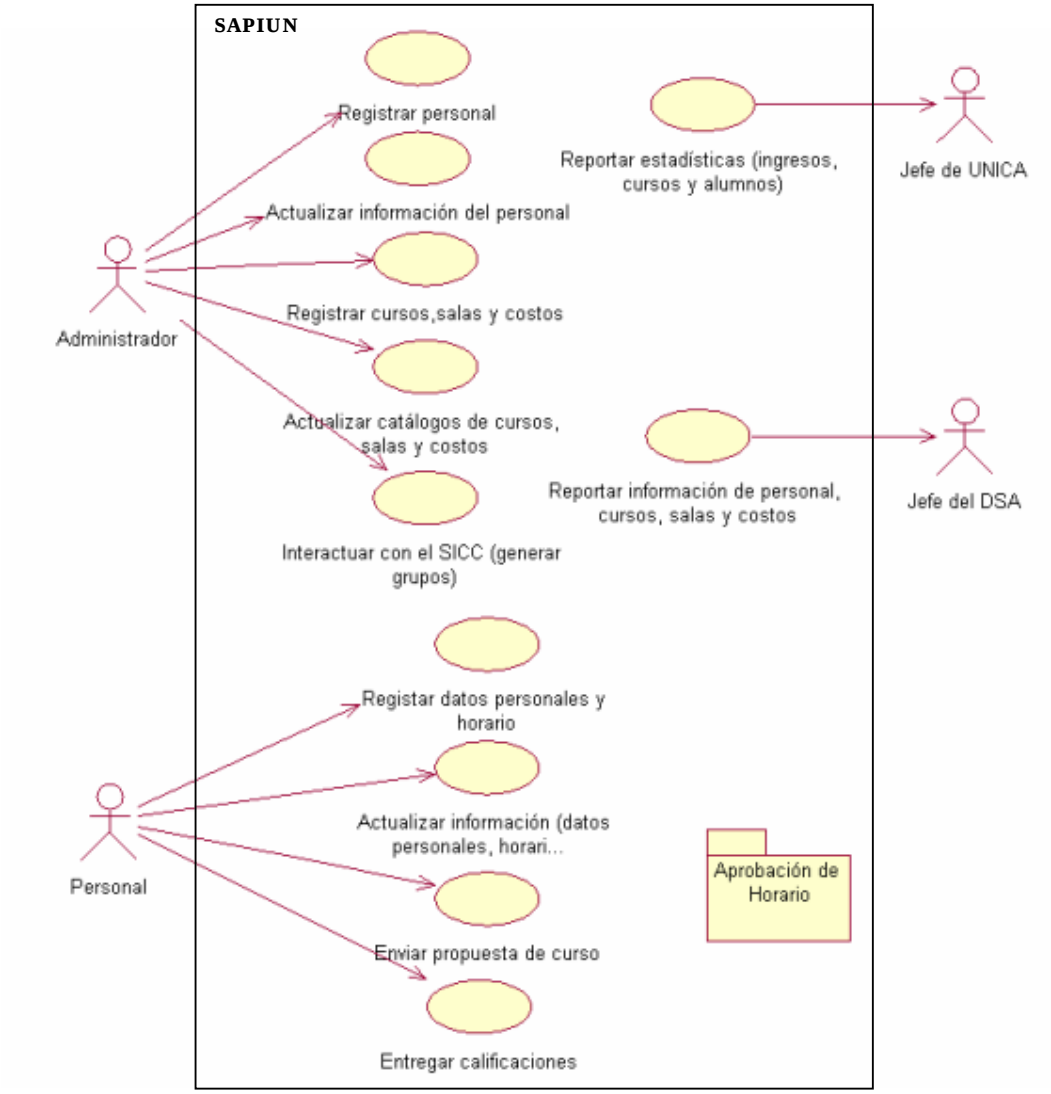


Figura 3.7 Diagrama de Casos de uso del SAPIUN

Como se observa en el diagrama anterior, se tienen cuatro actores. El primero de ellos es el administrador del SAPIUN, quien se encargará de realizar los procesos de registro de los datos personales, departamento, puesto y horario de trabajo del personal; así mismo, realizará las modificaciones de cualquiera de los datos antes mencionados. Registrará la información correspondiente a los catálogos de cursos, salas y costos que constituyen la base para la generación de grupos dentro del SICC; mediante el SICC asignará uno o más grupos al instructor que forma parte del personal.

El segundo actor es el personal que labora en UNICA quien contará con un portal en la Web en el que realizará el registro/actualización de sus datos personales y horario de trabajo. Mediante este mismo portal hará el envío de calificaciones de los grupos a los que haya impartido curso; así como la propuesta para la impartición de un nuevo curso.

El tercer actor es el Jefe de UNICA, quien recibirá el reporte de las estadísticas referentes al total de ingresos obtenidos, tipos de alumnos inscritos, cursos impartidos/cancelados por periodo y modalidad de curso.

El cuarto actor se refiere al Jefe de Departamento DSA el cual recibirá los reportes referentes al personal adscrito, cursos que imparte con su respectivo temario, salas disponibles con una descripción del hardware y software con el que cuentan y costos para cada tipo de modalidad de curso.

Una vez descrito el modelo general, se describirá el diagrama del subsistema de aprobación de horario que proporciona a los jefes de cada uno de los departamentos la posibilidad de consultar la información de todo el personal con la opción de aprobar/desaprobar el horario de trabajo del personal a su cargo.

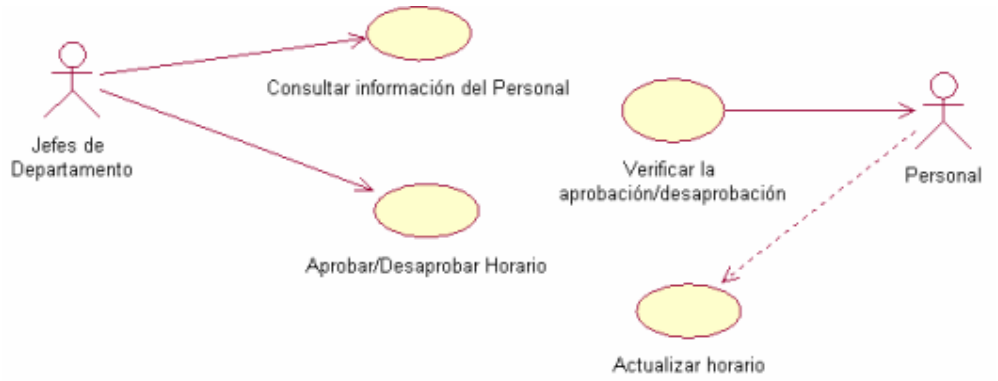


Figura 3.8 Diagrama de Casos de uso del subsistema de Aprobación de Horario

Una vez que el jefe de departamento haya aprobado/desaprobado un horario de trabajo el personal deberá consultar el portal para verificar si su horario fue aprobado con lo que se daría fin al proceso, en caso contrario tendrá que hacer las modificaciones pertinentes, para iniciar nuevamente el proceso de aprobación de su horario.

3.3.2. Diagrama de Clases

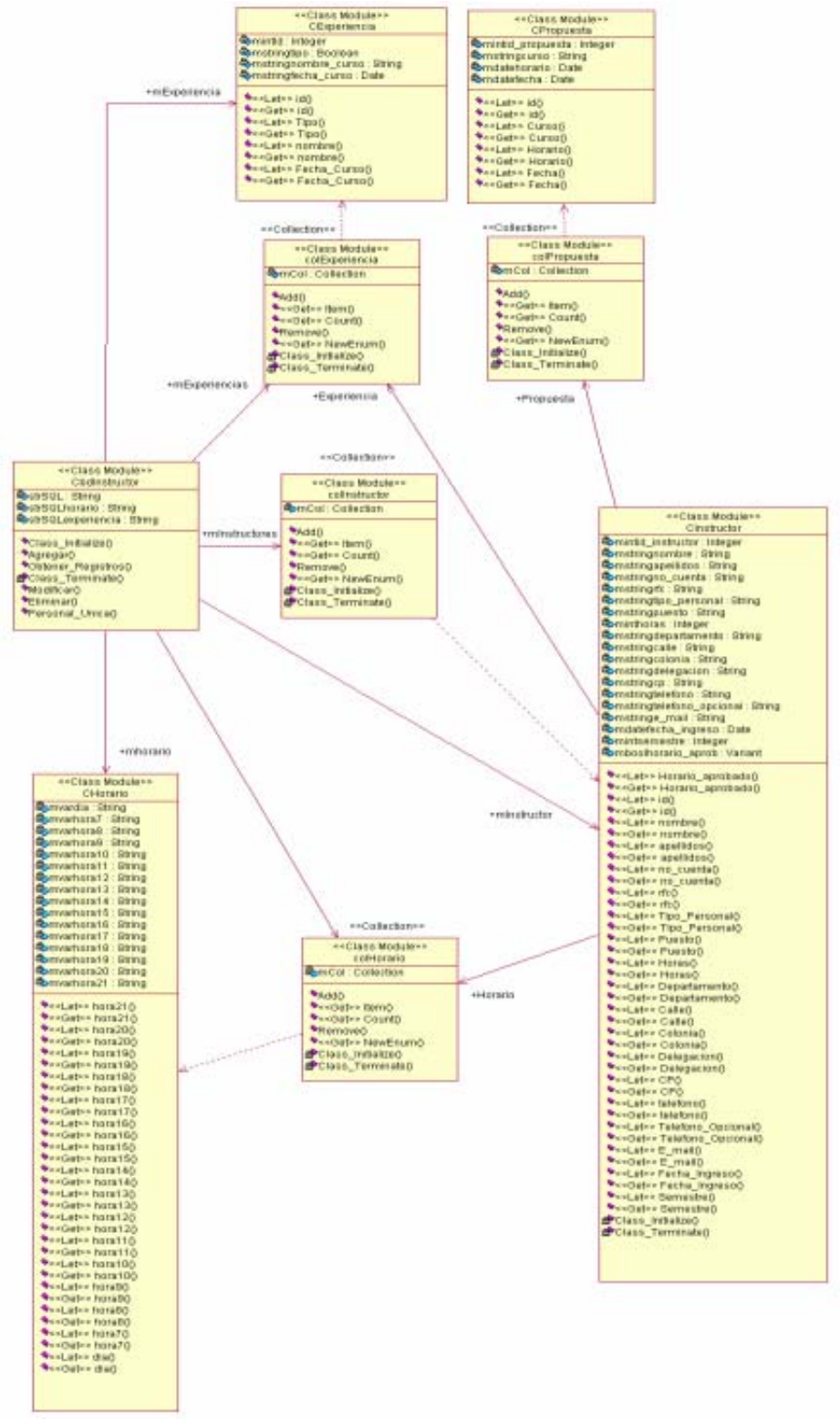


Figura 3.9 Diagrama de Clases del módulo de Instructores utilizando Visual Basic

El diagrama de clases forma parte de la vista estática del sistema; como ya se ha comentado en el capitulo anterior será donde definan las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización. Es decir, donde se plasmará el diseño orientado a objetos.

El diagrama de clases de la figura 3.9 nos muestra la parte central del sistema, ya que del modulo de instructores parte la creación del SAPIUN e interacción con el SICC. Se crearon cuatro clases bases entre que las que destacan instructor y horario, estas contienen toda la información del personal; las clases de experiencia y propuesta servirán como contadores, siendo la primera encargada de la información referente a la capacitación e impartición de cursos, y la segunda manejará la información relacionada con las propuestas de los instructores para la impartición de cursos. Finalmente tenemos la clase de agregación CbdInstructor que se encarga del control del modulo, así como, de realizar las transacciones con la base de datos.

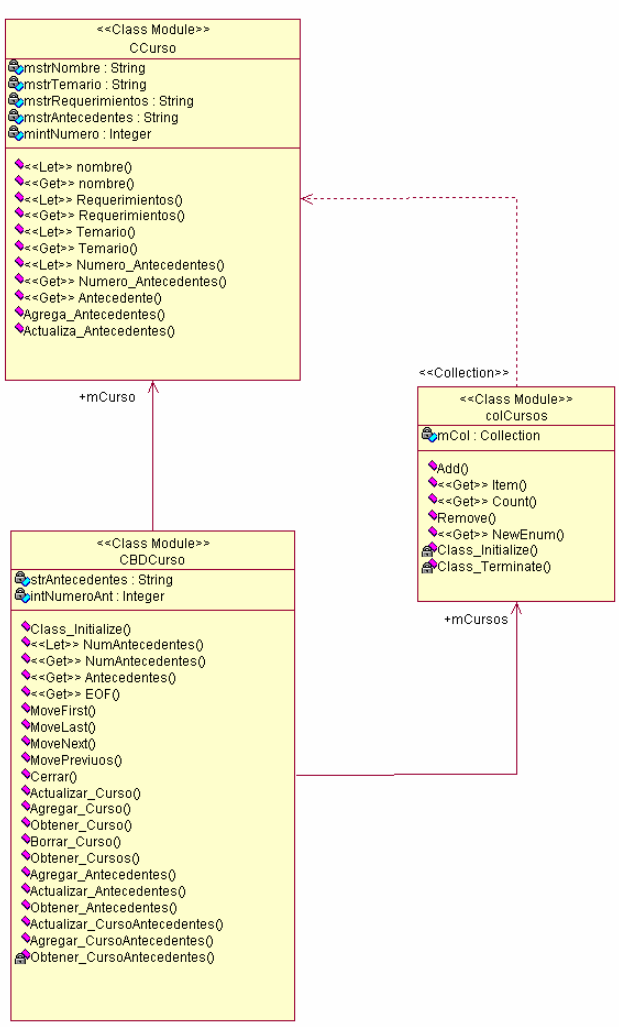


Figura 3.10 Diagrama de Clases del módulo de Cursos utilizando Visual Basic

El módulo de cursos está conformado por una clase base CCurso que contiene la información de los cursos que imparte UNICA y que sirven para la creación de grupos del SICC; y una clase de agregación CBDCurso que controlará el funcionamiento del módulo y la interacción con la base de datos.

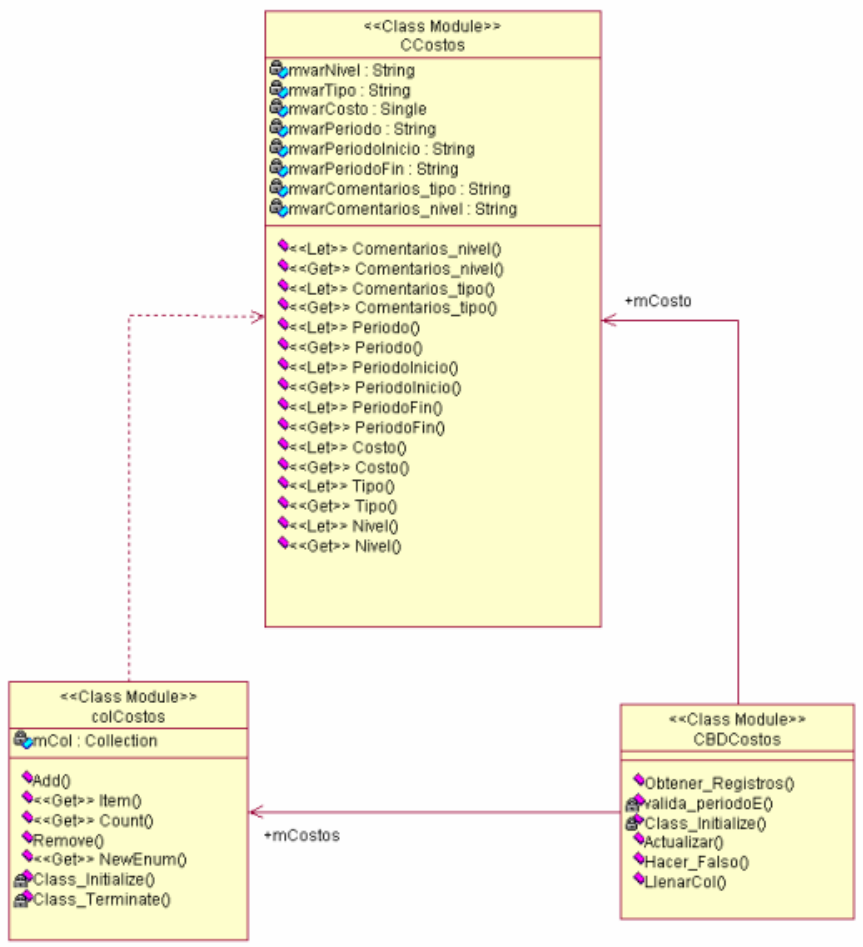


Figura 3.11 Diagrama de Clases del módulo de Costos utilizando Visual Basic

Como parte de la interacción del SAPIUN con el SICC, fue necesaria la implementación de una clase base que almacene la información referente a los costos que se asocian a los grupos creados por el SICC. Además como se ha mencionado cuenta con su clase de agregación que hará las transacciones pertinentes con la base de datos.

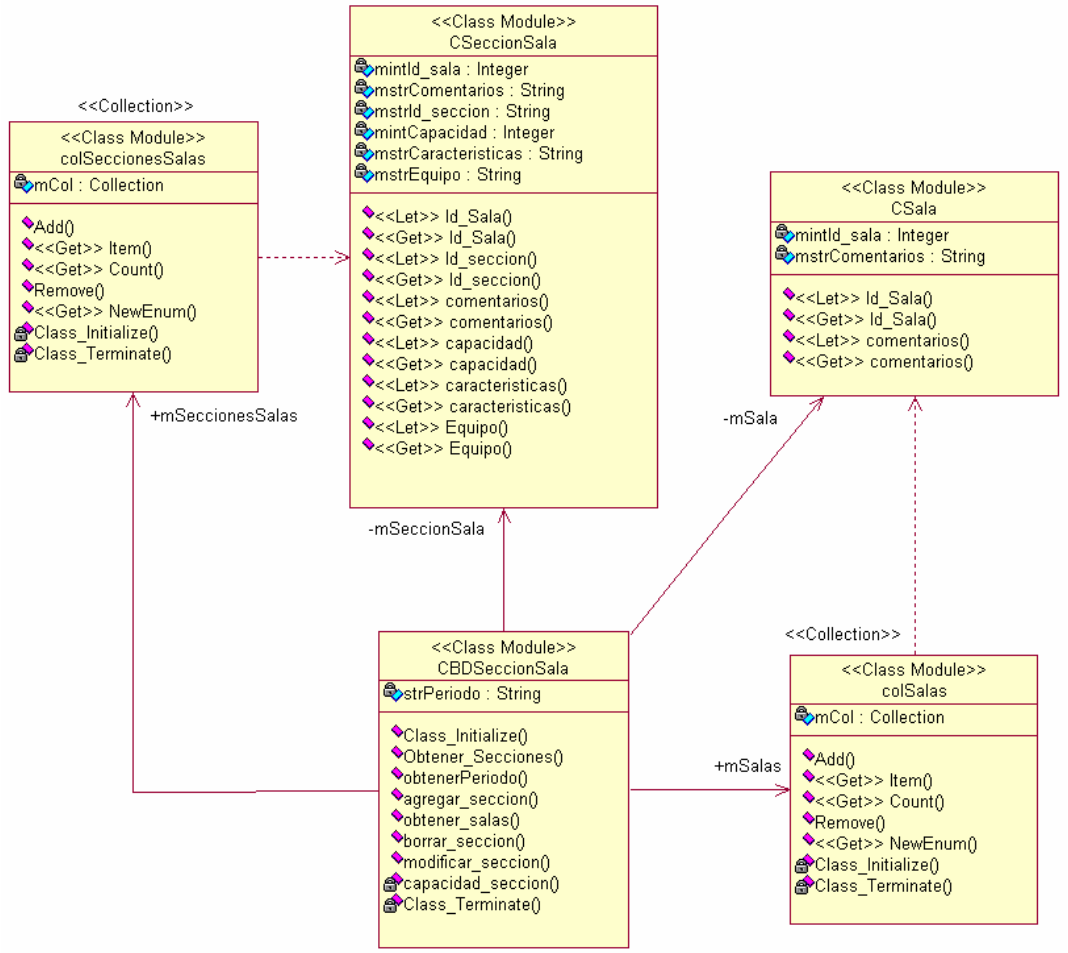


Figura 3.12 Diagrama de Clases del módulo de Salas utilizando Visual Basic

El módulo de salas contiene la información de las salas y sus secciones con la descripción del hardware y software que disponen cada una; para tal efecto se crearon dos clases base; una clase CSala que almacenará la información general de las salas con las que cuenta UNICA, y una clase CSeccionSala que contiene el detalle de las secciones de cada una de las salas. Por ultimo una clase de agregación CBDSeccionSala que realizará el control del módulo y de las transacciones con la base de datos. Adicionalmente este módulo proporciona los datos para asignar un lugar a los grupos creados por el SICC.

3.3.3. Diagrama de Estados

Para implementar, mantener o probar la clase es necesario comprender las relaciones de dependencia entre el estado de un objeto y su reacción ante otros eventos.

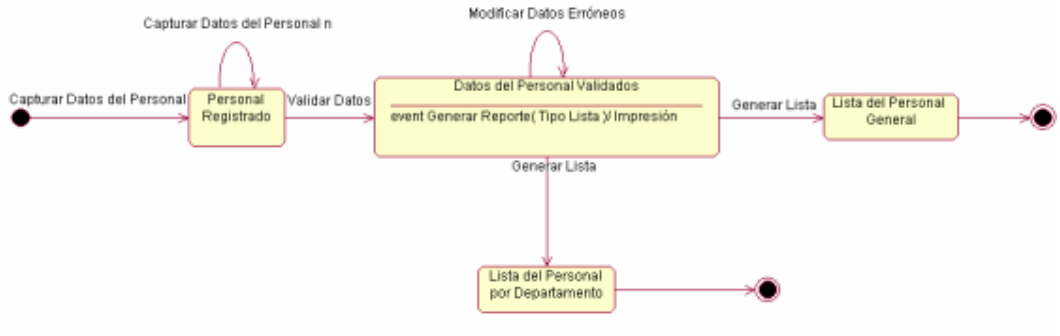


Figura 3.13 Diagrama de Estados para el registro de Personal

El proceso inicia con el registro de los datos del personal que labora en UNICA, el cual se realizará para cada uno, a través de dos interfaces; la primera es la del administrador quien tendrá la posibilidad de validar y m0dificar datos erróneos de modo inmediato, la segunda estará disponible vía Internet, misma que será alimentada por el personal, una vez registrada su información esperará a que ésta sea validada por su jefe inmediato. La principal validación compete al horario de trabajo, mismo que si no cumple con lo dispuesto en el reglamento deberá realizar la modificación pertinente, en caso contrario solo permitirá realizar modificación de sus datos personales. Una vez que la información ha sido recopilada se procede a generar una lista general del personal o bien una lista por departamento.

Toda la información que se almacene estará disponible para todos y cada uno de los jefes de departamento para su consulta.

Figura 3.14 Diagrama de Estados para el registro de Cursos

El administrador del sistema, se encargará de realizar el registro de los cursos que se imparten en UNICA. Inicialmente se capturan los antecedentes previos a los cursos para proceder a registrar los cursos que se impartirán en un período determinado, para después generar una lista general o detalle del curso el cual incluye un temario.

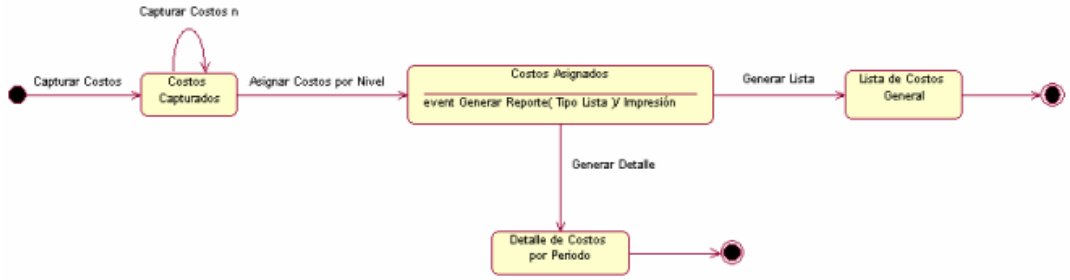


Figura 3.15 Diagrama de Estados para el registro de Costos

Para interactuar con el sistema SICC, es necesario realizar el registro de los costos que se han de cobrar para cada uno de los grupos creados por este, una vez registrados, en la creación de grupos se les asignará un nivel en función de su costo. Una vez concluido el proceso se generará una lista general o el detalle de costos por periodo.

