U N I V E R S I D A D N A C I O N A L
A U T Ó N O M A D E M É X I C O

FACULTAD DE INGENIERÍA

"Sistema de Gestión de Reuniones vía WEB"

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:

Martha Angélica Nakayama Cervantes

DIRECTOR DE TESIS:

Ing. Carlos Alberto Román Zamitiz

MÉXICO, D.F. **NOVIEMBRE de 2005**

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicatoria

A mi mamá, que ha sido lo más importante en mi vida
A mis tíos, que me han dado todo su apoyo y cariño
A mi primo, que ha sido un hermano para mí
A mis abuelos, que desde donde estén comparten este momento conmigo
A Alexis, por todo su amor, comprensión y apoyo incondicional
A mis amigos, por todo lo que compartimos juntos y estar conmigo en todo momento
A la UNAM por ser simplemente la mejor

ÍNDICE

Tabla de ilustraciones

Introducción

En la actualidad dados los avances tecnológicos es muy común encontrar en cualquier lugar sistemas de cómputo y software hecho a la medida. Para tener mayor productividad, muchas organizaciones han incorporado sistemas de este tipo para poder llevar un mejor control y desempeño en todas sus áreas, así como para ser competitivos tecnológicamente hablando.

Además con la importancia que la Internet ha adquirido a fechas recientes, la mayoría de los sistemas tienen un enfoque WEB, ya que gracias a dicho enfoque se tiene una mayor cobertura y expansión.

Por otro lado, en la mayoría de las empresas, negocios organizaciones o instituciones se llevan a cabo reuniones para toma de decisiones importantes, coordinar proyectos, etc. pero pocas veces se tiene una reunión exitosa por falta de una metodología y las herramientas necesarias para optimizar el trabajo de cada uno de los participantes.

Por tales motivos se propone crear un sistema que aproveche las ventajas que nos ofrece la Internet, es decir, un sistema vía WEB, que sirva como herramienta de una metodología de gestión de reuniones.

Para ello en el primer capítulo se presentarán las bases teóricas de Bases de Datos, Ingeniería de Software, Orientación a Objetos, UML, etc. que se aplicarán a lo largo del desarrollo del sistema.

Las técnicas de modelado de sistemas nos señalan que debemos empezar el desarrollo de un sistema por la obtención de requerimientos para saber las necesidades de los usuarios finales, por lo que en el segundo capítulo se hablará de los requerimientos, su importancia y su aplicación.

En el tercer capítulo se tratará el análisis de los requerimientos obtenidos y los diagramas necesarios y su importancia para poder tener una estimación adecuada de la duración e impacto del sistema.

Una vez realizadas las etapas anteriores se debe llevar a cabo el diseño, que se detallará en el cuarto capítulo, esto basado en el análisis previo y las posibles tecnologías a utilizar no olvidando que de un buen diseño depende el éxito o el fracaso del sistema en cuestión.

La etapa de desarrollo se verá en el quinto capítulo y es aquí cuando tendremos el sistema funcionando, aunque es posible que se necesite repetir el ciclo de desarrollo del software para realizar mejoras y / o mantenimiento.

Una vez terminado el sistema se hará la liberación del sistema, para esto en el sexto capítulo se tendrán los pasos a seguir para implementar el sistema y llevar el control de las versiones si posteriormente se realiza mantenimiento o mejoras.

Finalmente en el séptimo capítulo se tendrán las conclusiones a las que se llegó al finalizar el sistema, su impacto y beneficios para los usuarios finales.

Se puede asegurar que a lo largo del desarrollo del sistema se estarán cumpliendo con todas las etapas mencionadas y se utilizarán las tecnologías más adecuadas para el análisis, diseño y desarrollo, así como también se cuidará de cumplir con los objetivos deseados y cubrir las necesidades de los usuarios.

CAPITULO 1 ANTECEDENTES

1.- Ingeniería de Software

Hoy en día es común encontrar computadoras en cualquier lugar al que vayamos. La computación se ha extendido a todo tipo de industrias y a todos los ámbitos de la sociedad. Esto implica que la gente haga uso de sistemas computacionales para operar todas las computadoras tanto en el hogar como en el trabajo, la industria, etc. Por lo tanto el desarrollo de sistemas es una de las actividades de mayor demanda y el uso de tecnologías de información y de la ingeniería de software va cada día más en aumento.

Una descripción del software de un libro de texto puede ser la siguiente:

"Instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el comportamiento deseado; estructuras de datos que facilitan a los programas manipular adecuadamente la información"

Para poder comprender lo que es el software y la ingeniería de Software, es importante examinar las características del software que lo diferencian de otras cosas que los hombres pueden construir. Cuando se construye hardware, el proceso creativo humano se traduce finalmente en una forma física. El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto, el software tiene unas características considerablemente distintas a las del hardware:

1. *El software se desarrolla, no se fabrica en un sentido clásico.*
   Aunque existen algunas similitudes entre el desarrollo y la construcción del hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen o son fácilmente corregibles en el software.