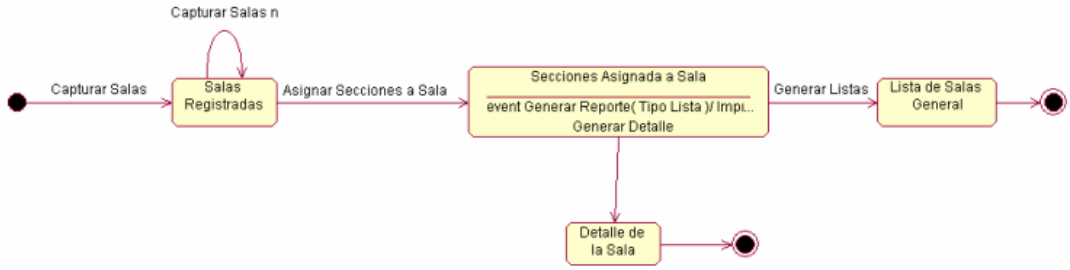


Figura 3.16 Diagrama de Estados para el registro de Salas

Otro proceso que interactúa con el SICC y que también es realizado por el administrador del SAPIUN es el registro de las salas, a las que se les asocia una o más secciones para la asignación de los grupos creados en el SICC. Una vez registrada la información se genera una lista general o bien un detalle por sala, con las características de los equipos y software con el que cuenta.

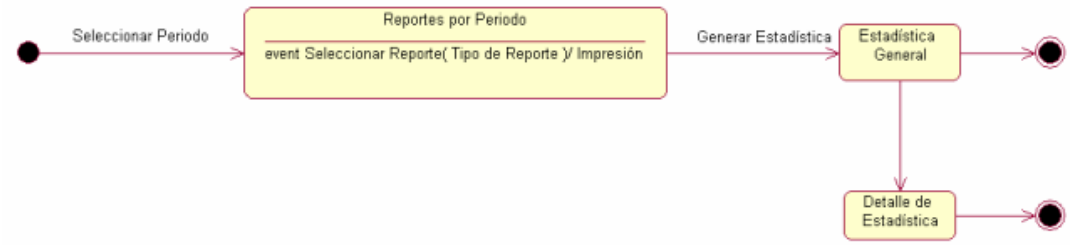


Figura 3.17 Diagrama de Estados para la generación de Estadísticas

Como parte final se deberán generar los reportes de estadísticas por período, para lo cual es necesario seleccionar el tipo de reporte (Por grupo, Tipo de alumno, Ingresos) general o bien profundizar en el detalle.

3.4. Implementación del modelo OMT y UML en un esquema de Base de Datos

El diagrama de clases presenta un mecanismo de implementación neutral para modelar los aspectos de almacenado de datos del sistema. Las clases persistentes, sus atributos, y sus relaciones pueden ser implementadas directamente en una base de datos orientada a objetos. Aun así, en el entorno de desarrollo actual, la base de datos relacional es el método más usado para el almacenamiento de datos, por lo que el diagrama de clases se puede usar para modelar la estructura lógica de la base de datos, con clases representando tablas, y atributos de clase representando columnas. El diagrama de clases de UML se puede usar para modelar algunos aspectos del diseño de bases de datos relacionales, pero no cubre toda la semántica involucrada en el modelado relacional, mayoritariamente la noción de atributos clave que relacionan entre sí las tablas unas con otras. La intención es construir un modelo lógico conforme a las reglas de normalización de datos.

Para el SAPIUN se utilizó una base de datos relacional basada en la identificación de las entidades y de las relaciones que se dan entre ellas y que deseamos modelar; lo cual nos permite representar de forma abstracta los datos que se pretenden almacenar.

Los elementos que constituyen el modelo relacional son:

- ✓ *Entidad*: Objeto, real o abstracto, distinguible de otros objetos. Al grupo de entidades con cualidades similares acerca de los cuales se almacena información se le denomina TIPO (o, simplemente, conjunto de entidades).
- ✓ *Atributo*: Propiedad asociada a un conjunto de entidades (esto es, mediante los atributos representamos propiedades de los objetos). Para cada atributo hay un conjunto de valores permitidos llamado DOMINIO.
- ✓ *Clave*: Conjunto de atributos que permite identificar unívocamente a una entidad dentro de un conjunto de entidades.
- ✓ *Relación (conexión o asociación*): Conexión semántica entre dos conjuntos de entidades.

A continuación se muestra el modelo relacional de la base de datos que utilizan el SAPIUN y el SICC, cabe mencionar que ambos sistemas comparten la misma base de datos debido a que la funcionalidad del SICC depende en gran medida de los datos que se almacenan por medio del SAPIUN.

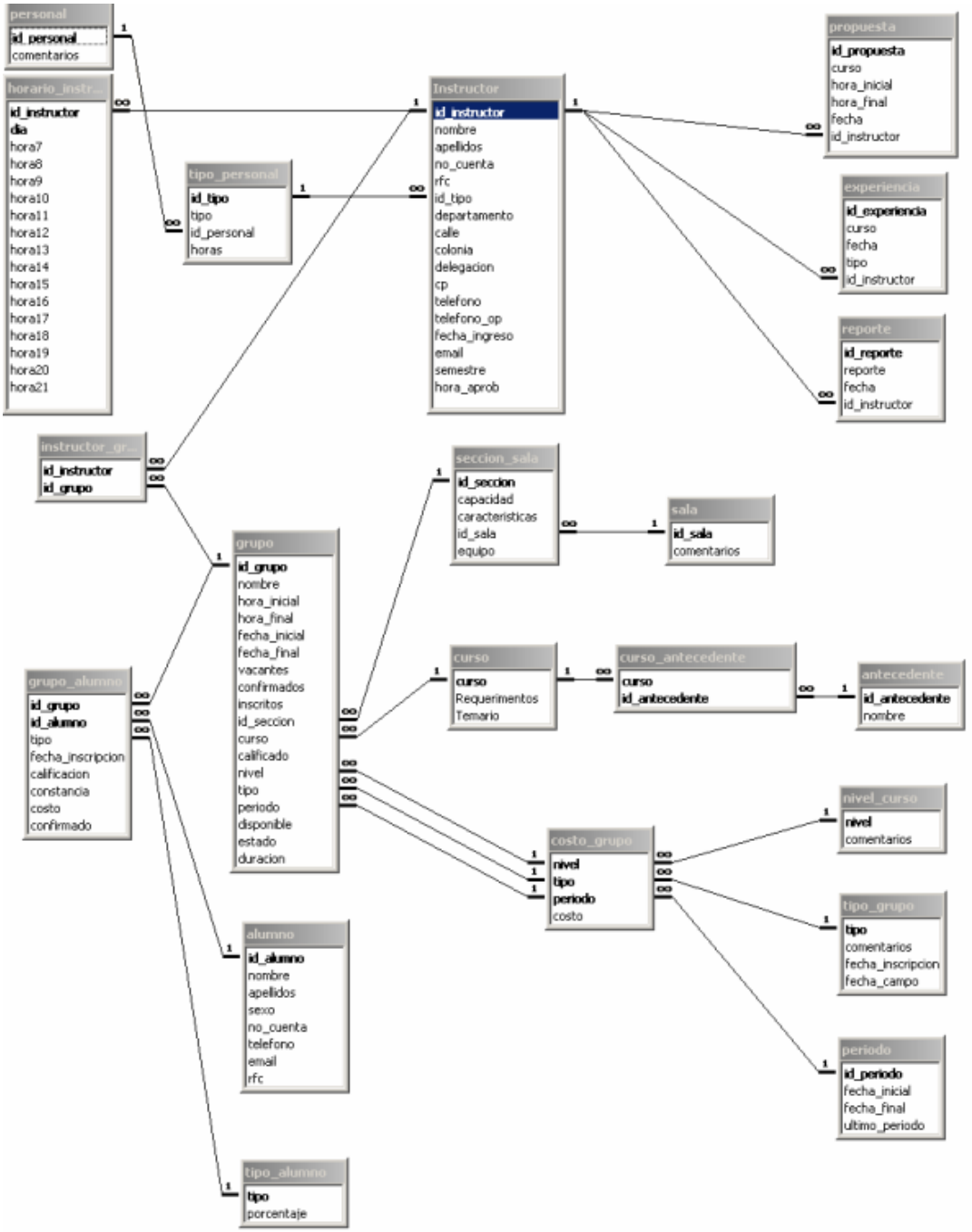


Figura 3.18 Diagrama Entidad Relación del SAPIUN

Programación con Visual Basic y su Integración con el SICC

Capítulo IV

PROGRAMACIÓN DEL SAPIUN CON VISUAL BASIC Y SU INTEGRACIÓN CON EL SICC

4.1. Visual Basic, relación con OMT y UML

Por su facilidad de uso, Visual Basic (VB) nos permite abordar cualquier tipo de proyecto sin importar su complejidad; sin embargo, el desarrollo siempre tiene que estar basado en un buen análisis y diseño que dependerán directamente de la elección de una metodología. Como ya se ha visto en capítulos anteriores para el caso del SAPIUN se utiliza OMT.

La elección de VB como lenguaje de programación se debe principalmente a que se puede considerar como un lenguaje orientado a objetos, aunque técnicamente hablando muchos consideran que no cumple con todos los elementos propios de los lenguajes orientados a objetos, ya que no soporta la herencia, y que por definición es la principal característica en un lenguaje orientado a objetos. La herencia como tal es una poderosa herramienta que nos permite construir varios objetos desde una clase base; por ejemplo, se puede crear un objeto perro, gato o ratón desde una clase común llamada animal. La clase animal cuenta con propiedades como lo son la estatura, tamaño, color y métodos como son caminar, dormir y correr.

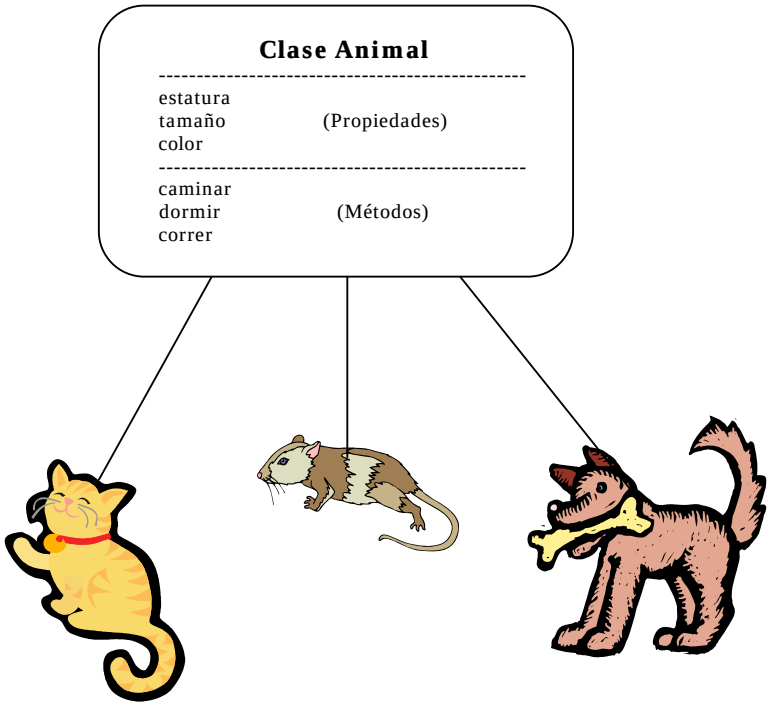


Fig. 4.1 Herencia mediante clase base

Desafortunadamente en VB no se tiene una clase base de donde heredar (clase Animal), pero hace uso de clases idénticas en una colección para crear el objeto perro, gato y ratón, que toman como base la definición de una clase para después incluirlas en una colección. De tal forma que se hace uso de un modelo de objetos; esto describe la contención, es decir muestra cómo objetos complejos contienen colecciones de otros objetos.

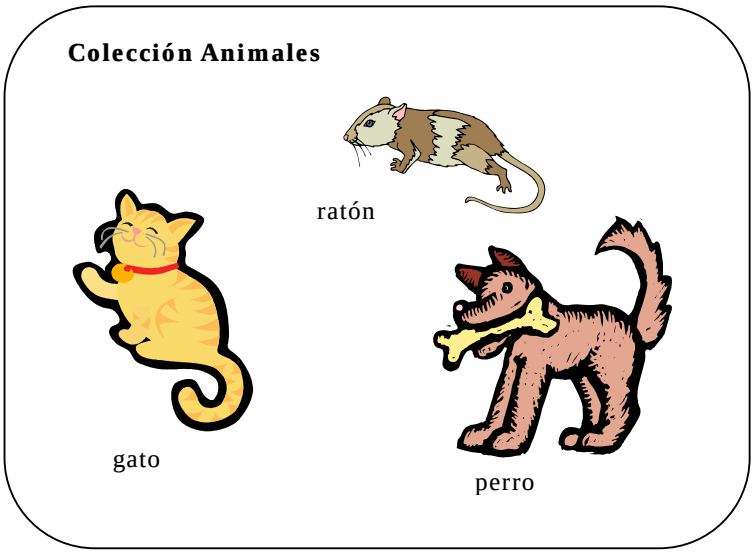


Fig. 4.2 Herencia mediante colecciones

La escasez de herencia no evita que los objetos tengan relación, al contrario lo dificulta, aunque las dependencias entre los objetos se definen mediante propiedades y colecciones de objetos VB ofrece varias alternativas, que si bien no son parte de una metodología orientada a objetos los componentes que nos da si lo son, ya que están diseñados desde UML.

Entre las ventajas que éstos ofrecen, se encuentran:

- ✓ Realizan tareas específicas de manera eficiente y confiable, dependiendo de cómo sean utilizados.
- ✓ Son lo bastante robustos para trabajar con carga o en circunstancias inesperadas.
- ✓ Están totalmente protegidos, (podemos sustituir el componente sin afectar otros componentes que están dentro del sistema).
- ✓ Son de fácil mantenimiento y de código reutilizable.
- ✓ Requieren de bajo presupuesto.

4.2. Implementación de clases y objetos con VB

Como ya se ha visto una clase o módulo de clase es una plantilla o molde que servirá para definir todas las propiedades y métodos, tanto privados como públicos, y las características de un objeto. La ventaja de usar clases es que es posible encapsular todo el código y tratar la clase como si se tratara de una caja negra, es decir, se sabe que es lo que hacen, pero no es imprescindible saber el código que se usa para que funcione. Por lo tanto una de las características principales que una clase debe tener es que no dependa del exterior, es decir, que sea autosuficiente; todo la información que necesite se debe suministrar mediante las propiedades que dicha clase exponga al exterior, (mediante las propiedades públicas) y de igual manera, toda la información que un objeto deba mostrar al mundo exterior, se haga mediante las propiedades y métodos que dicha clase exponga al exterior.

4.2.1. Elementos que componen una clase

VB permite definir clases de objetos utilizando módulos de clase. Precisamente la clave de la programación orientada a objetos está en abstraer las propiedades y los métodos comunes a un conjunto de objetos y almacenarlos en una clase. Las propiedades y métodos, reciben genéricamente el nombre de *miembros de clase* y cada módulo de clase define una sola clase; las *propiedades* son datos relativos al objeto que pueden ser variables convencionales, como cadenas y enteros, pero también pueden ser objetos o colecciones y los métodos son las acciones que puede realizar el objeto, son similares a las subrutinas o funciones.

Para crear las propiedades, existen dos formas:

- ✓ *Declarando las propiedades como variables*: Cuando se declara una variable Public en un módulo de clase, dicha variable se convierte en una propiedadad. Cuando se declara una variable como Private, la ventaja es que solo puede ser modificada dentro del módulo de clase al que pertenece, para hacerla visible se utiliza un procedimiento Property.
- ✓ *Creando un procedimiento Property*: La ventaja es que es posible hacer comprobaciones extras que con las variables sería imposible hacer. Por ejemplo, al asignar un valor a la propiedad e-mail, se podría comprobar que dicho valor contenga el signo arroba (@). Para poder utilizar las propiedades de esta forma exiten tres clases de procedimientos de propiedad: *Property Get* que retorna el valor de una propiedad, *Property Let* que establece el valor de una propiedad y *Property Set* que establece una referencia a un objeto. Estos procedimientos son generalmente utilizados por parejas: esto es, *Property Get* con *Property Let* y *Property Get* con *Property Set* y la sintaxis utilizada para invocarlos es*:*

  Property Get *variable = [objeto.] nombre_propiedad[(argumentos)]*

  Property Let *[objeto.] nombre_propiedad[(argumentos)]=argumento*

  Property Set *Set [objeto.] nombre_propiedad[(argumentos)] = variable*

A continuación se muestra la implementación de Property Get y Property Let en la clase CInstructor del SAPIUN:

```
Option Explicit
'Definicion de la clase instructores
Private mintid_instructor As Integer
Private mstringnombre As String
Private mstringapellidos As String
...
```

Declaración de propiedades como elementos privados

```
Public Property Let id(ByVal id_instructor As Integer)
mintid_instructor = id_instructor
End Property
Public Property Get id() As Integer
id = mintid_instructor
End Property
Public Property Let nombre(ByVal nombre_instructor As String)
mstringnombre = nombre_instructor
End Property
Public Property Get nombre() As String
nombre = mstringnombre
End Property
Public Property Let apellidos(ByVal apellidos_instructor As String)
mstringapellidos = apellidos_instructor
End Property
Public Property Get apellidos() As String
apellidos = mstringapellidos
End Property
...
```

Acceso a las propiedades mediante el uso de Get y Let

Ahora implementando PorpertyGet y Porperty Set en la clase CBDInstructor:

```
...
'Agregar clase y colección para instructores.
...
Private mvarCExperiencia As CExperiencia
...
```

Declaración de propiedades como elementos privados

```
Public Property Get CExperiencia() As CExperiencia
    If mvarCExperiencia Is Nothing Then
        Set mvarCExperiencia = New CExperiencia
    End If
    Set CExperiencia = mvarCExperiencia
End Property
Public Property Set CExperiencia(vData As CExperiencia)
    Set mvarCExperiencia = vData
End Property
```

Acceso a los objetos referenciados por la clase mediante el uso de Get y Set

Ya que los procedimientos e incluso las variables públicas, que se convierten en métodos y propiedades respectivamente, sólo son accesibles o visibles mientras la clase exista; los módulos de clase presentan un constructor y un destructor, como se especifica a continuación:

- ✓ Initialize (inicializar): ocurre cuando se crea una instancia de una clase. Se usa, generalmente para inicializar cualquier dato usado por la instancia de una clase en el código.
- ✓ Terminate (terminar): ocurre cuando toda referencia a una instancia de una clase son removidas de memoria al establecer todas las variables que la refieren a un objeto a Nothing o cuando la ultima referencia al el objeto se encuentra fuera de alcance. Usado generalmente para limpiar la memoria de objetos creados dentro de esta instancia y generar un error si hay alguna anomalía o guardar alguna información del objeto, etc.

La sintaxis es la siguiente:

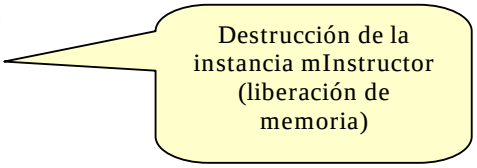
```
Private Sub Class_Initialize()
'Iniciar el objeto
End Sub
Private Sub Class_Terminate()
'Operaciones para destruir el objeto
End Sub
```

En la clase CBDInstructor del SAPIUN esta declaración es de la siguiente forma:

```
Public mInstructor As CInstructor
...
Public Sub Class_Initialize()
'Para instructores
Set mInstructor = New CInstructor
...
End Sub
```

Creación de la instancia mInstructor

```
Private Sub Class_Terminate()
Set mInstructor = Nothing
...
End Sub
```



Los métodos de la clase son procedimientos y funciones definidas en el módulo; en la clase CBDInstructor se implementó el siguiente método para la modificación de datos de instructor:

```
Public Function Modificar(ByRef instructor_mod As CInstructor)
Dim Horario As CHorario
...
Set rshorario = New ADODB.Recordset
For Each Horario In instructor_mod.Horario
   rshorario.Source = "SELECT * FROM horario_instructor WHERE " _
                    + "id_instructor='" + CStr(Trim(instructor_mod.id)) + "'" _
                    + " AND dia='" + UCase(Trim(Horario.dia)) + "'"
   rshorario.ActiveConnection = cn
   rshorario.LockType = adLockPessimistic
   rshorario.Open
   rshorario!hora7 = Horario.hora7
   rshorario!hora8 = Horario.hora8
   ...
   rshorario!hora20 = Horario.hora20
   rshorario.Update
   rshorario.Close
Next
cn.Close
End Function
```

4.2.2. Implementación de colecciones

Un objeto Colección es un tipo especial de objeto cuyos elementos se pueden referenciar de forma individual, y además, puede, hacer referencia a la propia colección; como una colección es un objeto, tiene que crearse como una instancia de una clase incorporada en VB que a su vez se denomina *Collection*.

La sintaxis para la creación de una instancia de la clase *Collection* es la siguiente:

```
Dim colX As New Collection
```

En otras palabras una colección es un tipo de objeto que almacena un conjunto de referencias a otros objetos. Por ejemplo, la tabla de contenido de un libro es una colección, ya que almacena un conjunto de referencias a los capítulos y a sus números de página, o bien, un directorio telefónico también es una colección porque almacena un conjunto de referencias a personas y negocios.

La ventaja de tener este conjunto de referencias es que es posible desplazarse por ellos de forma rápida y coherente, esto implica actuar en todos sus miembros, uno tras otro; explorar una tabla de contenido, marcar los elementos de una lista de temas pendientes y recorrer un directorio telefónico, son ejemplos de desplazamiento por una colección.

Una colección tiene los siguientes métodos:

- *Item*: Es el método por omisión de una colección; es la forma para hacer referencia (o devolver) un elemento específico de una colección. Su sintaxis es:

  ```
  ObjetoColeccion.Item(índice)
  ```

  El parámetro índice especifica la posición de un miembro de la colección o una clave que puede utilizar para acceder rápidamente a un elemento de la colección.

- *Add*: Una vez creada la colección se tiene que utilizar la palabra Add para añadir elementos a la colección. Su sintaxis es:

  ```
  ObjetoColeccion.Add item[, key As String][, before As Long][, after As Long]
  ```

- *Remove*: Cuando se necesite eliminar elementos de una colección se tiene que utilizar el método Remove. Su sintaxis es:

  ```
  ObjetoColeccion.Remove index
  ```

A continuación se muestra la implementación de la colección colInstructor en el SAPIUN:

```
'variable local para contener colección
Private mCol As Collection
Public Function Add(Semestre As Integer, Fecha_Ingreso As Date, E_mail As String, Telefono As String, ... , apellidos As String, nombre As String, ... , Optional ByVal Experiencia As colExperiencia, Optional ByVal Horario As colHorario, Optional sKey As String) As CInstructor
    'crear un nuevo objeto
    Dim objNewMember As CInstructor
    Set objNewMember = New CInstructor
    'establecer las propiedades que se transfieren al método
    objNewMember.Semestre = Semestre
    objNewMember.Fecha_Ingreso = Fecha_Ingreso
    objNewMember.E_mail = E_mail
    objNewMember.Telefono = Telefono
    ...
    objNewMember.apellidos = apellidos
    objNewMember.nombre = nombre
    ...
    If IsObject(Experiencia) Then
        Set objNewMember.Experiencia = Experiencia
    Else
        objNewMember.Experiencia = Experiencia
    End If
    If IsObject(Horario) Then
        Set objNewMember.Horario = Horario
    Else
        objNewMember.Horario = Horario
    End If
    If Len(sKey) = 0 Then
        mCol.Add objNewMember
    Else
        mCol.Add objNewMember, sKey
    End If
    'devolver el objeto creado
    Set Add = objNewMember
    Set objNewMember = Nothing
End Function
```

```
Public Property Get Item(vntIndexKey As Variant) As CInstructor
    'se usa al hacer referencia a un elemento de la colección
    'vntIndexKey contiene el índice o la clave de la colección,
    'por lo que se declara como un Variant
    'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
    Set Item = mCol(vntIndexKey)
End Property

Public Property Get Count() As Long
    'se usa al obtener el número de elementos de la
    'colección. Sintaxis: Debug.Print x.Count
    Count = mCol.Count
End Property

Public Sub Remove(vntIndexKey As Variant)
    'se usa al quitar un elemento de la colección
    'vntIndexKey contiene el índice o la clave, por lo que se
    'declara como un Variant
    'Sintaxis: x.Remove(xyz)
    mCol.Remove vntIndexKey
End Sub
```

4.3. Funciones y procedimientos

La base de una aplicación en VB la forman sus procedimientos conducidos por eventos; esto es, el código que es invocado cuando un objeto reconoce que ha ocurrido un determinado evento. Cuando varios procedimientos conducidos por eventos necesiten ejecutar un mismo proceso, la mejor forma de proceder es colocar el código común en un procedimiento estándar, perteneciente a un módulo estándar, de esta forma se elimina la necesidad de duplicar código.

Un procedimiento estándar es invocado cuando se hace una llamada explícita al mismo y puede escribirse como procedimiento Sub o como función Function. Un procedimiento Sub es un segmento de código independiente del resto, que una vez llamado por el programa, ejecuta un número determinado de instrucciones, sin necesidad de devolver ningún valor al mismo. Por otra parte una función es una forma especial de realizar un procedimiento, es decir es un procedimiento al que le pasamos uno o varios parámetros con los que realizará una operación y obtendrá un resultado que puede leerse desde otra parte de la aplicación.

La sintaxis correspondiente a una función es la siguiente:

```
[Static] [Private] Function nombre ([ parámetros]) [As tipo]
[ sentencias]
[ nombre = expresion]
[Exit Function]
[ sentencias]
[ nombre = expresion]
End Function
```

Donde nombre es el nombre de la función. Será de un tipo u otro dependiendo del dato que devuelva, para especificar el tipo se utiliza la cláusula As Tipo (Integer, Long, Single, Double, Currency, String o Variant). Los parámetros son los argumentos que son pasados cuando se llama a la función. VB asigna el valor de cada argumento en la llamada al parámetro que ocupa su misma posición. El nombre de la función, que es el valor de retorno, actúa como una variable dentro del cuerpo de la función. El valor de la variable expresión es almacenado en el propio nombre de la función. Exit Function permite salir de una función antes de que ésta finalice y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada a la función. La sentencia End Function marca el final del código de la función y, al igual que la Exit Function, devuelve el control del programa a la sentencia siguiente a la que efectuó la llamada, pero lógicamente una vez finalizada la función.

La llamada a una función se hace de diversas formas. Por ejemplo, una de las más usuales es la siguiente:

```
variable = nombre([argumentos])
```

Donde argumentos son una lista de constantes, variables o expresiones separadas por comas que son pasadas a la función. En principio, el número de argumentos debe ser igual al número de parámetros de la función. Los tipos de los argumentos deben coincidir con los tipos de sus correspondientes parámetros, de lo contrario puede haber fallos importantes en la ejecución del programa. En cada llamada a una función hay que incluir los paréntesis, aunque ésta no tenga argumentos.

La sintaxis que define un procedimiento Sub es la siguiente:

```
[Static] [Private] Sub nombre [( parámetros)]
[ sentencias]
[Exit Sub]
[ sentencias]
End Sub
```

La explicación es análoga a la dada para funciones. La llamada a un procedimiento Sub puede ser de alguna de las dos formas siguientes:

```
Call nombre[(argumentos)]
```

o bien, sin pasar los argumentos entre paréntesis, sino poniéndolos a continuación del nombre simplemente separados por comas:

```
nombre [argumentos]
```

A diferencia de una función, un procedimiento Sub no puede ser utilizado en una expresión pues no devuelve ningún valor. Por supuesto una función puede ser llamada al modo de un procedimiento Sub, pero en este caso no se hace nada con el valor devuelto por la función. A continuación se muestra la implementación de un procedimiento y una función en el SAPIUN.

```
Public Sub obtener_val_horario()
'Procedimiento que toma los valores de horario de los instructores
Dim j As Integer
With frmInstructores
   If instructor_i.Horario.Count <> 0 Then
      For j = 1 To 5
         instructor_i.Horario.Remove (1)
      Next j
   End If
   For j = 1 To 5
          instructor_i.Horario.Add .mfgHorario.TextMatrix(0, j),
         …
         .mfgHorario.TextMatrix(15, j)
   Next j
End With
End Sub
```

```
Public Function verificar_horario() As Boolean
'Procedimiento que Verifica que el instructor cumpla con sus
'20 horas de trabajo en UNICA
Dim i As Integer, j As Integer
Dim horas_trabajo As Single
With frmInstructores
   For i = 1 To 15
      For j = 1 To 5
         Select Case Trim(.mfgHorario.TextMatrix(i, j))
            Case "T"
               horas_trabajo = horas_trabajo + 1
            Case "t", "t/c", "c/t"
               horas_trabajo = horas_trabajo + 0.5
         End Select
      Next j
   Next i
   'Servicio_Horario
   If horas_trabajo < CInt(Trim(Left(lblHoras.Caption, 3))) Then
      MsgBox "No completa sus " + Trim(lblHoras.Caption) + " de trabajo en
UNICA, Verifique su horario", vbExclamation + vbOKOnly, "Advertencia SAPIUN"
   Else
      verificar_horario = True
   End If
End With
End Function
```

Como puede observase en los ejemplos anteriores, se describe brevemente como se implementa el diagrama de clases visto en el apartado 3.2 del capitulo anterior, utilizando VB.

En la creación del SAPIUN según el diagrama de clases propuesto se generan primeramente las clases bases CInstructor, CHorario, CExperiencia y CPropuesta, todas éstas manejan propiedades del tipo Private, por lo que fue necesario crear los métodos de acceso mediante la utilización de Get, Let y Set. Como su nombre lo indica las clases bases en su función no son la parte principal de acceso al programa, sino la estructura general de repositorio de información por cada personal, como lo que se desea es tener la información de todo el personal como un objeto individual fue necesario crear las colecciones que almacenarán la información para cada persona, y que toman como base las clases genéricas para su creación dando como resultado ColInstructor, ColHorario, ColExperiencia y ColPropuesta que a su vez interactuán con el programa a través de la clase abstracta CBDInstructor. Ésta hereda las características del personal por medio de las colecciones, de este modo tiene toda la

información que necesita para administrar al personal con los métodos de Obtención, Inserción, Modificación y Eliminación.

En la programación del sistema sólo se utilizará una clase abstracta por módulo con lo que se pretende tener acceso a la información desde cualquier parte del código (procedimientos y funciones); haciéndolo cada vez más portable y de fácil mantenimiento. Es decir para su creación se utilizo el siguiente código:

```
'Creación de la Instancia
Public mInstructores As New CBDInstructor
Set mInstructores=New CBDInstructor
```

Y su utilización dentro del programa se realizo a través del siguiente código:

```
'Acceso a un instructor, mediante el lugar que ocupa en la colección (index).
mInstructores.Instructor(index)
'Acceso a una propiedad del instructor
mInstructores.Instructor(1).nombre
'Acceso al día y hora en la que labora
mInstructores.Instructor(1).mHorarios.Horario(idx_dia).hora7
```

Siguiendo éste proceso de implementación y dado que el sistema se creo de forma modular se realizará el mismo procedimiento para la creación de las clases abstractas CBDCursos, CBDSalas, CBDCostos y CReportes. De tal forma que el sistema mantendrá una relación estrecha con todas y cada una de sus partes, sin que interfieran unas con otras, pues su relación se mantendrá integra a través de los métodos de obtención, inserción, modificación y eliminación de la información.

4.4. Interfaces del SAPIUN

Una interfaz es el medio de comunicación entre el usuario y el programa, ya que permite la interacción con los diferentes elementos que la conforman (botones de orden, cuadros de texto, menús, combos, etc.)

4.4.1. Interfaces de VB

A continuación se muestran las interfaces que conforman el sistema de administración del SAPIUN:

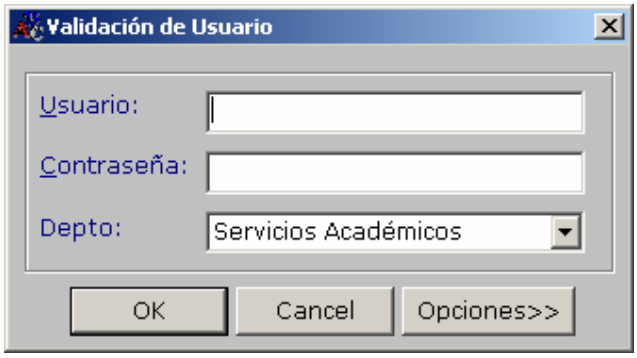


Fig. 4.3 Interfaz de Validación

Al ingresar al sistema se muestra la interfaz de validación de usuario; misma que cuenta con un botón de opciones que permite cambiar el origen de datos.

Fig. 4.4 Interfaz de Validación con despliegue de opciones

Una vez que se realizó la validación correspondiente, se mostrará la interfaz de bienvenida:

Fig. 4.5 Interfaz de Bienvenida

Después de permanecer presente durante unos segundos se presentará la siguiente interfaz.

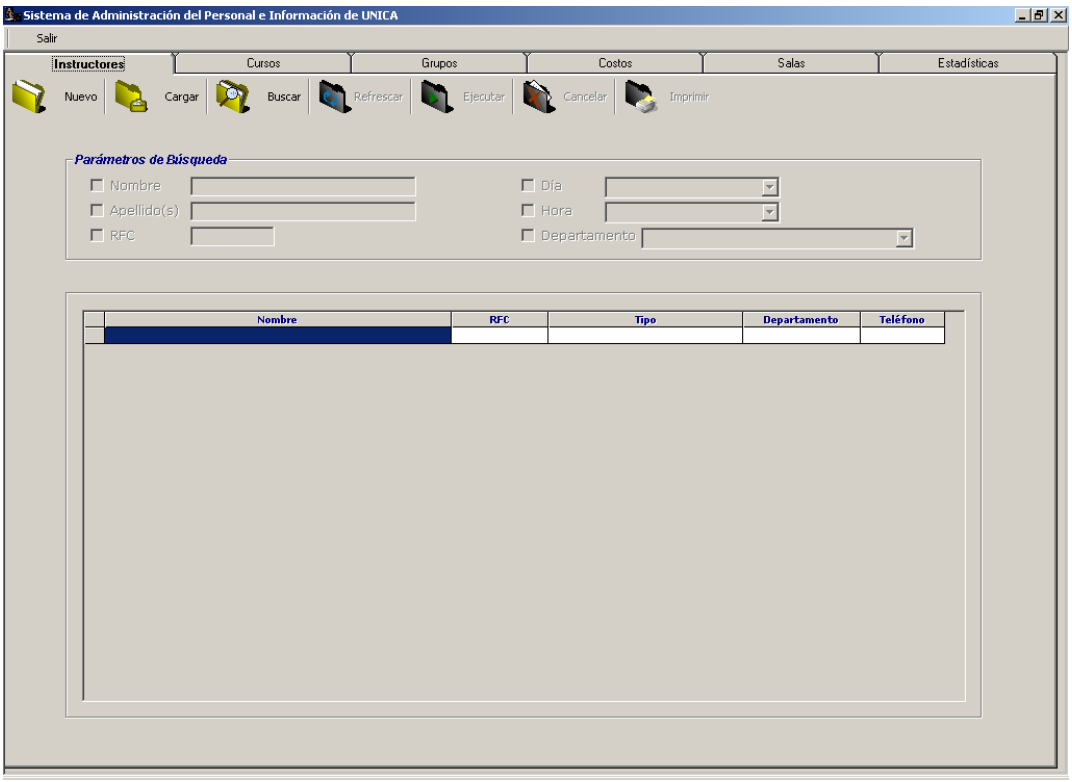


Fig. 4.6 Interfaz Inicial

Ésta interfaz permite visualizar de manera general los módulos con los que cuenta el SAPIUN como son: Instructores, Cursos, Costos, Salas y Estadísticas.

La interfaz de instructores muestra los datos principales de todo el personal que labora en UNICA. Cuenta con una barra de herramientas que permite realizar diferentes tareas, como son:

Nuevo: Ingresa una nueva persona en el sistema.

Cargar: Obtiene la información general de todo el personal que labora en UNICA.

Buscar: Realiza un filtrado de información de acuerdo a los parámetros seleccionados, entre los que se encuentran nombre, apellidos, rfc, día, hora y departamento.

Ejecutar: Realiza la búsqueda de acuerdo a los criterios seleccionados.

Refrescar: Ejecuta nuevamente la búsqueda o el cargado general de datos.

Cancelar: Cancela la búsqueda.

Imprimir: Genera una lista del resultado devuelto en pantalla.

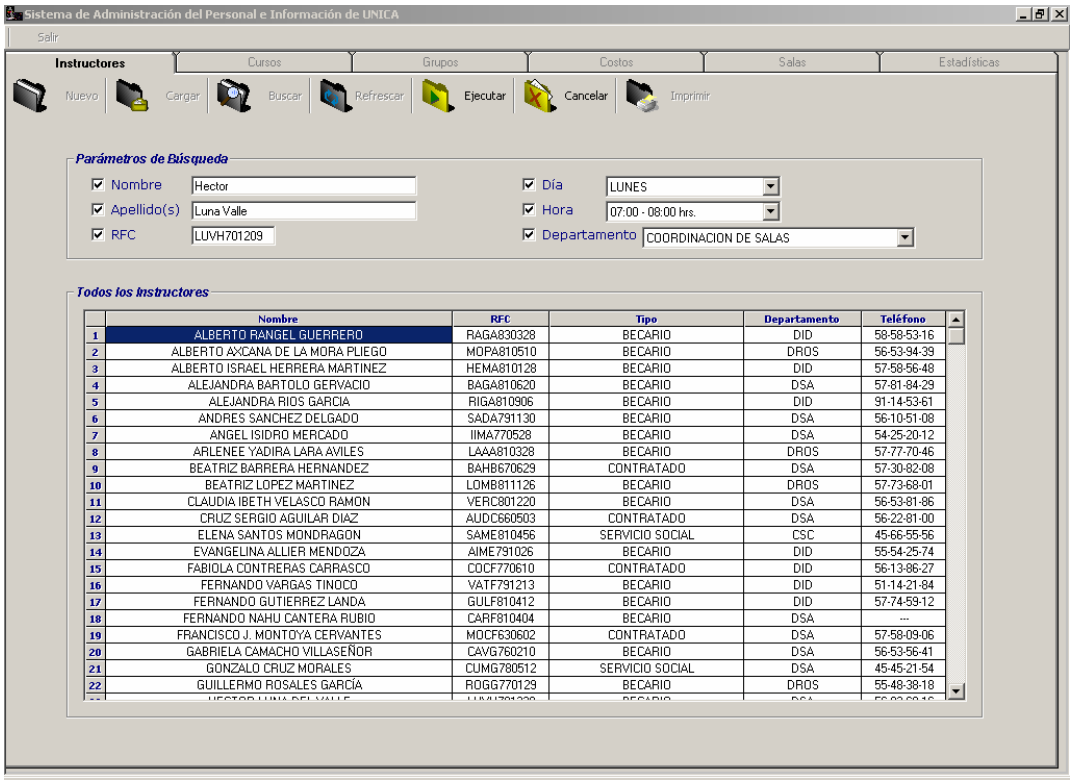


Fig. 4.7 Interfaz de Instructores

Si se desean mostrar todos los datos relacionados con una persona, bastará con hacer doble clic sobre el registro deseado para mostrar la siguiente interfaz.

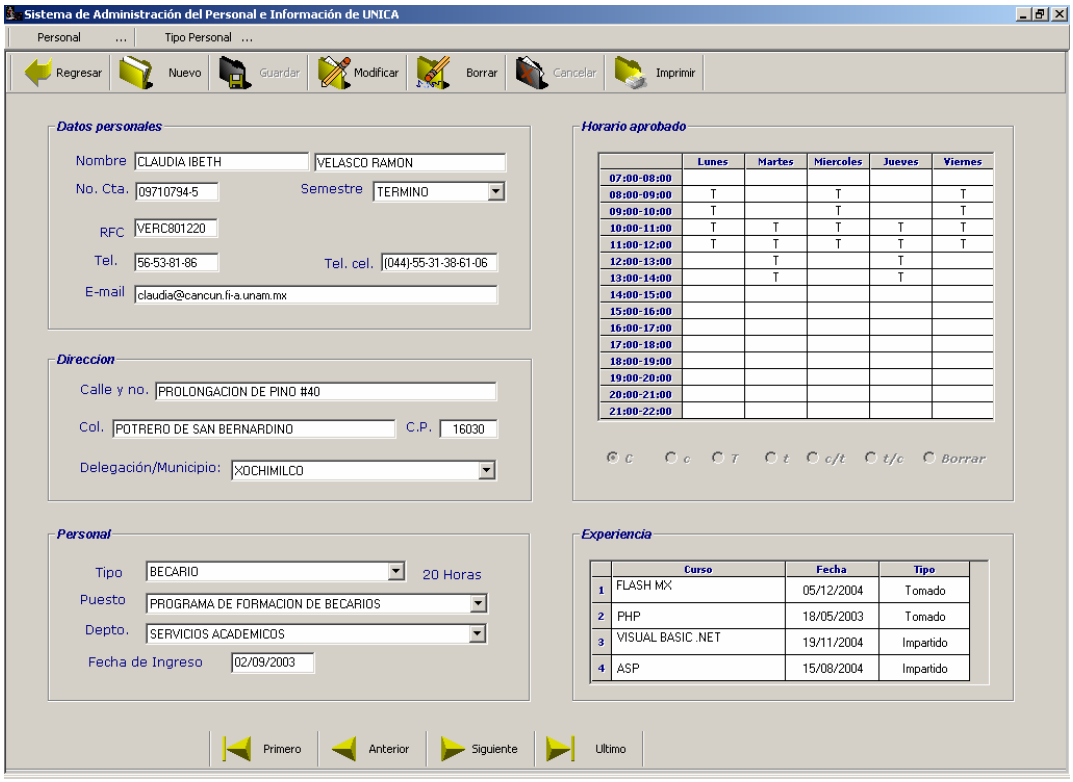


Fig. 4.8 Interfaz de Instructores detallada

Dicha interfaz muestra datos personales, tipo de personal, horario de trabajo y experiencia, además cuenta con una barra de herramientas con las siguientes funciones:

Nuevo: Permite dar de alta un nuevo integrante.

Guardar: Guarda los datos del nuevo integrante o del integrante modificado.

Modificar: Permite realizar un modificación en los datos del personal ya registrado.

Borrar: Elimina la información relacionada con el personal.

Cancelar: Cancela una modificación o inserción que se este llevando a cabo.

Imprimir: Genera un reporte con la información que se visualiza en la interfaz.

Adicionalmente cuenta con una barra de navegación, la cual permite desplazarse entre los diferentes registros. A esta pantalla se puede acceder directamente desde la pantalla inicial si se presiona el botón de nuevo.

La interfaz de cursos muestra un catálogo general de los cursos que imparte UNICA, al igual que la interfaz anterior, cuenta con una barra de herramientas que funciona de manera similar. Los parámetros de búsqueda son: cursos y antecedentes.

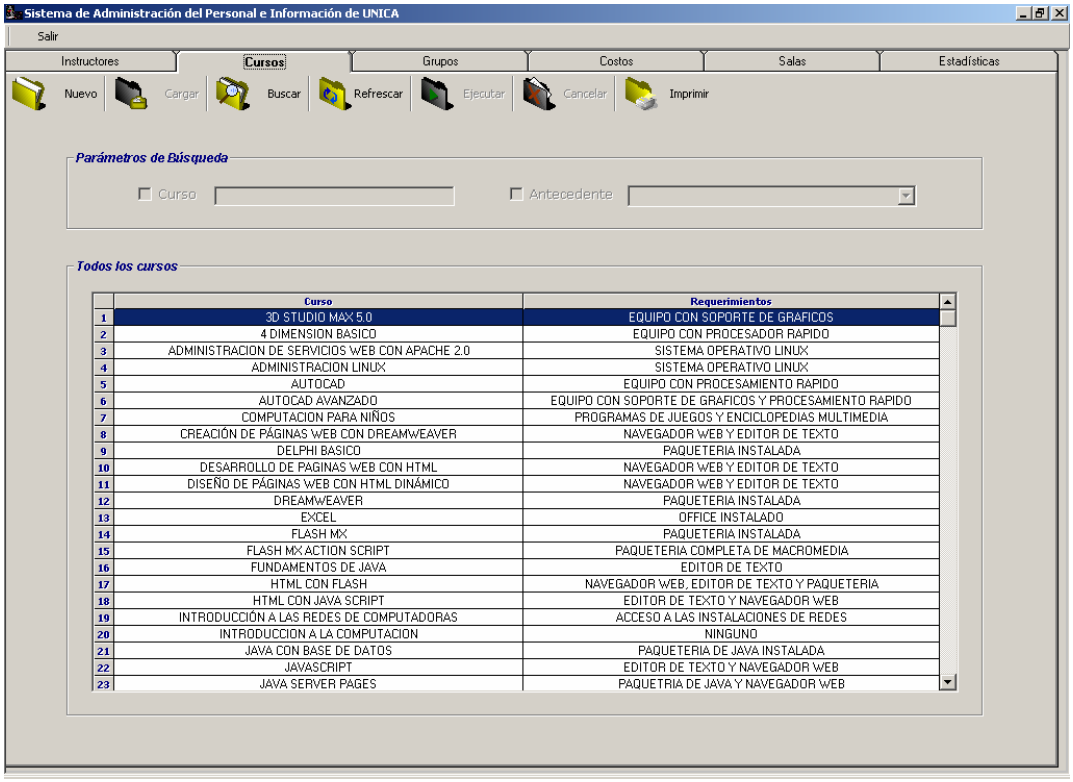


Fig. 4.9 Interfaz de Cursos

Al igual que la interfaz de instructores si se desean mostrar todos los datos relacionados con un curso, bastará con dar doble clic sobre el registro.

Esta interfaz nos muestra los antecedentes, requerimientos y temario con que cuenta el curso, al igual que la anterior interfaz nos muestra una barra de herramientas y una barra de navegación que funcionan de manera idéntica, con la distinción que aquí se trata de cursos.

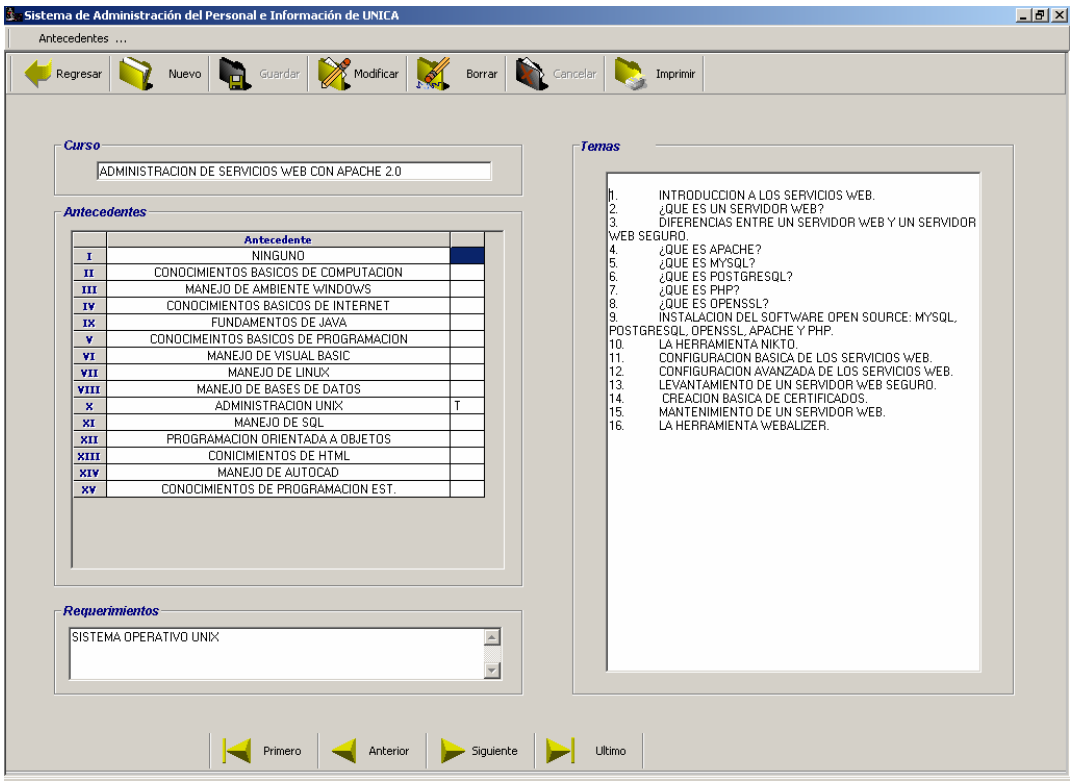


Fig. 4.10 Interfaz de Cursos detallada

La interfaz de costos muestra la información relacionada con los diferentes costos que se asignan, por periodo de impartición, tipo y nivel, a los diferentes cursos que imparte UNICA. De manera similar a las interfaces anteriores cuenta con una barra de herramientas y en este caso los parámetros de búsqueda son: periodo y tipo de curso.