2. *El software no se "estropea".*
   En las siguiente figura se describe, para el hardware, la proporción de fallos como función del tiempo, la cual indica que el hardware exhibe relativamente muchos fallos al principio de su vida y una vez corregidos los defectos, la tasa de fallos cae hasta un nivel estacionario donde permanece durante un cierto periodo de tiempo. Sin embargo, conforme pasa el tiempo, los fallos vuelven a presentarse a medida que los componentes del hardware sufren los efectos acumulativos de la suciedad, la vibración, los malos tratos, las temperaturas extremas y muchos otros males externos. Sencillamente, el hardware comienza a estropearse.

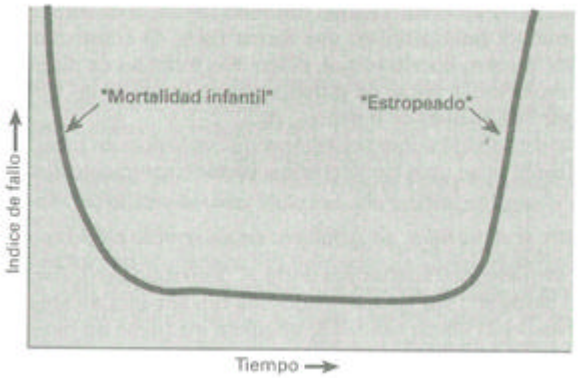


Figura 1 Curva de fallos del hardware

El software no es susceptible a los males del entorno que hacen que el hardware se estropee. Por tanto, en teoría la curva de fallos tendría la forma que muestra la figura 2. Los defectos no detectados harán que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen, suponiendo que no se introducen nuevos errores, la curva se aplana. Esa figura es una gran simplificación de los modelos reales del software.

Figura 2 Curva de fallos del software

Sin embargo la implicación es clara, el software no se estropea, pero se deteriora.

Esto, que parece una contradicción, puede comprenderse mejor considerando la siguiente figura, durante su vida, el software sufre cambios (mantenimiento). Conforme se hacen estos cambios, es bastante probable que se introduzcan nuevos defectos, haciendo que la curva de fallos tenga picos como se ve en la figura 3. Antes de que la curva pueda volver al estado estacionario original, se solicita otro cambio, haciendo que de nuevo se cree otro pico. Lentamente el nivel mínimo de fallos comienza a crecer; el software se va deteriorando debido a los cambios.

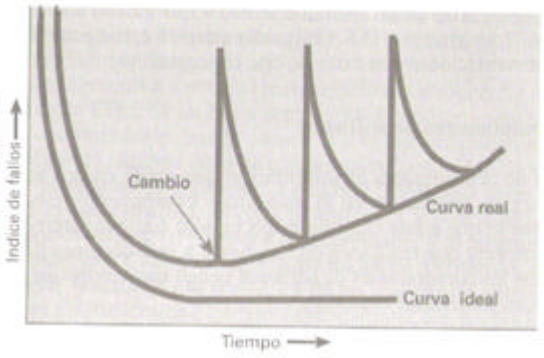


Figura 3 Curva real de fallos del software

3. *La mayoría del software se construye a medida, en vez de ensamblar componentes existentes.*
   Consideremos la forma en la que se diseña y construye el hardware , el ingeniero de diseño construye un esquema de la circuitería digital, hace algún análisis fundamental y va al catálogo de ventas de componentes digitales y después de seleccionar cada componente, puede solicitarse la compra. Por desgracia, los diseñadores del software no disponen de esa comodidad, con unas pocas excepciones, no existen catálogos de componentes de software. Se puede comprar software ya desarrollado, pero sólo como una unidad completa, no como componentes que puedan reensamblarse en nuevos programas (aunque esta situación esta cambiando rápidamente). Aunque se ha escrito mucho sobre “reusabilidad del software”, estamos comenzando a ver las primeras implementaciones con éxito de este concepto.

Lamentablemente el desarrollo del software no siempre genera software de buena calidad y es muy común encontrar sistemas cuyo tiempo de vida queda por debajo del estimado inicialmente. La demanda de software ha crecido mucho más rápido que nuestra capacidad para crearlo. Además, el país requiere un tipo de software más práctico, fiable y robusto que el que se esta desarrollando hoy en día.