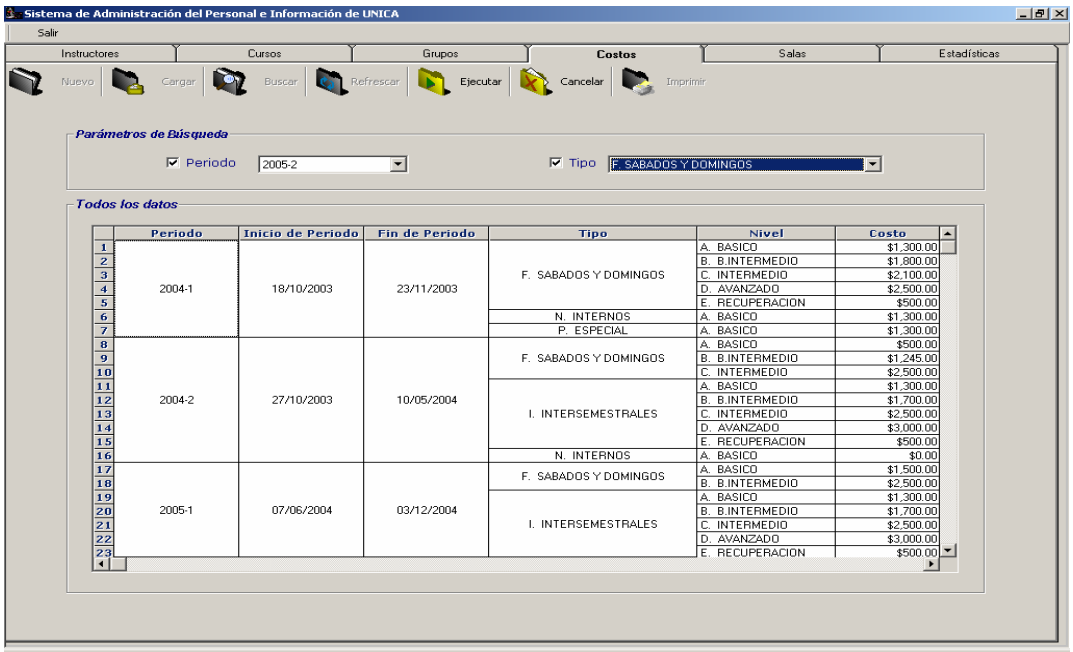


Fig. 4.11 Interfaz de Costos

Como en los casos anteriores para visualizar la información detalla del costo, basta con dar doble clic en el registro deseado; con lo que se muestra la siguiente interfaz, que cuenta, como ya se ha visto en las anteriores, con una barra de herramientas y una barra de navegación que funcionan de la misma manera, solo que en este caso se trata de la información de costos.

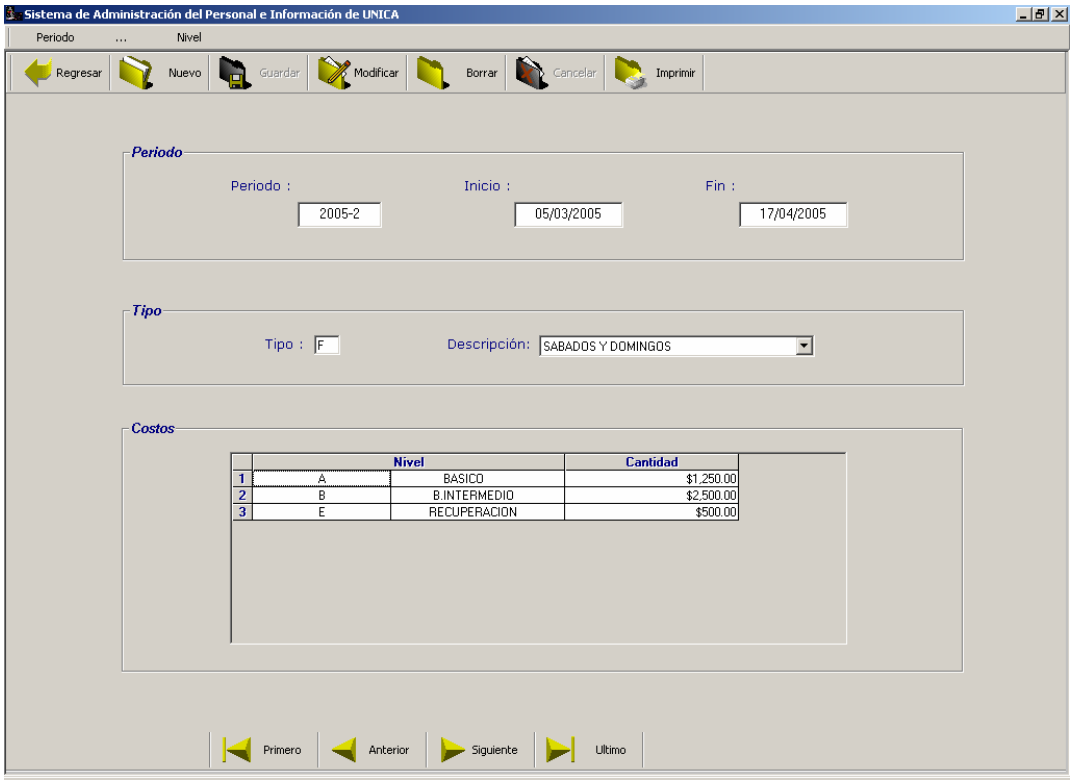


Fig. 4.12 Interfaz de Costos detallada

La interfaz de salas funciona de manera idéntica a las mencionadas anteriormente, en este caso la información que muestra se refiere a las salas y sus secciones, en las que es posible impartir los cursos con los que cuenta UNICA. Los parámetros de búsqueda para esta interfaz son: número se sala, sección o equipo de cómputo.

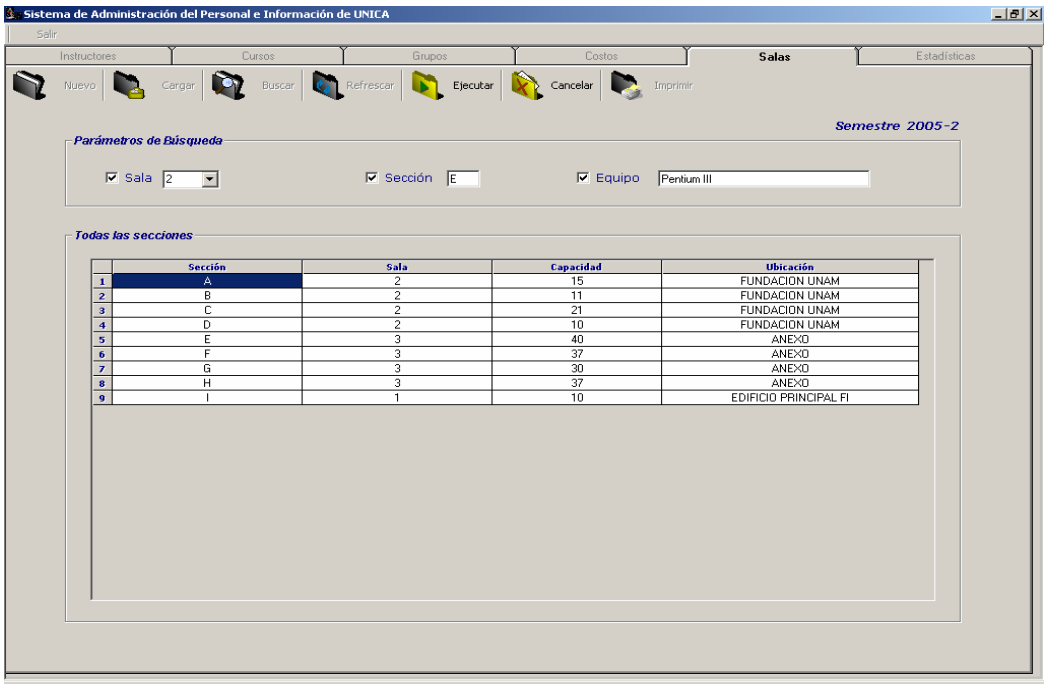


Fig. 4.13 Interfaz de Salas

De manera análoga a las anteriores para ver el detalle de cada sala solo se debe dar doble clic sobre el registro del que se desea tener el detalle; aquí se muestran el tipo de equipo de cómputo con el que cuenta la sala y sus características de software y capacidad.

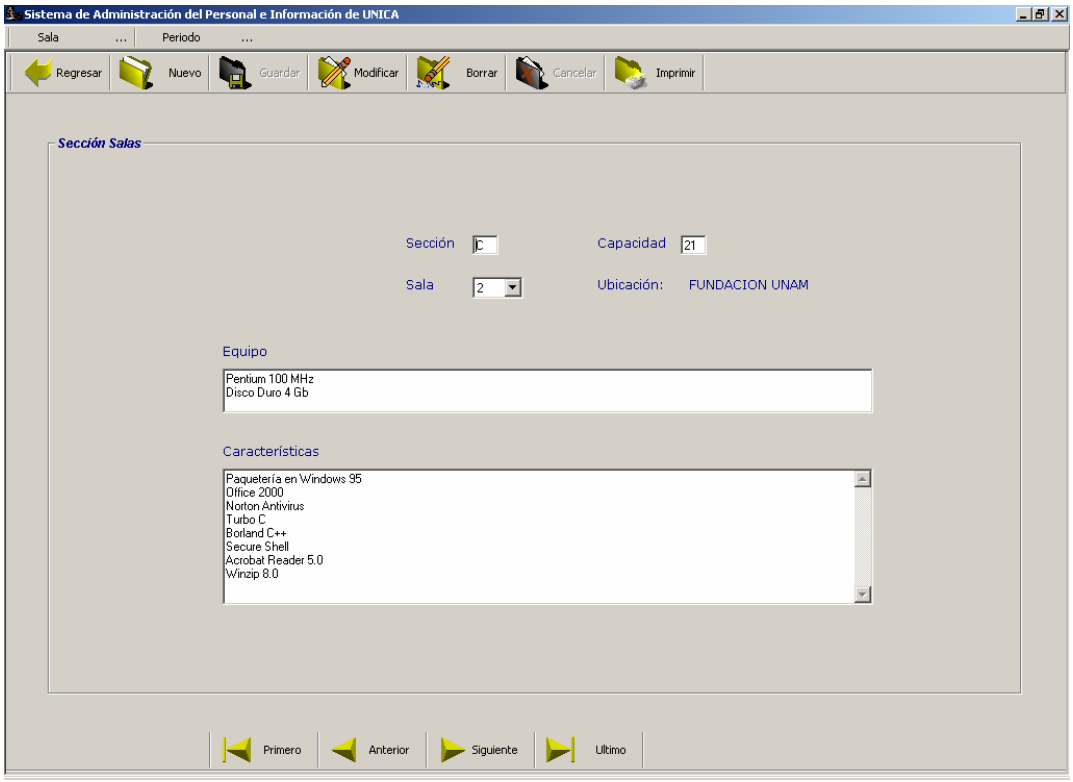


Fig. 4.14 Interfaz de Salas detallada

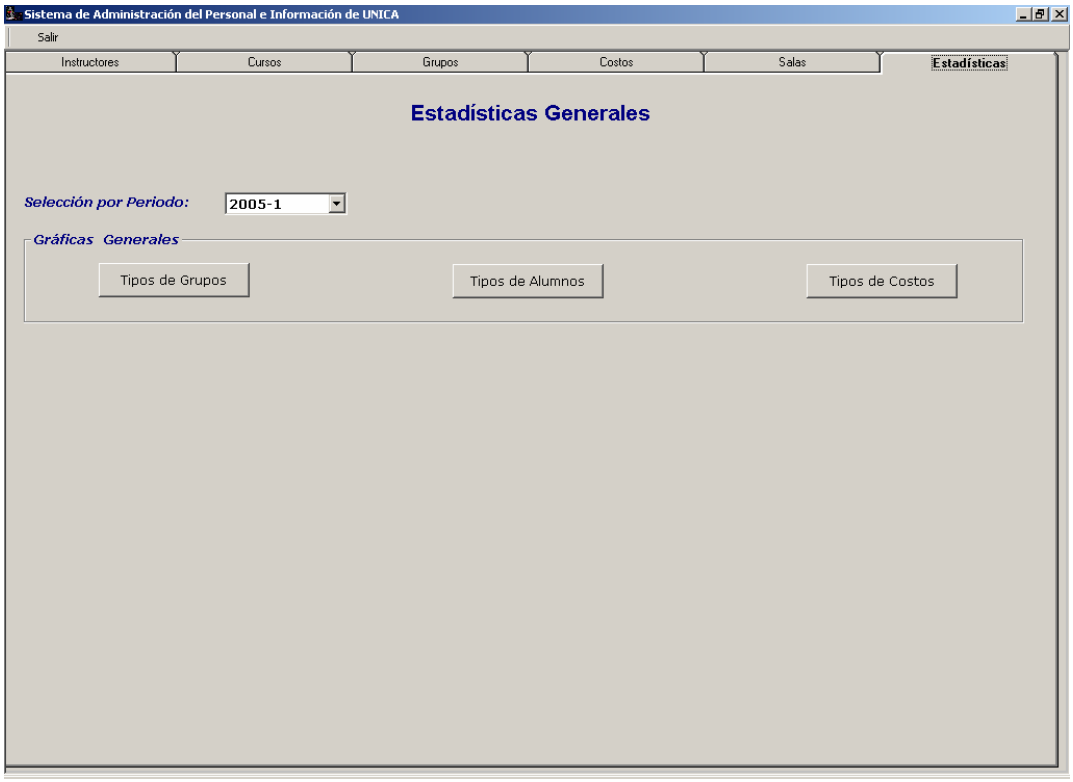


Fig. 4.15 Interfaz de Estadísticas

Finalmente la interfaz de estadísticas nos proporciona acceso a los reportes de las estadísticas por tipo de curso, tipo de alumnos inscritos a los cursos y tipo de costos por cada tipo de curso.

Los reportes que se obtienen son los siguientes:

Fig. 4.16 Reporte por tipo de grupo

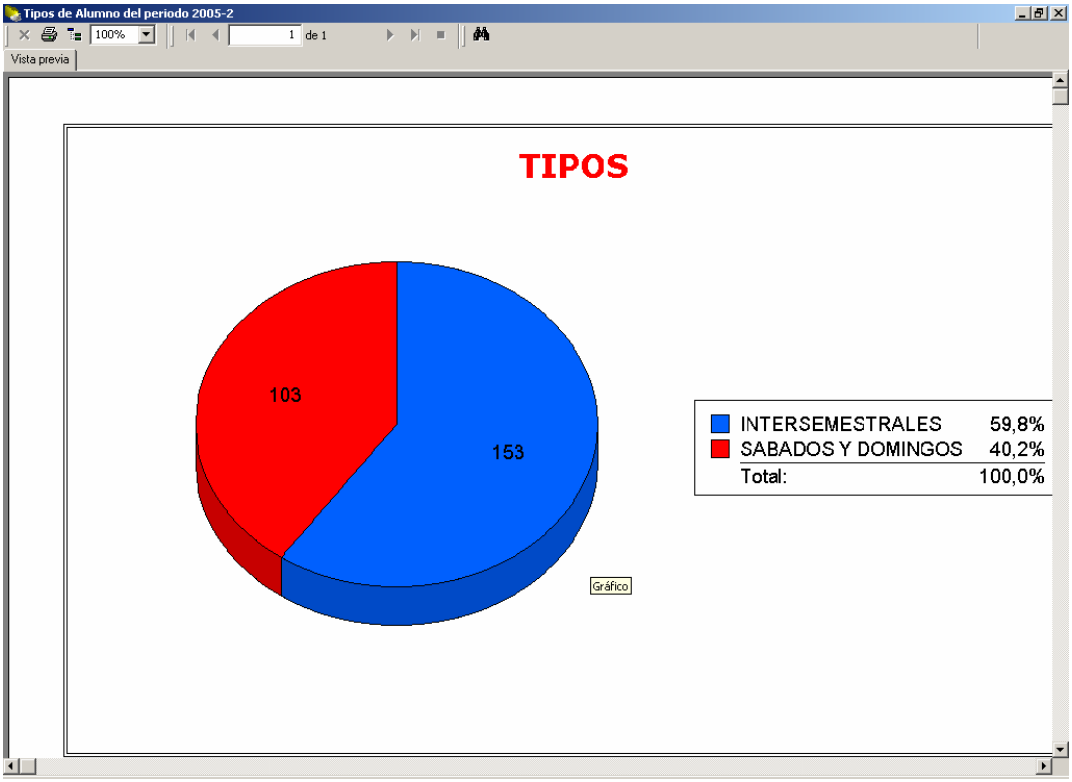


Fig. 4.17 Reporte por tipo de alumno

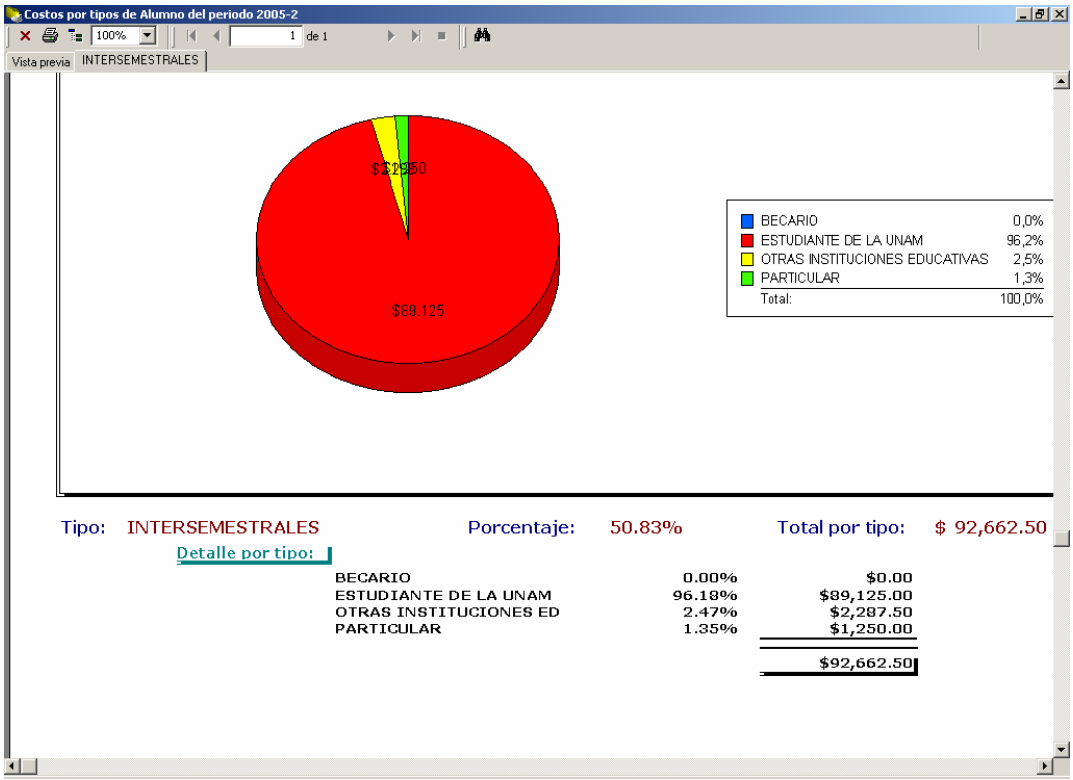


Fig. 4.18 Reporte por tipo de costo

En lo que se refiere al sistema de Aprobación de Horario se tienen las siguientes interfaces:

Fig. 4.19 Interfaz de Validación

Como en la caso del sistema de administración se cuenta con una interfaz de validación, en este caso cada jefe de departamento tendrá un usuario y contraseña diferente para visualizar solo los datos del personal a su cargo.

Después de que se ha validado al usuario y este es válido para entrar al sistema se muestra la pantalla de bienvenida:

Fig. 4.20 Interfaz de Bienvenida

A continuación se carga la interfaz principal del sistema:

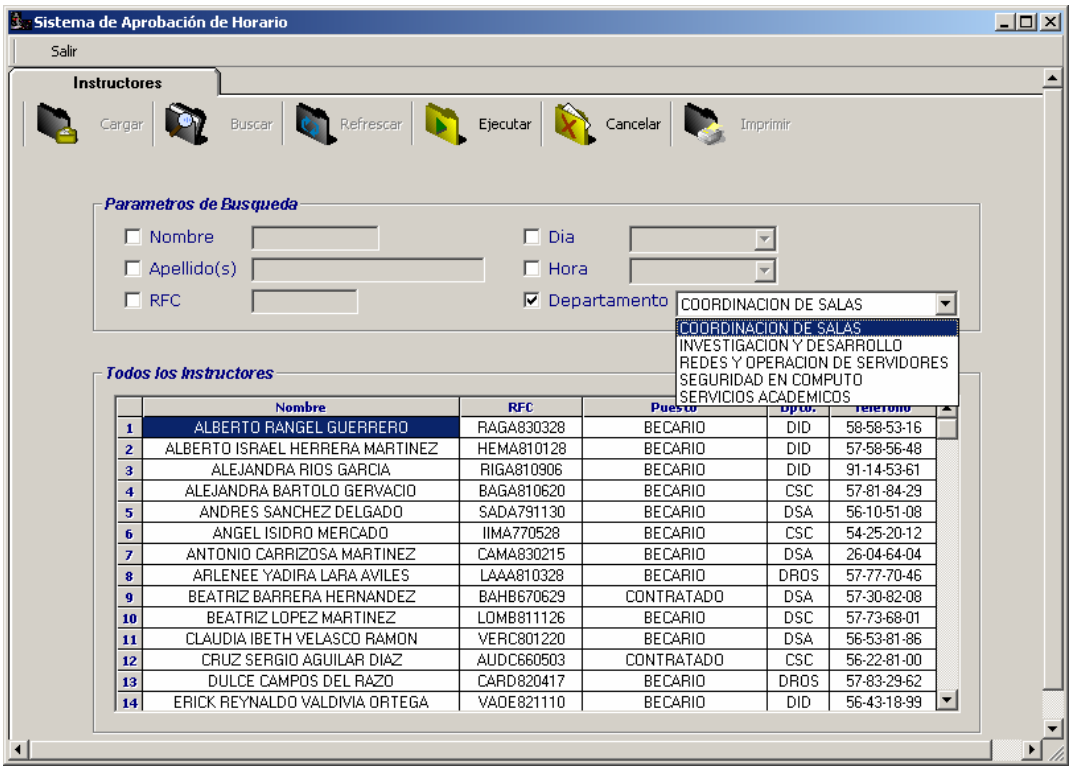


Fig. 4.21 Interfaz Principal del Sistema de Aprobación de Horario

Como puede observase solo cuenta con la interfaz de instructores, que tiene el mismo funcionamiento que la interfaz de instructores del sistema administrativo. En este caso cada jefe puede imprimir una lista general del todo el personal que labora en UNICA o solo del personal a su cargo con sus principales datos.

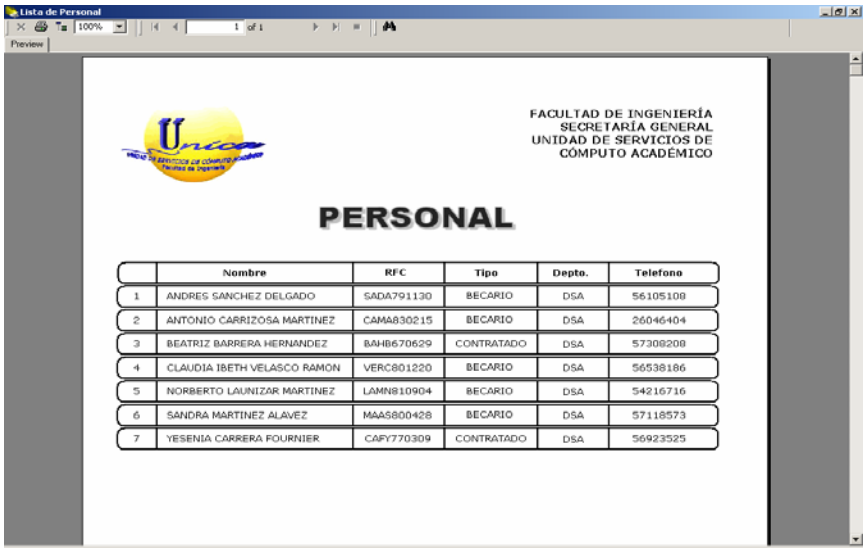


Fig. 4.22 Lista general del Personal por departamento

La diferencia principal se presenta en la interfaz detallada.

Ya que la barra de herramientas cambia ofreciendo la posibilidad de aprobar o no el horario del personal e imprimir el reporte con sus datos personales y horario de trabajo.

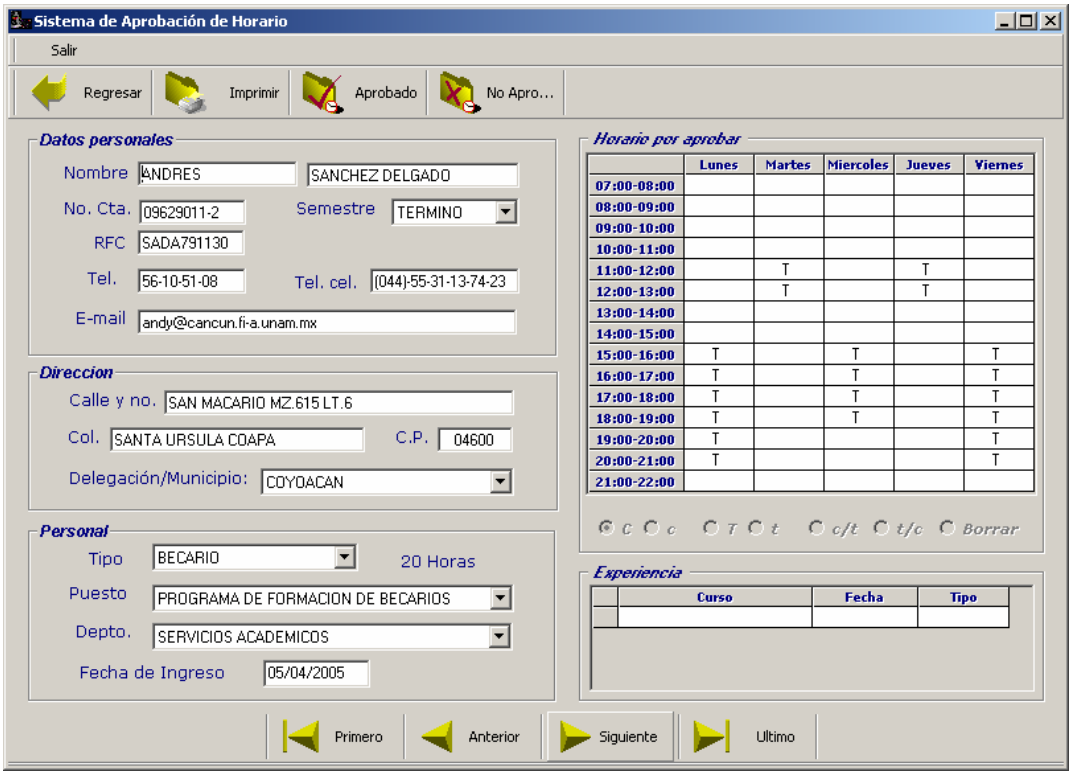


Fig. 4.23 Interfaz detallada del Personal

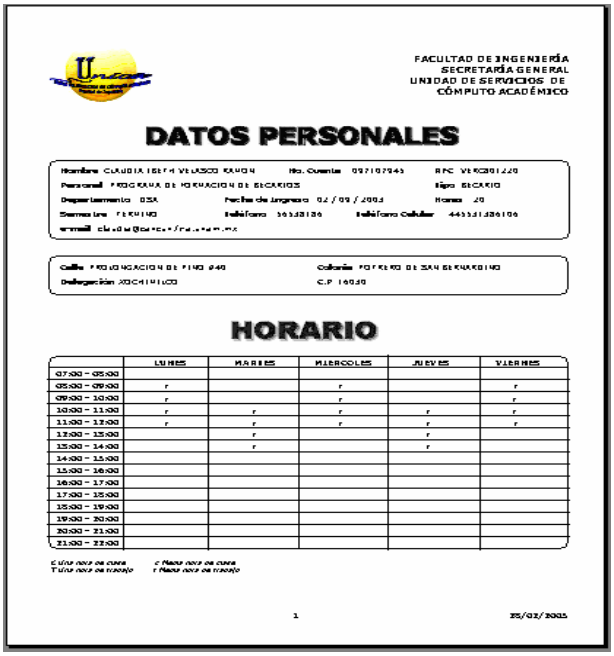


Fig. 4.24 Reporte detallado por persona

4.4.2. Interfaces de WEB

Para el caso del sistema en Web se tienen las siguientes interfaces:

La primera que se muestra es la interfaz de validación, en este caso para que cada miembro del personal de UNICA pueda tener acceso debe proporcionar su nombre(s) y su RFC.

Fig. 4.25 Interfaz de validación

Una vez que se ha confirmado que el usuario es válido se muestra la interfaz de bienvenida:

Fig. 4.26 Interfaz de bienvenida

Después de dar clic a siguiente se muestra la interfaz principal del sistema, en la se tienen tres opciones.

Fig. 4.27 Interfaz de opciones

En el apartado de Datos Personales, se tiene la posibilidad de cambiar los datos personales o bien modificar el horario de trabajo en UNICA, proceso que se realiza cada inicio de semestre.

Fig. 4.28 Interfaz de Datos Personales

En la parte de propuesta, el personal puede hacer las propuestas de cursos para cada periodo de impartición de los mismos, la interfaz es la siguiente:

Fig. 4.29 Interfaz de Propuesta

Finalmente en el apartado de calificaciones, cada instructor puede asentar las calificaciones finales de los cursos impartidos durante el periodo o bien modificarlas en el caso de alguna aclaración.

Fig. 4.30 Interfaz de Calificaciones

4.5. Integración con el SICC (Sistema de Inscripción y Control de Cursos)

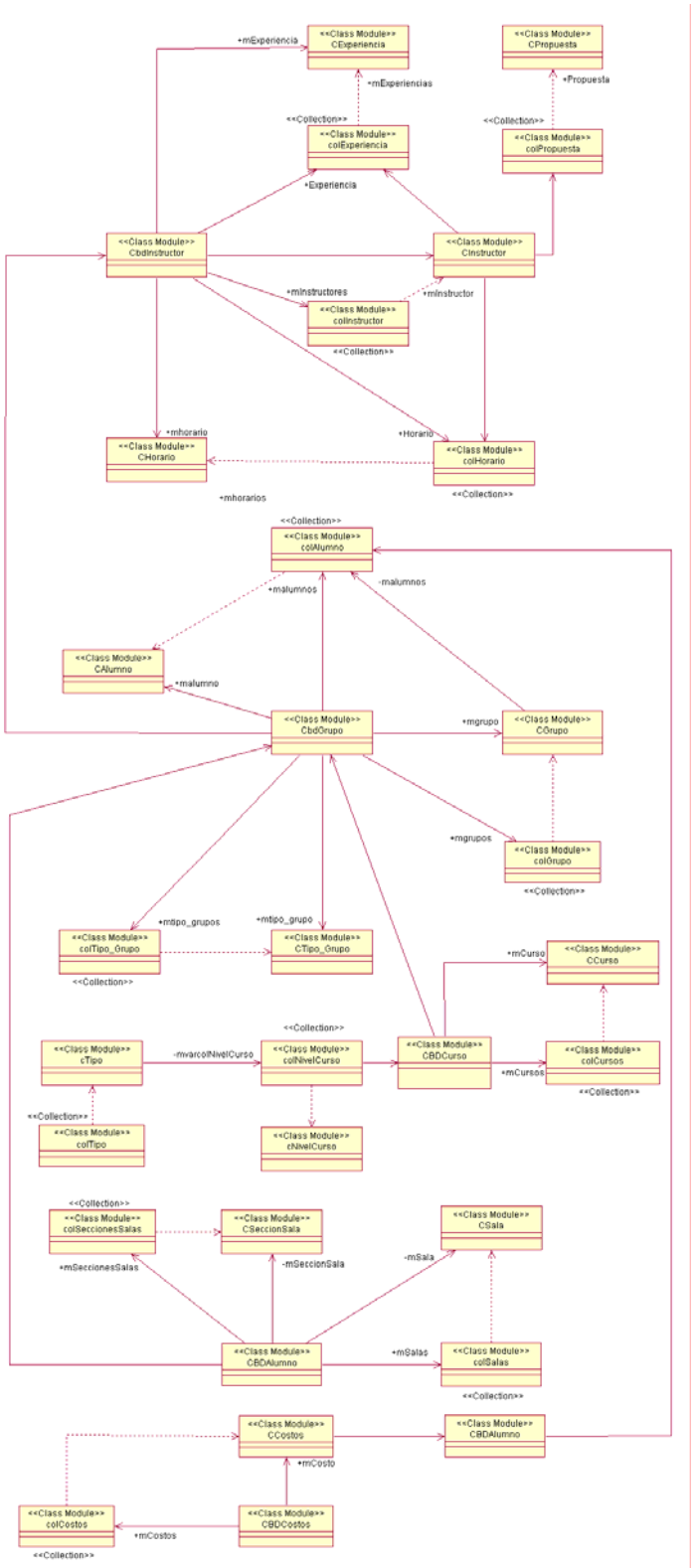


Fig. 4.31 Diagrama de Clases General

El diagrama muestra el esquema general en el que se integran los sistemas SICC y SAPIUN mostrando las relaciones entre las clases que pertenecen a cada sistema.

La integración del SICC con el SAPIUN se lleva a cabo con la interfaz de Grupos que es propia del sistema SICC y que gracias a que estos dos sistemas están diseñados bajo la misma filosofía es posible que esta interfaz forme parte del sistema administrativo del SAPIUN como un módulo más.

Puesta en Marcha

Capítulo V

PUESTA EN MARCHA DEL SAPIUN

5.1. Garantía de calidad del software

Cuando se desarrolla software, una de las actividades asociadas a este proceso es la prueba pues es una actividad fundamental dentro de cada una de las etapas del proceso de desarrollo de software. La prueba es indispensable, puesto que a partir de ella se puede determinar la calidad de los productos implementados, es decir permite al desarrollador determinar si el producto generado satisface las especificaciones establecidas; así mismo, permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución. A pesar de esto, en numerosas ocasiones su importancia se ha subestimado y hasta ignorado.

De una manera más formal de acuerdo a la IEEE (IEEE90) * el concepto de prueba (*testing*) se define como: Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente.

Cuando se habla de condiciones especificas se puede suponer la presencia de una especie de ambiente de operación de la prueba, para el cual deben existir determinados valores para las entradas y las salidas, así como también ciertas condiciones que delimitan a dicho ambiente de operación. Formalmente esto es conocido como *caso de prueba*.

La IEEE [IEEE90] define un caso de prueba como: Un conjunto de entradas, condiciones de ejecución y resultados esperados diseñados para un objetivo particular. A partir de las definiciones anteriores, en un proceso de prueba de software, se pueden identificar las siguientes acciones:

- ✓ Preparar una serie de casos de prueba.
- ✓ Llevar a cabo dichos casos de prueba.
- ✓ decidir cuando suspender la prueba.
- ✓ Evaluar los resultados generados por la prueba.
- ✓ Emitir un criterio de evaluación.

Desde hace ya algunos años, han surgido y evolucionado una variedad de métodos para realizar pruebas de software. Las alternativas más significativas en este contexto son las pruebas de caja blanca y las pruebas de caja negra; las primeras, pruebas orientadas a la estructura y las segundas al comportamiento del software.

* IEEE corresponde a las siglas del Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos, una asociación estadounidense dedicada a la estandarización en electrónica e informática.

5.2. Pruebas de Caja blanca y Caja negra

5.2.1. Pruebas de caja blanca

Las pruebas de caja blanca enfocan su atención a los detalles procedimentales del software, por ello la implementación de estas pruebas depende fuertemente de la disponibilidad de código fuente. Este tipo de pruebas, permiten generar casos para ejercitar y validar los caminos de cada módulo, las condiciones lógicas, los bucles y sus límites, así como también para las estructuras de datos, en otras palabras, aseguran que la operación interna del programa se ajusta a las especificaciones y que todos los componentes internos se han probado adecuadamente intentando garantizar que todos los caminos de ejecución del programa quedan probados. Las pruebas de caja blanca también son conocidas como pruebas de caja de cristal o pruebas estructurales.

Algunas de las pruebas más significativas dentro de este enfoque son:

- ✓ *Prueba de caminos*: En este tipo de prueba se realiza un análisis sobre una representación gráfica de un programa denominada *grafo de control*. En este grafo, los nodos representan bloques de instrucciones de un programa y los flujos de ejecución para dichas instrucciones se representan por medio de aristas. A partir de este grafo, se puede identificar un conjunto básico de caminos de ejecución, sobre el cual se pueden realizar pruebas con el propósito de ejercitar el flujo de ejecución de los caminos en una unidad.
- ✓ *Prueba de condiciones*: Basándose de igual forma en un grafo de control, pueden generarse casos de prueba para elementos individuales de expresiones lógicas. De esta forma se pretende probar cada condición con todas sus posibles alternativas.
- ✓ *Prueba de ciclos*: A partir del grafo de control, pueden generarse casos de prueba para las iteraciones definidas en los programas con el propósito de verificar si se realizan de forma correcta.
- ✓ *Prueba de definición de datos*: Estas pruebas son realizadas con el objetivo de encontrar posibles contradicciones o redundancias en la definición de los datos utilizados en el software. Para ello se realiza un análisis del comportamiento de cada uno de los datos o cada una de los flujos de ejecución.

5.2.2. Pruebas de Caja Negra

Este tipo de pruebas son conocidas también como pruebas funcionales o pruebas de comportamiento, concentran la atención en generar casos de prueba que permitan ejercitar los requisitos funcionales del software. A diferencia de las pruebas de caja blanca, que se basan en la lógica interna del software, este tipo de pruebas se concentran en su funcionalidad, por lo que mucho del trabajo se realiza interactuando con la interfaz del software, aunque no son una alternativa a las pruebas de caja blanca sino que complementan a éstas. Los casos de prueba generados en este enfoque, se diseñan a partir de valores entrada y salida. De esta forma, se puede determinar la validez de una salida para un conjunto de entradas proporcionadas. La aplicación de pruebas de caja negra permite detectar errores como funciones incorrectas o ausentes, errores en estructuras de datos, errores de rendimiento, así como errores de inicialización y terminación.

Estas son algunas de las pruebas más conocidas en este contexto:

- ✓ *Partición equivalente*: La idea de esta técnica, es en dividir los valores válidos y no válidos para entradas y salidas en un número reducido de particiones de forma que, el comportamiento del software sea el mismo para cualquier valor contenido en una partición particular. El propósito principal de una partición es reducir la cantidad de casos de prueba generados en el proceso.
- ✓ *Análisis de los valores límite*: La generación de casos de prueba en esta técnica, se enfoca en los valores limites, esto bajo la consideración de que existe una tendencia a fallar precisamente cuando el software trabaja con valores extremos de la variable de entrada. Generalmente los valores establecidos para generar los casos de prueba son el mínimo, valores un poco arriba del mínimo, valor máximo y valores un poco arriba del máximo.
- ✓ *Pruebas según la experiencia (error guessing):* En este tipo de prueba la generación de casos se realiza a partir de la intuición y la experiencia. La idea básica es redactar una lista de las posibles fallas o de las posibles situaciones en las cuales suele ocurrir algún problema y así desarrollar casos de prueba basados en la información contenida en estas listas.
- ✓ *Tablas de decisión*: Este tipo de prueba permite describir el comportamiento de un programa a partir de un conjunto de acciones que este realiza cuando se opera bajo determinadas condiciones. En este enfoque, las condiciones pueden ser interpretadas como entradas de un programa y las acciones como las salidas producidas. Para ello se pueden utilizar conectores lógicos y (and), o (or) y no (not). Al involucrar aspectos de lógica, se dice que este tipo de prueba se hace más rigurosa, que permite además transformar una especificación en lenguaje natural en una especificación más formal.

5.3. Análisis de las pruebas

5.3.1. Niveles de prueba del software

Un nivel de prueba permite especificar el alcance de la prueba de software que se realiza, se pueden identificar principalmente dos niveles:

Bajo nivel

En este nivel están consideradas todas aquellas pruebas que se realizan a componentes individuales de un programa. Las pruebas que se pueden realizar en este nivel son:

- ✓ *Pruebas de unidad*: Como su nombre lo indica, este tipo de prueba se aplica a elementos de software individualmente, excluyendo todos aquellos casos en los que se considere la interacción con otras unidades. El propósito fundamental de una prueba de unidad es descubrir diferencias entre la especificación del modulo de la interfaz y el comportamiento efectivo.
- ✓ *Pruebas de integración*: Las pruebas de integración se realizan con el propósito de ejercitar la arquitectura de un sistema. Una vez que ya se han probado que las unidades funcionan de forma correcta de forma aislada, se procede a probar cómo funcionan al integrarlas con otras de forma que se llegue a probar el comportamiento de un sistema final. La forma en que se pude organizar la integración de las unidades para la prueba puede realizarse siguiendo enfoques como el ascendente, descendente, big bang, o sandwich.

Alto nivel

En éste nivel las pruebas se orientan a un producto completo; para ello se proponen las siguientes alternativas:

- ✓ *Pruebas de sistema*: Este tipo de pruebas permiten probar el sistema como un todo así como también aspectos relacionados con la integración del producto a otros sistemas.
- ✓ *Pruebas de usabilidad*: El diseño de pruebas de usabilidad, requiere considerar a los usuarios que trabajan con el producto, con el propósito de observar sus respuestas hacia éste. De ésta forma se pueden observar discrepancias existentes entre las interfaces implementadas y los requerimientos de los estilos de trabajo de los usuarios finales.
- ✓ *Pruebas de función*: Este tipo de pruebas tiene como objetivo detectar inconsistencias entre la especificación funcional de un programa y su comportamiento actual.
- ✓ *Pruebas de aceptación*: Realizando estas pruebas se puede comparar el producto final con las necesidades finales de los usuarios.

- ✓ *Pruebas de regresión*: Las pruebas de regresión son recomendables cuando partes del software se modifican, ya que permiten verificar que los cambios realizados no generan comportamientos no deseados.

5.3.2. Pruebas realizadas en el SAPIUN

Para el sistema SAPIUN se realizaron pruebas de caja de blanca y caja negra. Se comenzará por mencionar las pruebas de caja blanca realizadas:

La primera prueba realizada fue la prueba de *definición de datos*, misma que consistió en establecer criterios para la asignación de nombres a variables y funciones, es decir, que estuvieran contenidas dentro del diccionario de datos final, por tal motivo fue necesario realizar una revisión detallada de cada una de las variables y funciones utilizadas. Ésta revisión nos permitió detectar el problema de duplicidad de variables y funciones que afectaban el proceso de integración, problema que fue ocasionado por el trabajo modular realizado pero que fue resuelto con la redefinición de cada una de las variables y funciones duplicadas.

La segunda prueba realizada fue la *prueba de condiciones*, que se llevo a cabo en conjunción con los casos de usos del sistema y cuya principal aplicación fue en el proceso de integración de los módulos del SAPIUN y el módulo de grupos del SICC. En esta etapa se dio seguimiento a cada uno de los casos de uso que conforman al sistema y su interrelación. Dicha prueba nos permitió detectar las carencias que se tenían al momento de realizar el proceso de integración por lo que se crearon nuevos procesos que permitieran la interacción entre los módulos, dicha afectación primordialmente se realizo en la interfaz administrativa del SAPIUN, ya que fue aquí donde se tuvo que programar el proceso (herramientas SAPIUN) que invoca a cada uno de los módulos.

Una tercera prueba realizada fue la *prueba de ciclos*, en esta prueba fue necesario hacer uso de los diagramas de estado y funcional del módulo de instructores ya que su principal aplicación fue con la validación del horario de trabajo, que como se recordará se debe cumplir un horario de acuerdo al tipo de personal dado. Esta prueba permitió detectar los problemas que se tenían en la validación de horarios vía Web e interfaz de instructores, dicho problema se mostraba cada vez que se realizaba un alta de horario vía Web y que no correspondía con lo que se capturaba en la interfaz de instructores, lo anterior nos permitió unificar el número de iteraciones necesarias para validar el horario de trabajo.

La cuarta prueba realizada fue la *prueba de caminos*, en esta etapa fue necesario hacer uso de los diagramas de estado y funcionales del sistema, pues de ésta manera fue posible detectar deficiencias en los caminos que debería seguir el sistema al momento de interactuar con sus diversos módulos, y principalmente con el SICC, el principal problema se detecto al momento de realizar la integración con el SICC ya que su diseño sufrió cambios al momento de su desarrollo por lo que fue necesario rediseñar el esquema de integración, ya que él sistema ya estaba puesto en producción y había que regirse en su funcionamiento primordial. Esta prueba contribuyo para detectar los caminos que se involucraban para realizar el módulo de estadísticas solicitado al final del desarrollo.

Entre las pruebas de caja negra realizadas se tienen:

La primera prueba realizada fue la de *partición equivalente* que nos permitió conocer las deficiencias que pudiera tener el sistema al estar puesto en producción, para tal efecto se realizaron pruebas con todos los valores válidos y no válidos que acepta el sistema, la prueba se realizó durante todo el desarrollo del sistema dando como resultado una mayor estabilidad y sobre todo que se generaran un menor número de errores por el mal manejo del sistema e integrar mensajes informativos que describieran las posibles fallas en su funcionamiento. No obstante se dieron lugar a los mensajes de error que se originaron por fallas externas al sistema, un ejemplo muy claro es el de conexión a base de datos, ya que si no se tiene disponible el servidor de base de datos el sistema deberá indicarnos la falta de comunicación, dando al usuario una mayor descripción del problema ocasionado en el sistema y no precisamente por una falla propia del sistema.

La segunda prueba realizada durante el desarrollo del sistema fue la *prueba de función*, que se apoyo principalmente en los casos de uso y diagramas de estado. Dicha prueba nos permitió crear una interfaz mucho más amigable y apegada al diseño independientemente de que se debían conservar las funciones básicas que realizan los sistemas bajo el sistema operativo Windows. Finalmente no se tuvo la necesidad de hacer grandes pruebas al final del desarrollo del sistema ya que éste fue probado durante su creación.

La tercera prueba aplicada al sistema fue la *prueba de usabilidad*, esta se apoyó principalmente en los diagramas de estado y casos de uso para llevar acabo dicha prueba fue necesario poner a disposición del usuario una primera versión, misma que probaría durante un tiempo determinado y al final enviaría sus comentarios para el afinamiento de la interfaz o bien detectar las posibles deficiencias en su funcionamiento y valores devueltos. Al finalizar la etapa y junto con las pruebas mencionadas anteriormente, sólo se afinó detalles de la interfaz como: rótulos y la impresión de reportes, al ordenar la información en formatos más especializados y que se entregaron al final.

En conjunción con la prueba anterior, se genero la *prueba de aceptación*, misma que en su mayor parte resultó en gran aceptación por parte del usuario, ya que en el objetivo inicial se obtuvo un mayor beneficio de lo esperado por el usuario, entre lo que podemos destacar: la reducción del tiempo en el proceso de inscripción, tener información con una mayor disponibilidad del personal adscrito, reportes en tiempo real de lo que esta ocurriendo con el proceso de inscripciones y sobre todo que no requiere de una atención personalizada por parte del DSA.

Finalmente podemos mencionar que todas y cada una de las pruebas apoyaron en gran medida al desarrollo del producto final, como lo demuestra el producto entregado a UNICA.

Ciclo de Vida

Capítulo VI

CICLO DE VIDA DEL SAPIUN

6.1. Ciclo de vida

Como se ha planteado en capítulos anteriores, el desarrollo de software va unido a un ciclo de vida compuesto por una serie de etapas que comprenden todas las actividades, desde el momento en que surge la idea de crear un nuevo productos de software, hasta que el producto deja definitivamente de ser utilizado por el último de sus usuarios.

El ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo, por lo que deberá estar cubierto por una metodología que indicará cómo hay que obtener los distintos productos parciales y finales. Una metodología que cubre todos los aspectos del ciclo de vida del desarrollo del sistema ofrece una solución atractiva ya que la metodología por sí misma asegura algo completo y consistente, que en el caso del SAPIUN es OMT.

De manera general las fases del ciclo de vida se definen de la siguiente manera:

- ✓ *Análisis:* Describe las características observables del sistema, como funcionalidad, funcionamiento y capacidad. Esta descripción incluye normalmente los modelos que representan la construcción lógica, y su colocación dentro de un ambiente de sistema.
- ✓ *Diseño*: Prepara definiciones en cuanto a cómo el sistema logrará sus requerimientos. Los modelos preparados en análisis se refinan, o se transforman, en los modelos del diseño que representan la naturaleza física del sistema.
- ✓ *Implementación:* Convierte los modelos desarrollados del diseño en el software ejecutable dentro del ambiente del sistema. Este implica la codificación del programa.
- ✓ *Prueba*: Se centra en asegurar que cada una de las entregas cumpla con las necesidades indentificadas por el/los usuarios. Las pruebas se complementan con las pruebas de integración de los diversos programas y la prueba del sistema, que incluye la aprobación del conjunto del sistema.
- ✓ *Validación*: Se evalúan después las características o las cualidades del sistema. Las características definen la secuencia de pasos, de entradas requeridas y de salidas, papeles implicados, así como la interacción con otros pasos.

Una vez delimitadas en cierta manera las etapas, habrá que ver la forma en que éstas se afrontan. Existen diversos modelos de vida, y la elección de un cierto modelo para un determinado proyecto puede ser de vital importancia; el orden de las etapas es un factor importante, por ejemplo tener una etapa de validación al final del proyecto, tal como lo sugieren los diversos modelos; hay que tener en cuenta que retomar etapas previas es costoso, y cuanto

más tarde se haga más costoso resultará, por tanto el hecho de contar con una etapa de validación tardía tiene su riesgo y, por su situación en le ciclo un posible tiempo de reacción mínimo en caso de tener que retornar a fases previas. Seguidamente se produce el periodo de explotación del sistema y los diversos cambios que experimente por efecto del mantenimiento, que deberá corregir errores, introducir mejoras y adaptar el sistema a nuevas necesidades que puedan surgir con el tiempo.

En lo que se refiere al SAPIUN, se proyecta un ciclo de vida que dependerá de la evolución de la tecnología con la que fue desarrollado, debido a que hasta el momento cubre las necesidades del usuario final (DSA), pues se trata de actividades que se han desarrollado desde la creación del departamento y que no han cambiado en su esencia y solo han requerido de mejoras en el manejo de la información y la presentación de la misma por medio de reportes. Aunque si cambiara de manera radical el funcionamiento del departamento se tendría que evaluar la posibilidad del desarrollo de un nuevo sistema que cubra las nuevos requerimientos.

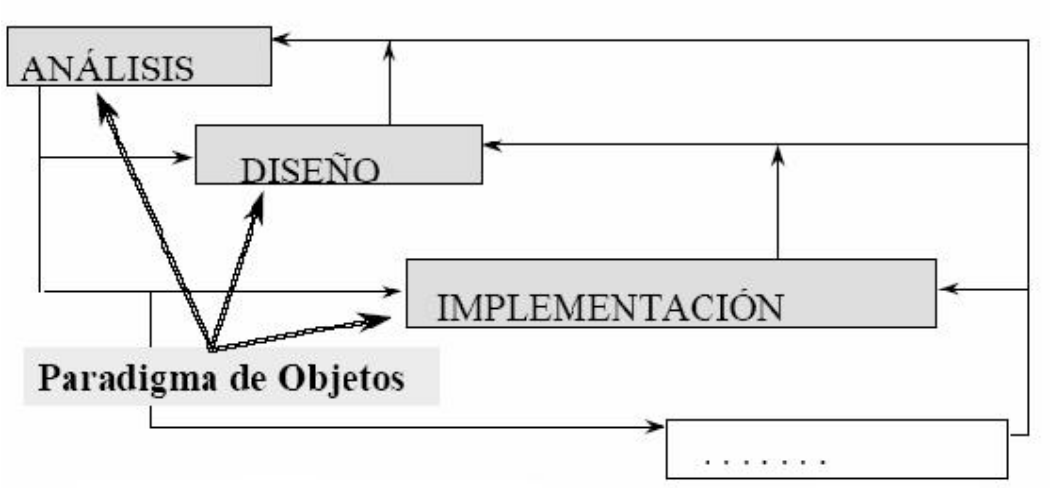


Figura 6.1 Ciclo de vida del SAPIUN

6.2. Mantenimiento

Cuando el sistema ya se encuentra en fase de producción (en funcionamiento para el usuario, cumpliendo los objetivos para los que ha sido creado); a partir de este momento se entra en la etapa de mantenimiento, que supondrá pequeñas operaciones tanto de corrección como de mejora de la aplicación, rendimiento y desempeño; debidas a la evolución del entorno y a las modificaciones producidas por los cambios de requisitos del usuario dirigidos a reforzar o ampliar el sistema. La fase de mantenimiento vuelve a aplicar las fases de definición y de desarrollo, pero en el contexto del sistema ya existente.

De acuerdo con lo anterior se pueden establecer los siguientes tipos de mantenimiento:

- ✓ *Correctivo*: Incluso llevando a cabo las mejores actividades de garantía de calidad, es muy probable que el cliente descubra defectos en el software. El mantenimiento correctivo cambia el software para corregir los defectos.
- ✓ *Evolutivo*: Conforme utilice el software, el usuario puede descubrir funciones adicionales que podrían interesar que estuvieran incorporadas en el software; en otras palabras, son las incorporaciones, modificaciones y eliminaciones necesarias para cubrir la expansión o cambio en las necesidades del usuario.
- ✓ *Adaptativo*: Con el paso del tiempo es probable que cambie el entorno original (cambios de configuración del hardware, software de base, gestores de base de datos, comunicaciones, etc.) para los que se desarrollo el software. Por lo tanto, el mantenimiento adaptativo consiste en modificar el software para acomodarlo a los cambios de su entorno externo.
- ✓ *Perfectivo*: Son las acciones llevadas a cabo para mejorar la calidad interna de los sistemas en cualquiera de sus aspectos: reestructuración del código, definición más clara del sistema y optimización del rendimiento y eficiencia. El mantenimiento perfectivo amplía el software más allá de sus requisitos funcionales originales.

Para el SAPIUN se han utilizado los siguientes tipos:

- ✓ *Correctivo*: Una vez liberada una versión inicial del SAPIUN, fue necesario realizar ajustes en los módulos de Costos y Salas. Para el módulo de Costos se modificó el esquema de asignación de cuotas para los cursos, ya que el DSA ofrecía un costo diferente según el tipo de alumno que se inscribiera y que variaba por cada tipo de curso que se impartía a lo largo del semestre, requerimiento que no fue proporcionado al inicio y que llevó a replantear el diseño y generar un nuevo esquema, lo cual implico invertir un tiempo mayor en su corrección. En el caso de Salas se incluyó un catálogo de ubicaciones y el módulo que lo administrara, pues en el esquema original no se tenía

contemplada la creación de nuevas salas en UNICA, por lo que se requirió integrar las nuevas interfaces al módulo.

- ✓ *Evolutivo:* En este caso, se agregó el módulo de Estadísticas ya que una vez concluido el calendario de cursos el DSA requirió de reportes que le indicaran la cantidad de cursos impartidos y cancelados, el ingreso total por periodo y el detalle por el tipo de alumno inscrito a cada curso, con el fin de estudiar y atender la demanda de las personas inscritas, así como aumentar el ingreso por curso; esto fue posible gracias a que el SAPIUN cuenta con toda la información necesaria para la creación de dicho módulo.
- ✓ *Perfectivo:* Este tipo de mantenimiento se fue llevado a cabo durante todo el proceso de desarrollo del SAPIUN principalmente en la etapa de pruebas del sistema, permitiendo mejorar de manera continua el producto final.

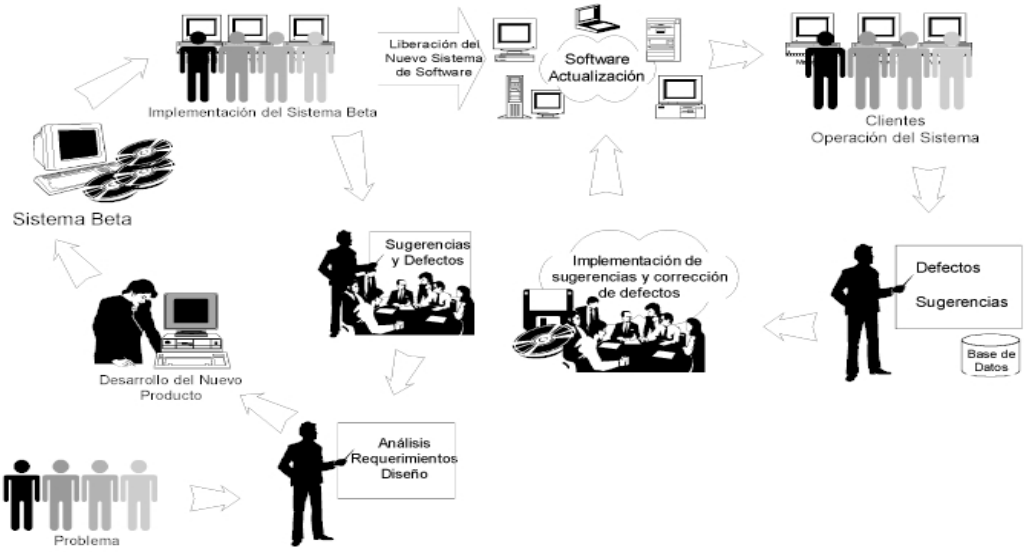


Figura 6.2 Modelo de mantenimiento del SAPIUN

6.3. Perspectivas a futuro

Como parte del plan de desarrollo integral de UNICA, se pretende que el SAPIUN sea el sistema general que procese la información necesaria para agilizar las actividades administrativas que se llevan a cabo. Desde su inicio se han tomado diversas acciones, que permiten agregar los diversos módulos que se desarrollarán en un futuro próximo gracias al modelo de OMT bajo el cual fue diseñado, además de que permite realizar las mejoras sin causar un impacto de gran escala al momento de realizar la inserción de otros módulos, como se mostró en la integración del sistema SICC.

Como parte de este desarrollo el DID se encuentra trabajando de manera conjunta con el DSA, para tomar los requerimientos administrativos y realizar el modelado correcto con base en el diseño propuesto, sin dejar de prestar atención a los cambios que el sistema requiere como parte de su mantenimiento y evolución. De esta forma el SAPIUN brinda a las nuevas generaciones de becarios la posibilidad de ampliar sus conocimientos.

6.4. Contribución

El SAPIUN ha transformado radicalmente la forma en la que se lleva a cabo la administración general por parte del DSA, ya que ha reducido en gran medida el tiempo utilizado para la generación de calendarios de cursos, reportes, y control del personal que labora en UNICA, así como de los recursos materiales de los que dispone. Esto trajo como beneficio una mejora en la calidad de los cursos, pues se contemplan todas y cada una de las características que se deben de cubrir de manera inmediata. Además de permitir a los becarios programar sus cursos tomando como base las estadísticas proporcionadas al final de cada periodo.

En la parte de análisis, diseño y programación, el SAPIUN, deja una nueva metodología de desarrollo, como lo hemos visto a lo largo del presente trabajo, sirviendo primordialmente como guía para la creación de nuevos sistemas. De esta manera se ha convertido en la columna vertebral de administración de UNICA, siendo el principal proveedor de información para los sistemas que actualmente se encuentra en desarrollo, entre los que podemos mencionar el SECC (Sistema de Evaluación y Control de Cursos), el SICEB (Sistema de Calificación y Evaluación de Becarios) y CORA (Control de Reportes y Asesorías), que en su diseño forman parte del esquema propuesto para el SAPIUN.

Conclusiones

Capítulo VII

CONCLUSIONES

En este trabajo de Tesis presenté el sistema de información SAPIUN que se basa en la metodología de desarrollo OMT y el lenguaje de modelado UML, para este sistema se realizó un análisis, diseño y desarrollo de acuerdo a los principios que propone la metodología y auxiliados por UML y herramientas CASE de cuarta generación. La resolución del problema consistió en desarrollar un sistema que permitiera tener una administración completa de la información que se maneja en UNICA. Para tal efecto fue necesario realizar un análisis exhaustivo del proceso administrativo que se lleva a cabo en el DSA definiendo cada uno de los requerimientos y la forma de procesar la información; con la finalidad de agilizar el proceso y ganar tiempo en el desarrollo de otras actividades.

Para realizar este desarrollo se realizaron 5 módulos y se integró el sistema del SICC:

- ✓ Modulo 1. Información del Personal. Se realizaron pantallas de captura para el administrador que permitieran visualizar los datos personales de la gente adscrita por departamento así como del personal en general. Una pantalla de captura para el personal, que permite la actualización de información personal y laboral de forma inmediata. Se generó un listado general del personal, por departamento e individual para el archivo.
- ✓ Modulo 2. Información de los Cursos. Se creó un catalogo completo de los cursos que se imparten con su respectivo temario, antecedentes y requerimientos de software y hardware. Con el mismo detalle se crearon los reportes de los mismos.
- ✓ Modulo 3. Información de Salas de Cómputo. Se realizó el control de salas que nos permite tener conocimiento del equipo con el que se cuenta y la paquetería instalada y ubicación de las mismas. De la misma forma se crearon reportes detallados con la información respectiva.
- ✓ Modulo 4. Información de Costos. Ahora se permite asignar los costos a los diversos cursos que se imparten de acuerdo al tipo de alumno y el período en el cual se aplican. De igual manera se realizó el reporte general y el detallado por los tipos de alumnos, esto con la finalidad de saber cuales son los descuentos que se le aplican al momento de realizar su inscripción.
- ✓ Modulo 5. Información de Estadísticas. Se realizó el módulo de estadísticas que permite realizar un estudio completo de los alumnos que se inscriben y a que cursos. Da a conocer en tiempo real los ingresos obtenidos en el actual período o en los anteriores.

- ✓ Integración del SICC. Todos estos módulos permitieron integrar al Sistema de Inscripciones y Control de Cursos, que para su funcionamiento requirió de realizar la administración de su materia prima que lo conforma, como lo son los módulos antes mencionados. Dando origen al módulo de estadísticas que nos permite realizar un estudio completo cada vez que abre un período de inscripción. Con la finalidad de mejorar y ofrecer servicios de calidad.

Al utilizar OMT como metodología se creó un sistema totalmente modular, que permite agregar nuevas funciones, en un futuro sin alterar el objetivo principal para el cual fue creado, caso que se ejemplificó al momento de realizar la integración con el SICC. Requerimiento que fue solicitado por la Ing. Barragán, ya que como parte del crecimiento de UNICA se integrarán en un futuro próximo, diversos sistemas que robustecerán y simplificarán el funcionamiento del Departa0mento de Servicios Académicos y que toman como guía esta nueva forma de trabajo planteada en el presente trabajo de tesis.

Una vez realizado el análisis del sistema bajo OMT, fue necesario hacer uso de un lenguaje que permitiera una interacción clara y sencilla entre el usuario final y el desarrollador, por lo que se eligió utilizar UML, pues al ser un lenguaje creado bajo los principios de OMT brindó la posibilidad de expresar cada una de las etapas del sistema de manera gráfica y de muy fácil entendimiento, permitiendo que el usuario y el desarrollador visualizarán todos los posibles procesos que intervienen en el funcionamiento del sistema. Lo que permitió incursionar en una nueva forma de trabajo sino que además se integró un solo grupo en el desarrollo del sistema, sin que el usuario quedara fuera del proceso, sino que al contrario fuese el supervisor del trabajo que se realizaba.

Para la implementación se consideraron diversos lenguajes de programación, pero después de un análisis previo de los recursos con los que dispone UNICA se opto por utilizar Visual Basic 6.0 como lenguaje de programación para SAPIUN. Visual Basic resultó ser un entorno de desarrollo bastante completo a pesar de no soportar en su totalidad la filosofía orientada a objetos, pues en los objetivos planteados al inicio del desarrollo se requerían de interfaces totalmente amigables como lo son las bajo el sistema operativo Windows, por lo que no se presentaron inconvenientes que en algún momento hicieran dudar acerca de la elección; sino que al contrario, fue una agradable experiencia encontrar que a pesar de sus limitaciones permitió trabajar con la metodología propuesta y lograr un producto totalmente robusto y funcional en todos sus aspectos.

A continuación se listan las conclusiones obtenidas a partir del desarrollo realizado:

- ✓ Se desarrolló un sistema capaz de controlar y administrar los procesos académico - administrativos que se realizan en UNICA.
- ✓ Mediante este sistema se administra el personal y se genera toda la información relacionada con los cursos a partir del SAPIUN, y que integrados con el SICC forma un solo ambiente de trabajo.
- ✓ Se redujo de manera significativa el tiempo destinado al proceso académico – administrativo, es decir, ahora el sistema es capaz de:
  - o Generar reportes individuales del personal que indican su horario de trabajo y sus datos personales
  - o Generar listas de cursos con su respectivo temario
  - o Características del equipo con el que cuentan cada una de salas
  - o Costos de cursos, listas de alumnos y diplomas
  - o Generar estadísticas que permiten realizar una mejor programación de los cursos que imparte UNICA.
- ✓ Se implementó una metodología de software para el desarrollo de sistemas, se profundizó en la investigación de nuevos procesos que involucra la ingeniería de software así como las herramientas de software necesarias para su implementación generando un grupo de expertos en el tema, lo que su vez se tradujo en nuevas propuestas de cursos, asesorías especializadas y apoyos en la realización de otros sistemas, por lo que hoy en día, se puede hablar de la formación de recursos humanos de calidad con conocimientos específicos en le tema expuesto.
- ✓ Finalmente puedo decir que el SAPIUN se ha convertido en el sistema de información de UNICA. Actualmente comparte la información académico – administrativa adquirida, a varios sistemas que fueron desarrollados por el DID. Dichos sistemas utilizan la misma base de datos que el SAPIUN para llevar a cabo sus propios procesos.

Manual de Usuario

SAPIUN

Sistema de Administración del Personal e Información de UNICA

SISTEMA DE ADMINISTRACIÓN

1.1. Entrada al sistema

Al ingresar al sistema se presenta interfaz de validación de usuario; la cual tiene por objetivo autenticar al usuario:

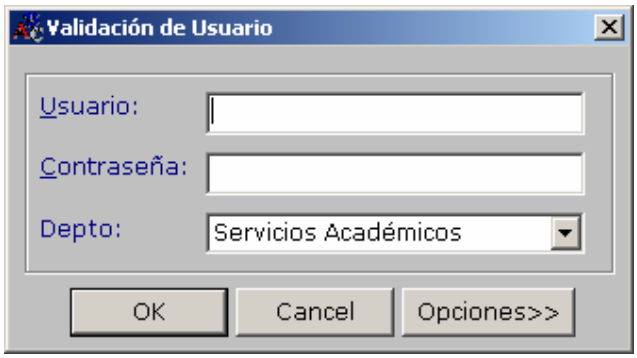


Fig. A.1 Interfaz de Validación

Adicionalmente ésta interfaz cuenta con un botón de opciones que permitirá cambiar el origen de datos, es decir el nombre del servidor, base de datos y puerto de comunicación.

Si se trata de un usuario válido se mostrará la interfaz de bienvenida al sistema:

Fig. A.2 Interfaz de Bienvenida

Después de unos segundos se mostrará la interfaz principal, ésta indica de manera general como se encuentra dividido el sistema:

- ✓ Instructores: Información del personal que labora en UNICA.
- ✓ Cursos: Información de los cursos a impartir en UNICA.
- ✓ Grupos: Información de los grupos abiertos para inscripción, parte del Sistema de Inscripción y control de Cursos, SICC.
- ✓ Costos: Permite administrar los costos de los cursos por periodo.

- ✓ Salas: Información de las salas de cómputo con las que cuenta UNICA.
- ✓ Estadísticas: Se presentan los reportes por periodo, de las estadísticas generales por tipo de curso, tipo de alumnos inscritos y tipo de costos.
- ✓ Salir: Salir del sistema.

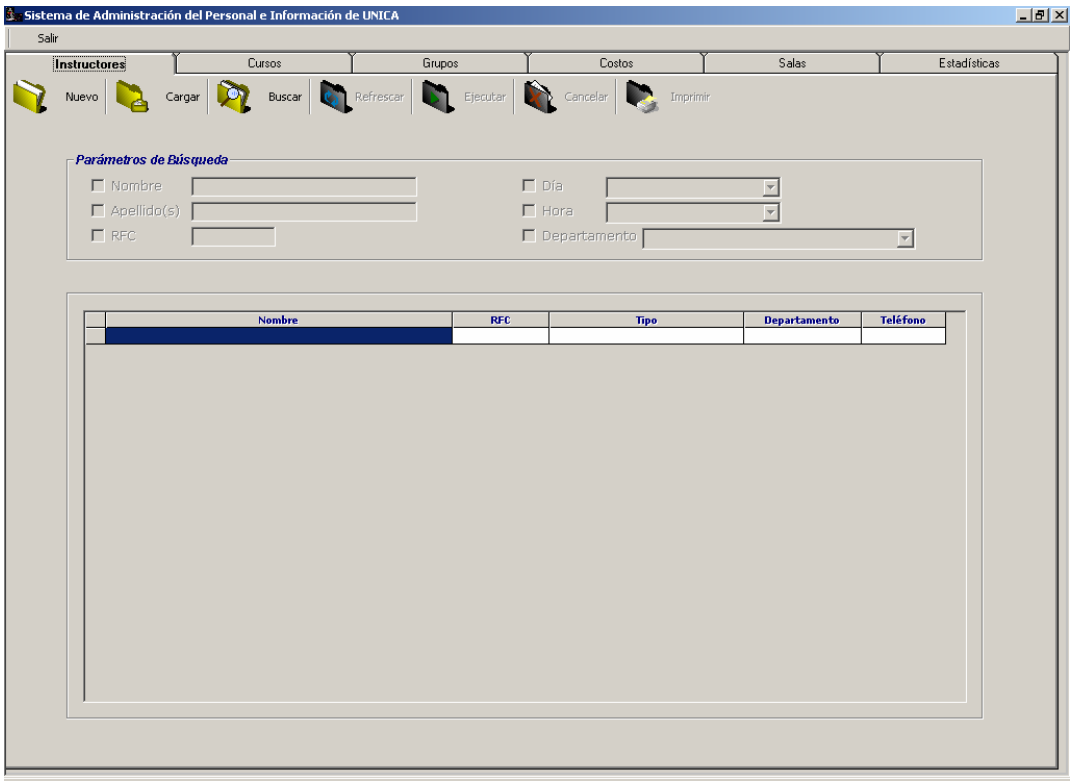


Fig. A.3 Interfaz Principal

Módulo de instructores

En esta interfaz se tienen las siguientes opciones para administrar la información del personal:

- ✓ *Nuevo*: Dar de alta un nuevo miembro del personal.
- ✓ *Cargar*: Obtener la información general del personal de UNICA (nombre, RFC, tipo, departamento y teléfono).
- ✓ *Buscar*: Búsqueda de los datos de un miembro del personal de acuerdo a los parámetros seleccionados; estos son nombre, apellidos, RFC, día, hora y departamento.
- ✓ *Ejecutar:* Realiza la búsqueda de acuerdo a los parámetros seleccionados (la opción se habilita al momento de seleccionar un parámetro para la búsqueda).
- ✓ *Refrescar*: Ejecuta nuevamente la búsqueda o el cargado general de datos.

- ✓ *Cancelar*: Cancela la búsqueda.
- ✓ *Imprimir*: Permite imprimir una lista del resultado devuelto en pantalla.

Al dar clic al botón *Nuevo* aparece la siguiente interfaz:

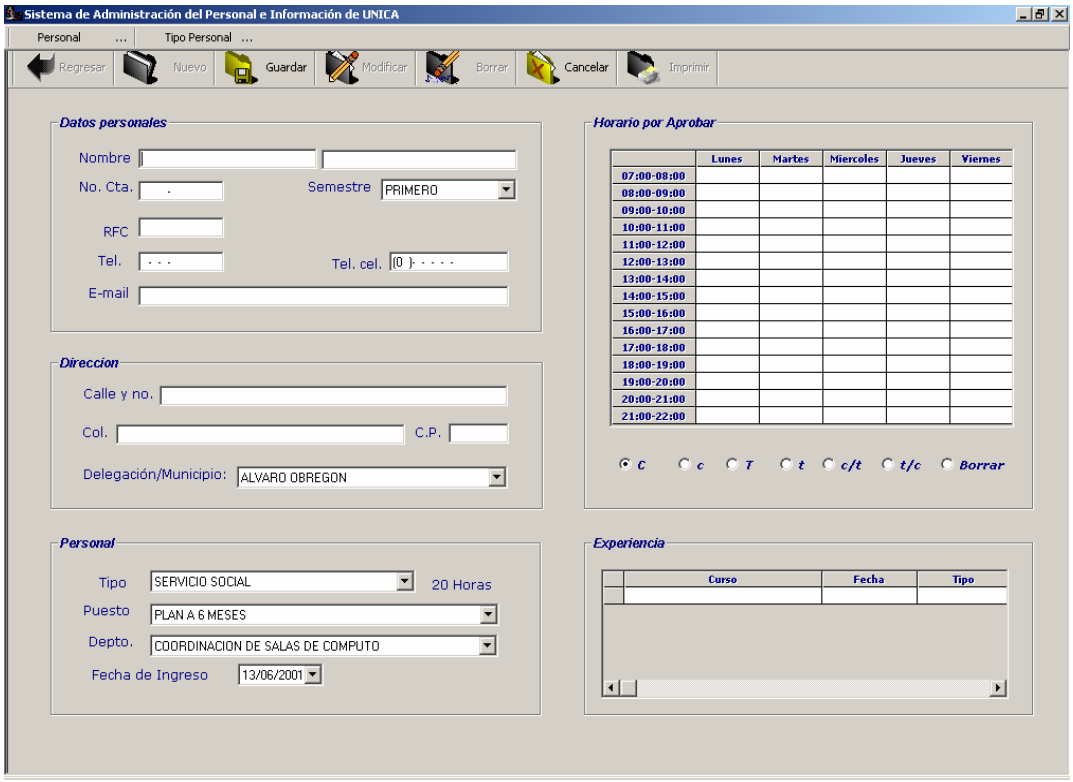


Fig. A.4 Interfaz para Nueva Persona

Dentro de esta, se cuenta con un menú que tiene las siguientes opciones:

- ✓ *Nuevo*: Permite ingresar información de la nueva persona.
- ✓ *Modificar*: Permite realizar alguna modificación en los datos de la persona.
- ✓ *Borrar*: Permite borrar el registro del personal seleccionado.
- ✓ *Guardar*: Guarda las modificaciones hechas o bien los datos del nuevo personal.
- ✓ *Cancelar*: Cancela la inserción o modificación de datos.
- ✓ *Imprimir*: Muestra un reporte con la información del personal y su horario de trabajo.
- ✓ *Regresar*: Regresa a la interfaz principal del sistema.

Adicionalmente en la parte superior de la barra de herramientas se cuenta con dos opciones especiales:

- ✓ Tipo Personal: Permite ingresar un nuevo tipo de personal al registro del personal.

Fig. A.5 Interfaz de Tipo Personal

- ✓ Personal: Permite agregar o modificar comentarios al personal registrado.

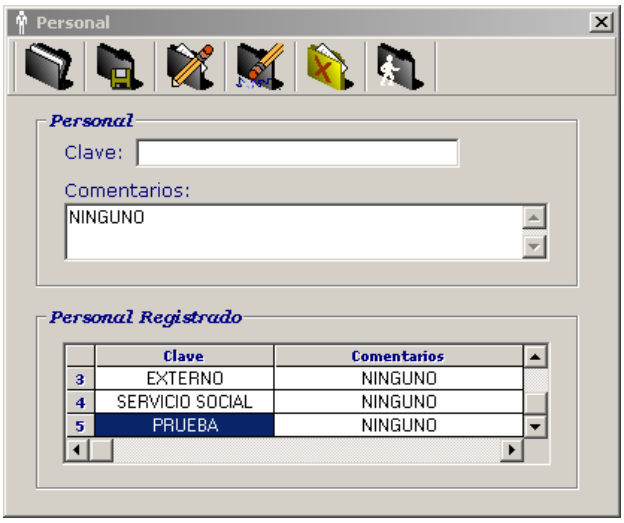


Fig. A.6 Interfaz de Personal

Si se desea ver la información de algún instructor en particular basta con dar doble clic sobre el registro que se desea ver (esta característica es similar en las interfaces de Cursos, Grupos, Costos y Salas).

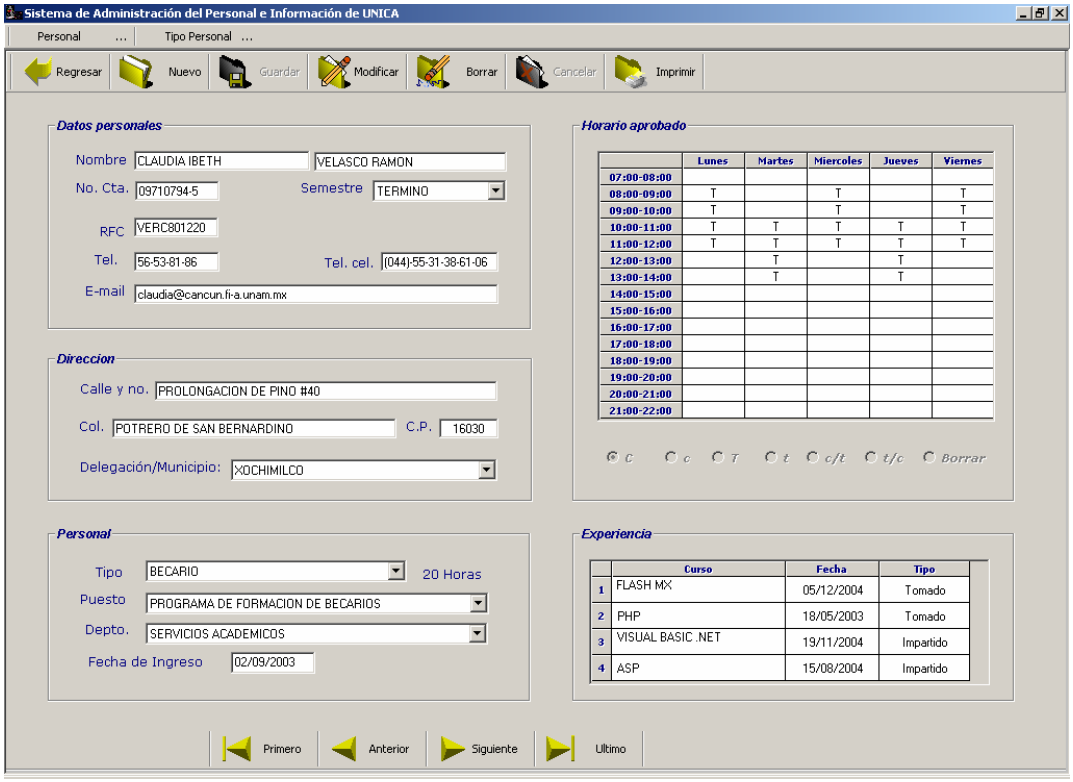


Fig. A.7 Interfaz de Instructores detallada

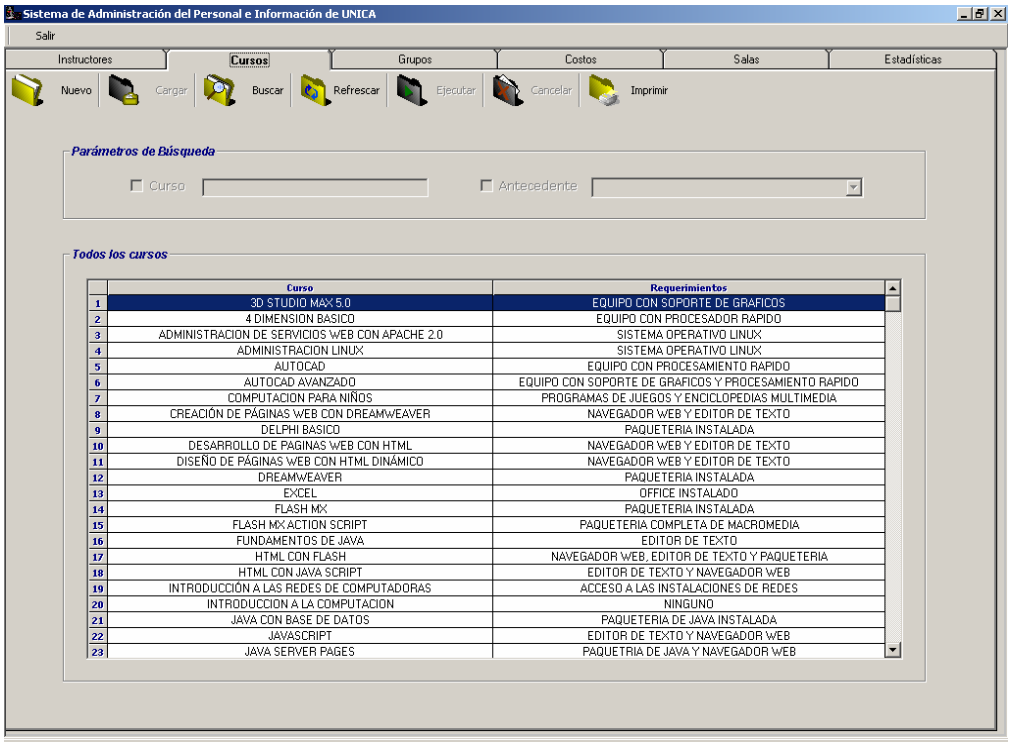
Para desplazare entre los diferentes registros que existen, se utilizará la barra de navegación que se encuentra en la parte inferior de la interfaz.

Módulo de cursos

En esta interfaz se tienen las siguientes opciones:

- ✓ *Nuevo*: Dar de alta un nuevo curso.
- ✓ *Cargar*: Cargar en pantalla todos los cursos existentes.
- ✓ *Buscar*: Búsqueda de algún curso de acuerdo a diferentes parámetros; estos son cursos y antecedentes.
- ✓ *Ejecutar:* Realiza la búsqueda de acuerdo a los parámetros seleccionados (la opción se habilita al momento de seleccionar un parámetro para la búsqueda).
- ✓ *Refrescar*: Ejecuta nuevamente la búsqueda o el cargado general de datos.

- ✓ *Cancelar*: Cancela la búsqueda.
- ✓ Imprimir: Permite imprimir el reporte de los cursos que se están visualizando.



Al dar clic sobre el botón *Nuevo* aparece la siguiente interfaz:

Fig. A.8 Interfaz de Cursos

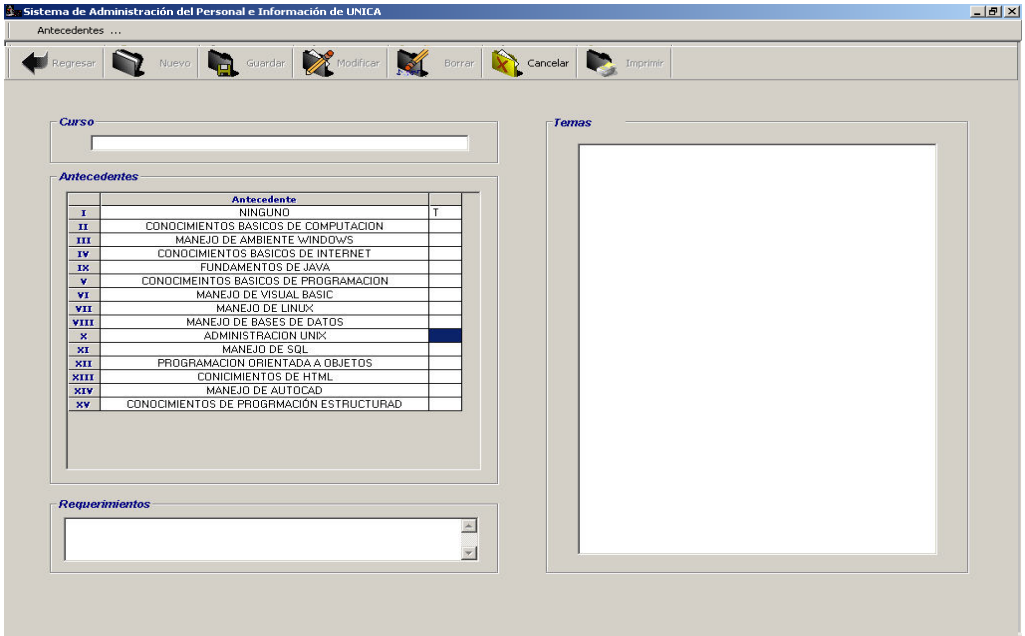


Fig. A.9 Interfaz para Nuevo Curso

En esta interfaz se cuenta con un menú que tiene las siguientes opciones:

- ✓ *Nuevo*: Permite ingresar un nuevo curso.

- ✓ *Modificar*: Permite realizar alguna modificación en los datos del curso.
- ✓ *Borrar*: Permite borrar el registro del curso seleccionado.
- ✓ *Guardar*: Guarda las modificaciones o los datos de un nuevo curso.
- ✓ *Cancelar*: Cancela la inserción o modificación de datos.
- ✓ *Imprimir*: Muestra un reporte con la información del curso.
- ✓ *Regresar:* Regresa a la pantalla principal del sistema.

Adicionalmente en la parte superior de la barra de herramientas se cuenta con una opción especial:

- ✓ Antecedentes: Permite agregar o modificar los antecedentes de los cursos.

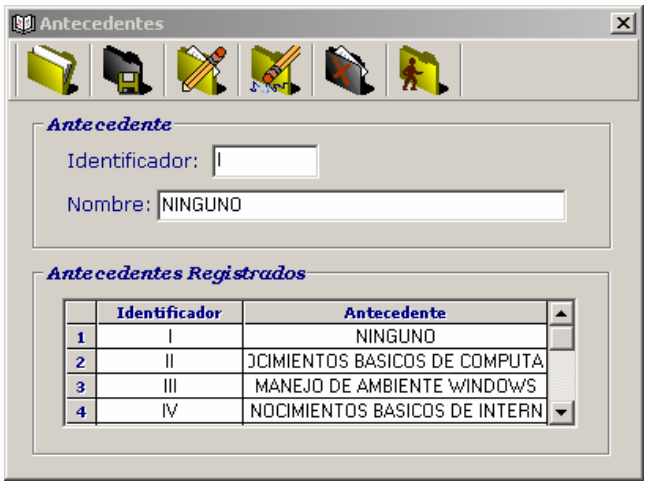


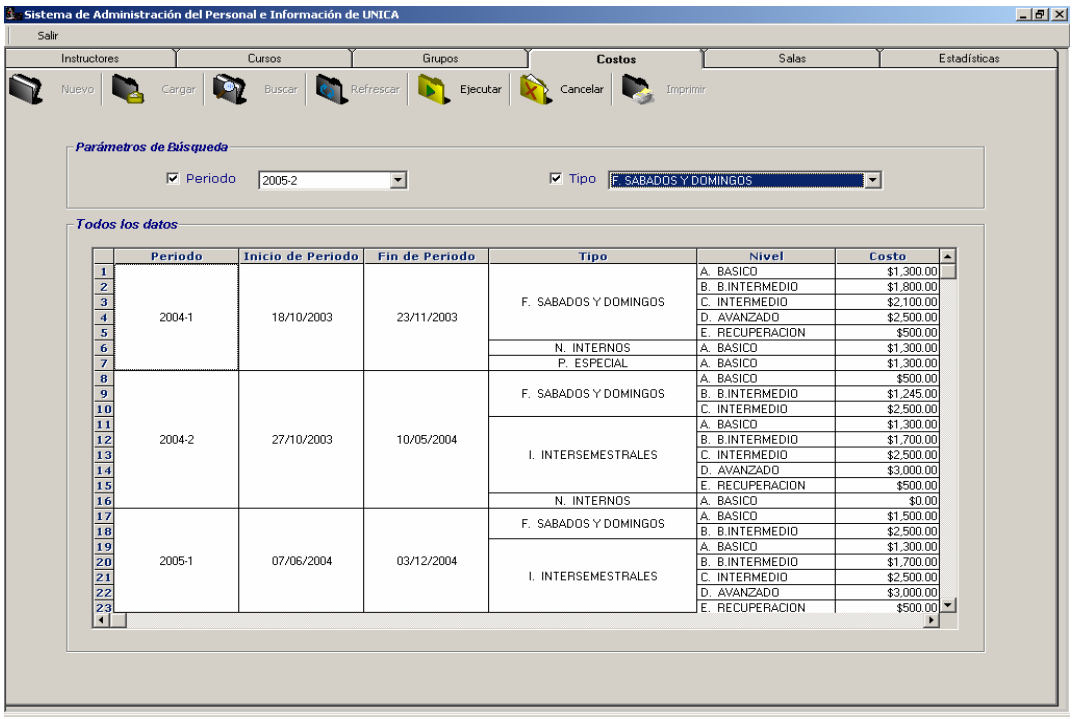
Fig. A.10 Interfaz de Antecedentes

Si se desea ver la información de algún curso en particular basta con dar doble clic sobre el registro que se desea ver.

Módulo de costos

Dentro de esta interfaz se tienen las siguientes opciones:

- ✓ *Nuevo*: Dar de alta un nuevo costo.
- ✓ *Cargar*: Cargar en pantalla todos los costos existentes.
- ✓ *Buscar*: Búsqueda de algún costo de acuerdo a los siguientes parámetros; periodo y tipo de curso.
- ✓ *Ejecutar:* Realiza la búsqueda de acuerdo a los parámetros seleccionados (la opción se habilita al momento de seleccionar un parámetro para la búsqueda).
- ✓ *Refrescar*: Ejecuta nuevamente la búsqueda o el cargado general de datos.
- ✓ *Cancelar*: Cancela la búsqueda.
- ✓ *Imprimir*: Permite imprimir los costos que se están visualizando.



	Periodo	Inicio de Periodo	Fin de Periodo	Tipo	Nivel	Costo
1	2004-1	18/10/2003	23/11/2003	F. SABADOS Y DOMINGOS	A. BASICO	$1,300.00
2					B. B.INTERMEDIO	$1,800.00
3					C. INTERMEDIO	$2,100.00
4					D. AVANZADO	$2,500.00
5					E. RECUPERACION	$500.00
6				N. INTERNOS	A. BASICO	$1,300.00
7				P. ESPECIAL	A. BASICO	$1,300.00
8	2004-2	27/10/2003	10/05/2004	F. SABADOS Y DOMINGOS	A. BASICO	$500.00
9					B. B.INTERMEDIO	$1,245.00
10					C. INTERMEDIO	$2,500.00
11				I. INTERSEMESTRALES	A. BASICO	$1,300.00
12					B. B.INTERMEDIO	$1,700.00
13					C. INTERMEDIO	$2,500.00
14					D. AVANZADO	$3,000.00
15					E. RECUPERACION	$500.00
16				N. INTERNOS	A. BASICO	$0.00
17	2005-1	07/06/2004	03/12/2004	F. SABADOS Y DOMINGOS	A. BASICO	$1,500.00
18					B. B.INTERMEDIO	$2,500.00
19				I. INTERSEMESTRALES	A. BASICO	$1,300.00
20					B. B.INTERMEDIO	$1,700.00
21					C. INTERMEDIO	$2,500.00
22					D. AVANZADO	$3,000.00
23					E. RECUPERACION	$500.00

Fig. A.11 Interfaz de Costos

Al dar clic al botón *Nuevo* aparece la siguiente interfaz:

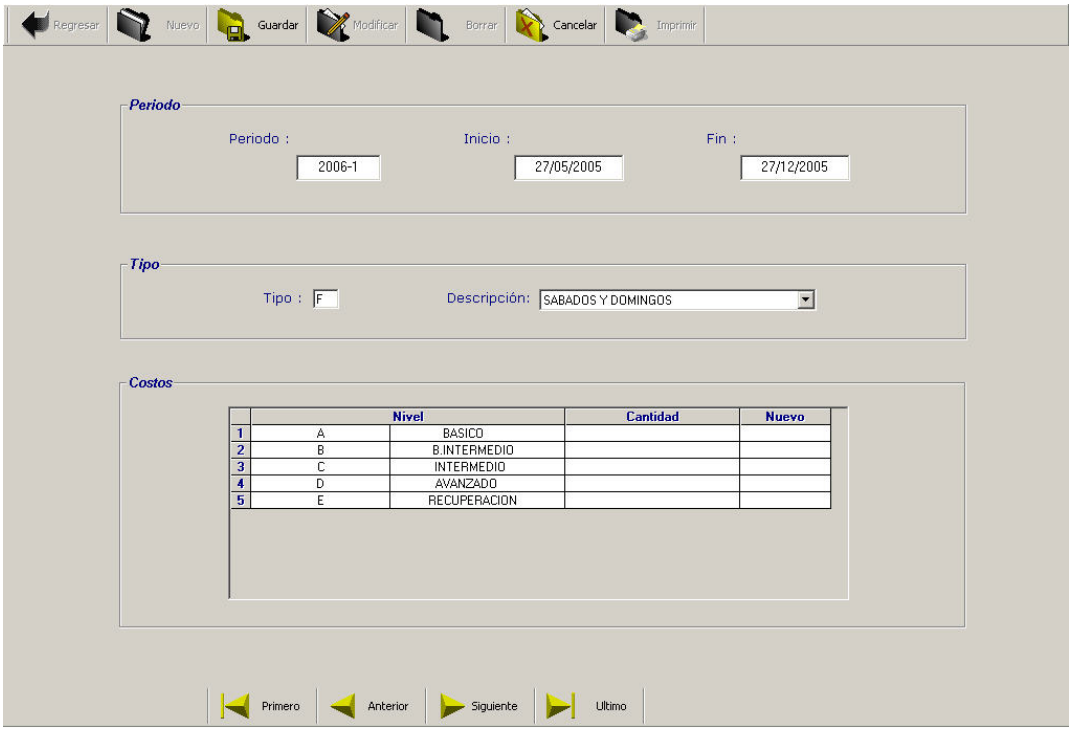


Fig. A.12 Interfaz para Nuevo Costo

Ésta cuenta con un menú que tiene las siguientes opciones:

- ✓ *Nuevo:* Permite ingresar un nuevo costo.
- ✓ *Modificar*: Permite realizar alguna modificación en los datos del costo seleccionado.
- ✓ *Borrar*: Permite borrar el registro del costo seleccionado.
- ✓ *Guardar*: Guarda los cambios hechos o guarda los datos del nuevo costo.
- ✓ *Cancelar*: Cancela la inserción o modificación de datos
- ✓ *Imprimir*: Nos muestra un reporte con la información de los costos.
- ✓ *Regresar:* Regresa a la pantalla principal de la aplicación.

Además cuenta con las siguientes opciones especiales:

- ✓ Período: Indicar un nuevo periodo, con su fecha de inicio y termino.

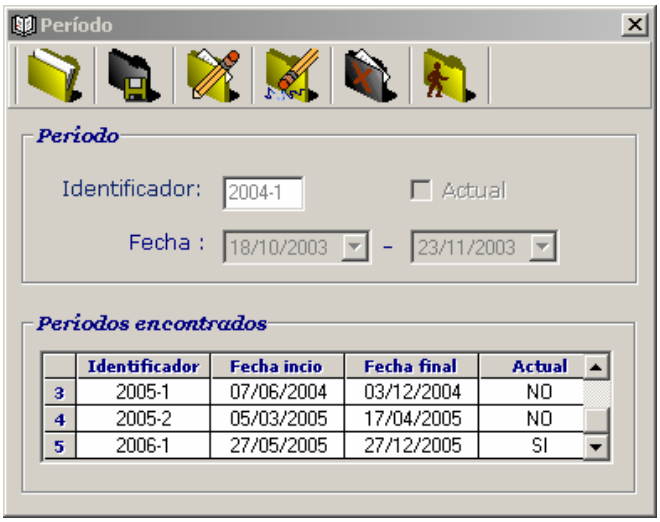


Fig. A.13 Interfaz de Período

- ✓ Nivel: Permite indicar el nivel de los cursos.

Fig. A.14 Interfaz de Nivel

Módulo de salas

Una vez que se ha dado clic en esta parte, es posible administrar la información de las salas con las que cuenta UNICA para impartir los cursos. En esta interfaz tenemos las siguientes opciones:

- ✓ *Nuevo*: Dar de alta una nueva sala
- ✓ *Cargar*: Cargar en pantalla todas los salas existentes.
- ✓ *Buscar*: Búsqueda de alguna sala de acuerdo a diferentes parámetros.
- ✓ *Refrescar*: Actualiza la ultima consulta realizada.
- ✓ *Ejecutar*: Ejecuta la búsqueda de alguna sala, (la opción se habilita al momento de seleccionar un parámetro para la búsqueda).
- ✓ *Cancelar*: Cancela las opciones seleccionadas.
- ✓ *Imprimir*: Permite imprimir el reporte de las salas que se están visualizando.

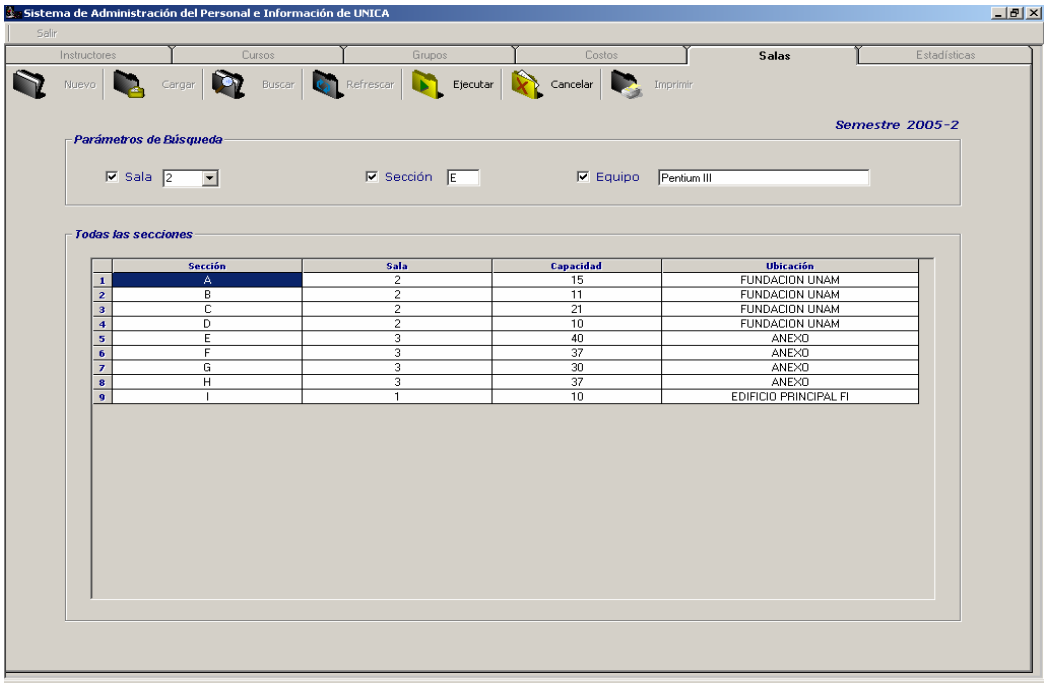


Fig. A.15 Interfaz de Salas

Al dar clic al botón *Nuevo* aparece la siguiente interfaz:

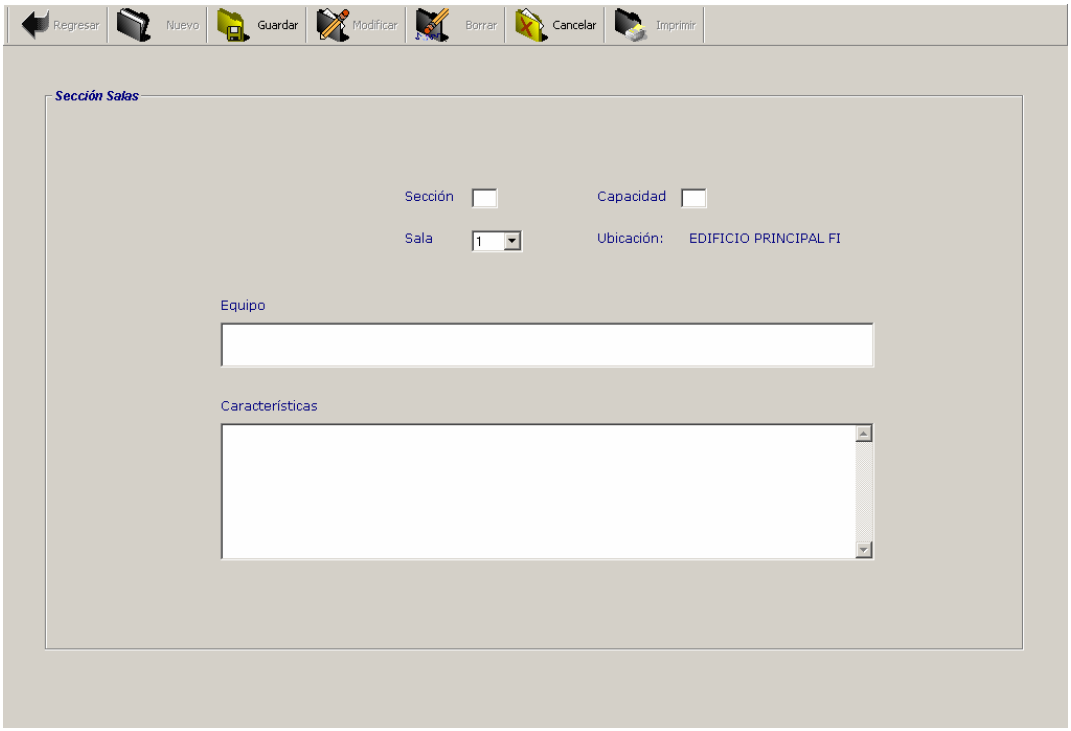


Fig. A.16 Interfaz para nueva Sala

Ésta cuenta con un menú que tiene las siguientes opciones:

- ✓ *Nuevo:* Permite ingresar una nueva sala.
- ✓ *Modificar*: Permite realizar alguna modificación en los datos de alguna sala seleccionada.
- ✓ *Borrar*: Permite borrar el registro de la sala seleccionado.
- ✓ *Guardar*: Guarda los cambios hechos o guarda los datos de la nuevo sala.
- ✓ *Cancelar*: Cancela la inserción o modificación de datos.
- ✓ *Imprimir*: Muestra un reporte con la información de las salas.
- ✓ *Regresar:* Regresa a la pantalla principal de la aplicación.

De manera análoga a las anteriores para ver el detalle de cada sala solo se debe dar doble clic sobre el registro del que se desea tener el detalle; aquí se muestran el tipo de equipo de cómputo con el que cuenta la sala y sus características de software y capacidad.

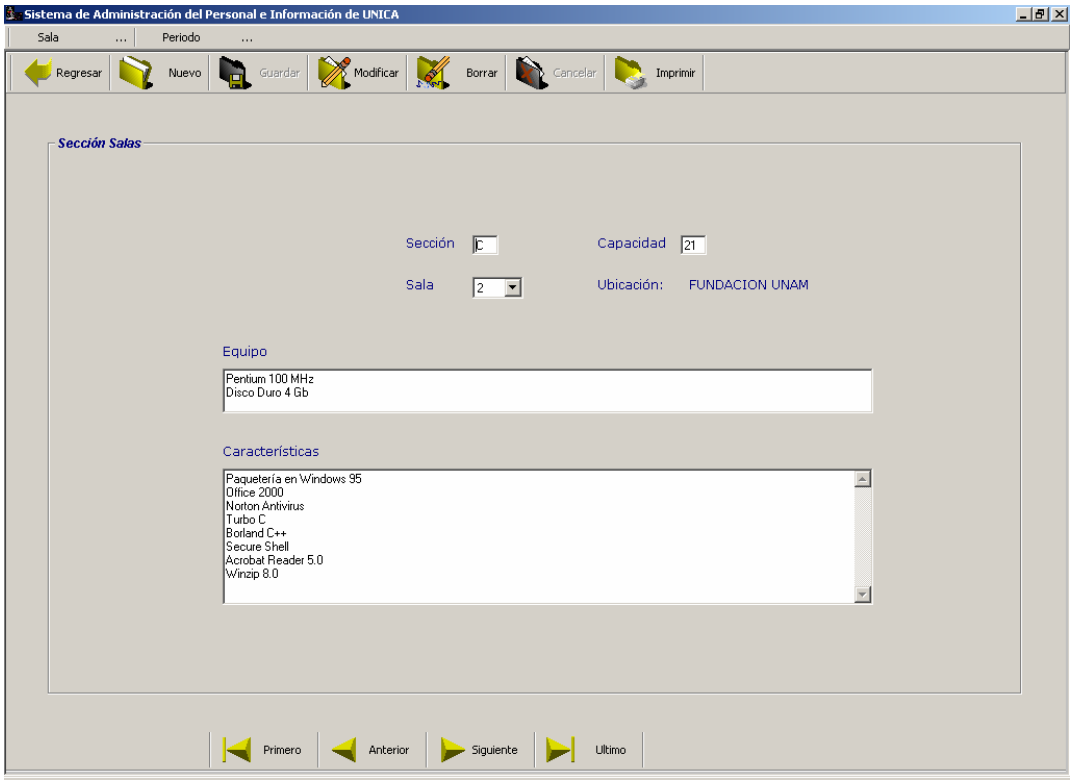


Fig. A.17 Interfaz de Salas Detalladas

Módulo de estadísticas

En esta sección, es posible visualizar los reportes estadísticos de los cursos, dependiendo del periodo seleccionado; estos son:

- ✓ *Tipos de Grupos*: Este reporte muestra el total de cursos abiertos y cancelados por periodo.
- ✓ *Tipos de Alumnos*: Indica gráficamente el porcentaje alumnos que predominan en los cursos.
- ✓ *Tipos de Costos*: Muestra los costos por cada tipo de curso en el periodo.

Fig. A.18 Reporte por tipo de grupo

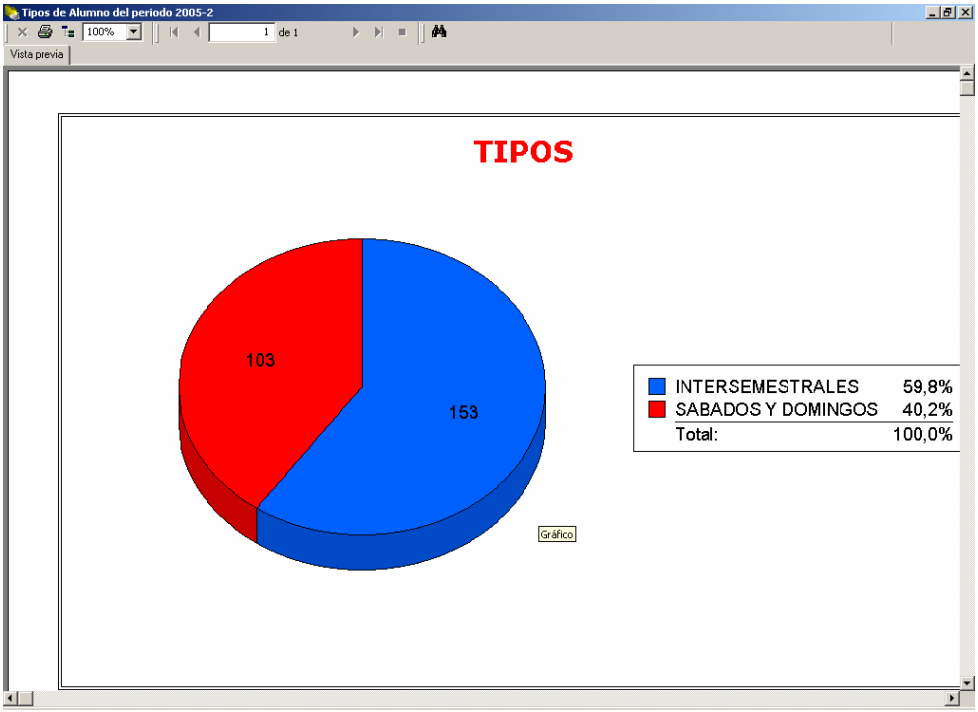


Fig. A.19 Reporte por tipo de alumno

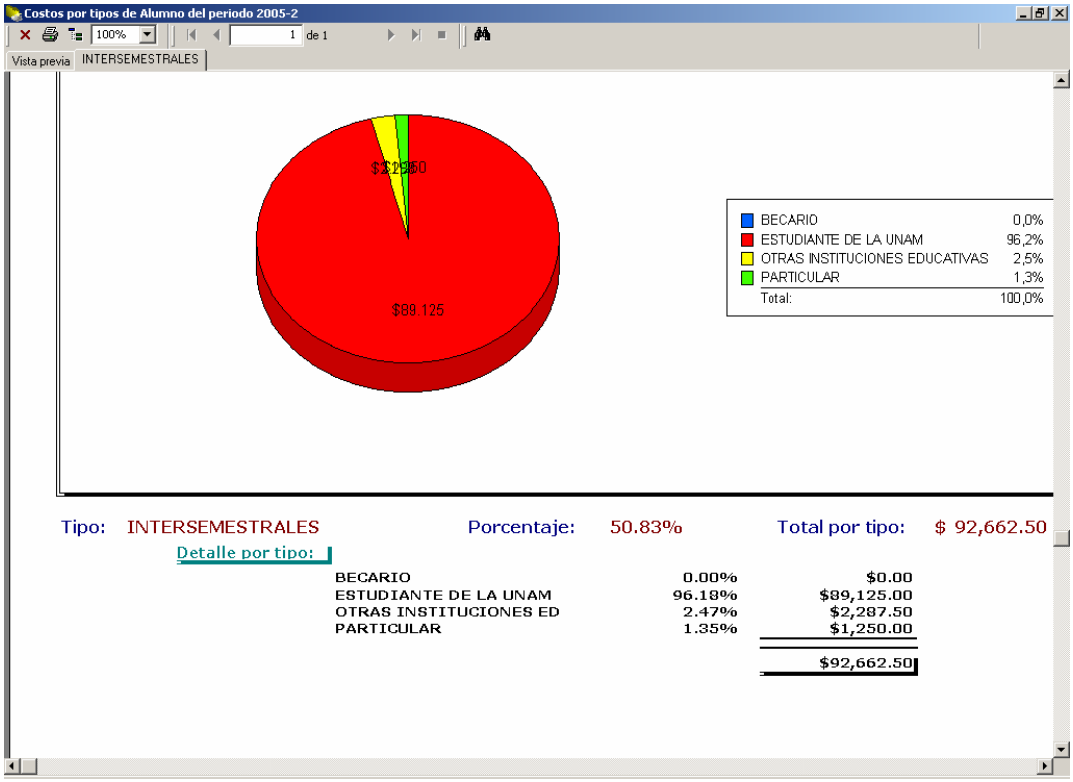


Fig. A.20 Reporte por tipo de costo

SISTEMA DE APROBACIÓN DE HORARIO

Como en la caso del sistema de administración se cuenta con una interfaz de validación, en este caso cada jefe de departamento tendrá un usuario y contraseña diferente para visualizar solo los datos del personal a su cargo.

Fig. A.21 Interfaz de Validación

Después de realizar la validación correspondiente, se presenta la interfaz de bienvenida:

Fig. A.21 Interfaz de Bienvenida

Inmediatamente después se presenta la interfaz principal del sistema, es este caso solo cuenta con la interfaz de instructores, y tiene el mismo funcionamiento que la interfaz de instructores del sistema administrativo. Ésta interfaz brinda a cada jefe la posibilidad de imprimir una lista general del todo el personal que labora en UNICA o solo del personal a su cargo con sus principales datos.

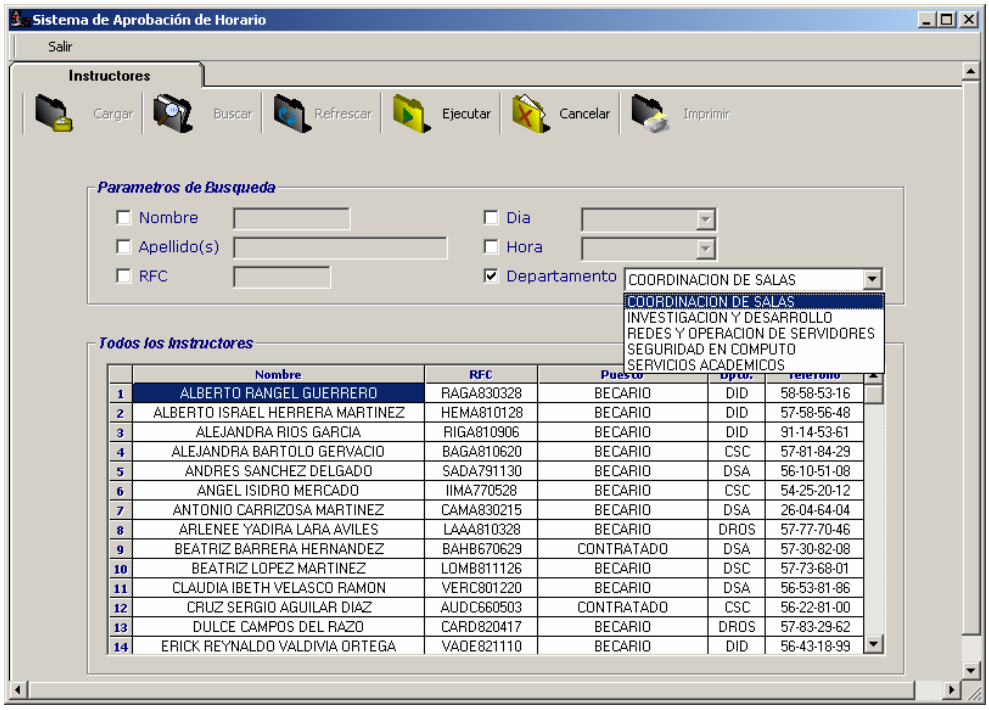


Fig. A.22 Interfaz Principal del Sistema de Aprobación de Horario

La diferencia principal con el sistema administrativo se presenta en la interfaz detallada de instructores; ya que la barra de herramientas cambia ofreciendo la posibilidad de aprobar o no el horario de trabajo del personal e imprimir el reporte con sus datos personales y horario de trabajo.

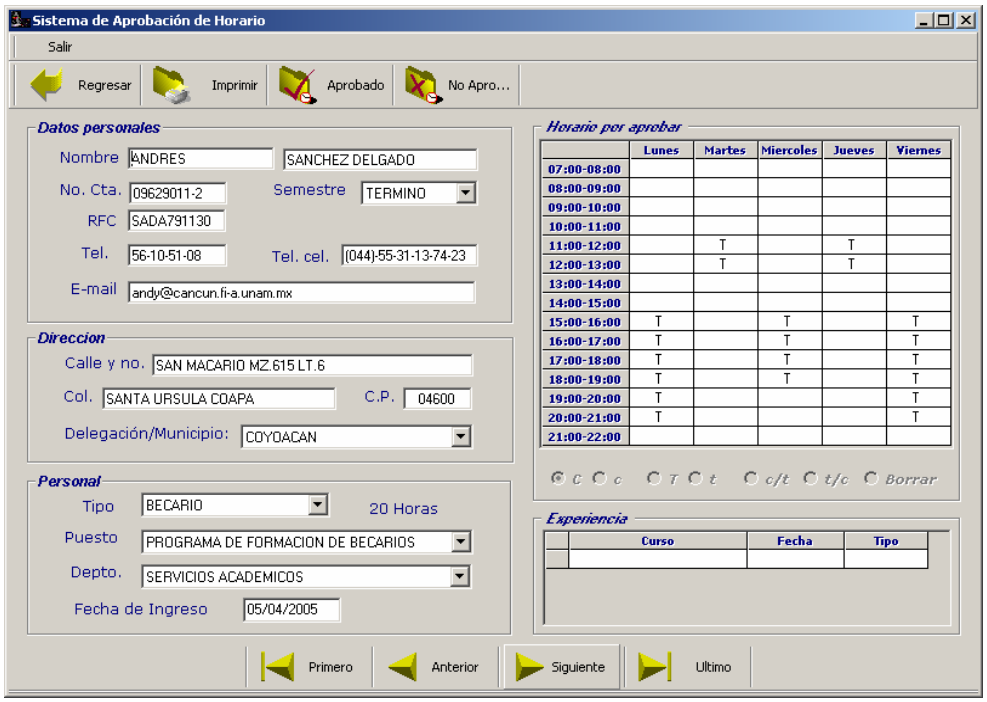


Fig. A.23 Interfaz detallada del Personal

SISTEMA EN WEB

Para el caso del sistema en WEB también se cuenta con una interfaz de validación, en este caso para que cada miembro del personal de UNICA pueda tener acceso, debe proporcionar su nombre(s) y RFC.

Fig. A.24 Interfaz de validación

Una vez que se ha confirmado que el usuario es válido se muestra la interfaz de bienvenida:

Fig. A.25 Interfaz de bienvenida

Después de dar clic a siguiente se muestra la interfaz principal del sistema, en la se tienen tres opciones:

Fig. A.26 Interfaz de opciones

En el apartado de Datos Personales, se tiene la posibilidad de cambiar los datos personales o bien modificar el horario de trabajo en UNICA, proceso que se debe realizar cada inicio de semestre.

Fig. A.27 Interfaz de Datos Personales

En la sección de propuesta, el personal puede hacer las propuestas de cursos para cada periodo de impartición de los mismos:

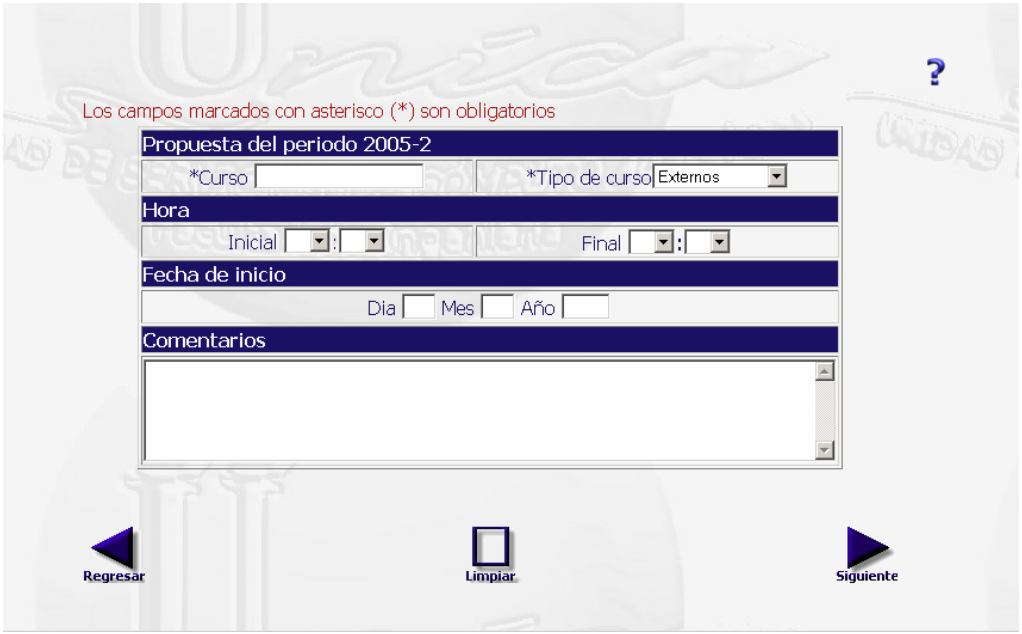


Fig. A.28 Interfaz de Propuesta

Finalmente en el apartado de calificaciones, cada instructor puede asentar las calificaciones finales de los cursos impartidos durante el periodo o bien modificarlas en el caso de alguna aclaración.

Fig. A.28 Interfaz de Calificaciones

Bibliografía

PRESSMAN, Roger S. Ingeniería de Software: un enfoque práctico. 4ª.edición. México, Mcgraw-Hill.

JACOBSON, Ivar El Proceso Unificado de Desarrollo de Software. Madrid. Addison Wesley.

REED, Paul R. Developing applications with Visual Basic and UML. ISBN 0-201-61579-7

ERIKSSON,Hans-Erick, Magnus Penker
Business modeling with UML: business patterns at work.
ISBN 0-471-29551-5

OESTEREICH, Bernd
Developing software with UML: object-oriented analysis design in practice.
ISBN 0201398265

CORNELL, Gary Visual Basic 6.0, Manual de referencia.
España,McGraw-Hill

FACULTAD DE INFORMATICA UPV/EHU INGENIERÍA EN INFORMATICA DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMATICOS. 2003. Curso 2003/04 Pruebas de Software. [en línea]. <http://siul02.si.ehu.es/~alfredo/ iso/Tema3.pdf>