Los problemas que afligen al desarrollo del software se pueden caracterizar bajo muchas perspectivas diferentes, pero los responsables de los desarrollos de software se centran sobre los aspectos de “fondo”: La planificación y estimación de costes son frecuentemente muy imprecisas; la “productividad” de la comunidad del software no se corresponde con la demanda de sus servicios y la calidad del software no llega a se a veces ni aceptable. Se han experimentado desajustes en los costes de hasta un orden de magnitud. Se ha errado en la planificación en meses o años. Se ha hecho muy poco para mejorar la productividad de los trabajadores de software. Los errores en los nuevos programas producen en los clientes insatisfacción y falta de confianza. Tales problemas son sólo las manifestaciones más visibles de otras dificultades del software:

- ✓ No tenemos tiempo de recoger datos sobre el proceso de desarrollo del software. Sin datos históricos como guía, la estimación no ha sido buena y los resultados previstos son muy pobres. Sin una indicación sólida de la productividad, no podemos evaluar con precisión la eficacia de las nuevas herramientas, técnicas o estándares.
- ✓ La insatisfacción del cliente con el sistema "terminado" se produce demasiado frecuentemente. Los proyectos de desarrollo del software se acometen generalmente con una vaga indicación de los requisitos del cliente. Normalmente, la comunicación entre el cliente y el que desarrolla el software es muy escasa.
- ✓ La calidad del software es normalmente cuestionable. Hemos empezado a comprender recientemente la importancia de la prueba sistemática y técnicamente completa del software. Están comenzando a emerger conceptos cuantitativos sólidos sobre la fiabilidad del software y las garantías de calidad.
- ✓ El software existente puede ser muy difícil de mantener. La tarea de mantenimiento del software se lleva la mayor parte de todo el dinero invertido en el software. El mantenimiento no se ha considerado un criterio importante en la aceptación del software.

Uno de los principales problemas en la actualidad es que en el desarrollo de software en gran número de proyectos, la programación se empieza tan pronto nos asignan los requerimientos del sistema, omitiendo las etapas de análisis y diseño. Esto genera software de mala calidad, considerando que aproximadamente el 70% del tiempo de desarrollo de un sistema debería ser de análisis y diseño mientras que el 30% restante debería ser para las etapas de programación y pruebas.

Lo anterior nos lleva a una etapa conocida como "La crisis del software" donde el software generado supera los costos, tiempos de desarrollo y queda por debajo de las expectativas del usuario y de su tiempo de vida. Esto es consecuencia del mal desarrollo del sistema.

Las buenas noticias son que todos los problemas descritos anteriormente pueden corregirse, la clave está en dar un enfoque de ingeniería al desarrollo del software, junto con la mejora continua de las técnicas y de las herramientas.

La Ingeniería de Software es la parte de la ingeniería que nos rige en el desarrollo de sistemas. Una de las primeras definiciones de ingeniería del software fue la propuesta por Fritz Bauer:

"El establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que sea fiable y funcione de manera eficiente sobre máquinas reales"

Aunque se han propuesto muchas más definiciones generales, todas refuerzan la importancia de una disciplina de ingeniería para el desarrollo del software.

Proceso de desarrollo de software

Un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. En la ingeniería de software el objetivo es construir un producto software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que el estado actual de la tecnología permite. En consecuencia, reduce el riesgo y hace el proyecto más predecible. El efecto global es el fomento de una visión y una cultura comunes.

Es necesario un proceso que sirva como guía para todos los participantes (clientes, usuarios, desarrolladores y directores ejecutivos). No nos sirve ningún proceso antiguo; necesitamos uno que sea el mejor proceso que la industria pueda reunir en este punto de su historia. Por último necesitamos un proceso que esté altamente disponible en la forma que todos los interesados puedan comprender su papel en el desarrollo en el que se encuentran implicados.

Un proceso de desarrollo de software debería también ser capaz de evolucionar durante muchos años. Durante esta evolución debería limitar su alcance, en un momento del tiempo dado, a las realidades que permitan las tecnologías, herramientas, personas y patrones de organización.

Tecnologías:

El proceso debe construirse sobre las tecnologías (lenguajes de programación, sistemas operativos, computadoras, estructuras de red, entornos de desarrollo, etc.) disponibles en el momento en que se va a emplear el proceso.

Herramientas:

Los procesos y las herramientas deben desarrollarse en paralelo. Las herramientas son esenciales en el proceso. Dicho de otra forma, un proceso ampliamente utilizado puede soportar la inversión necesaria para crear las herramientas que lo soporten.

Personas:

Un creador del proceso debe limitar el conjunto de habilidades necesarias para trabajar en el proceso a las habilidades que los desarrolladores actuales poseen, o apuntar aquellas que los desarrolladores puedan obtener rápidamente.

Patrones de organización:

El creador del proceso a las realidades del momento, hechos como las empresas virtuales; el trabajo a distancia, la mezcla de socios de la empresa, empleados asalariados, trabajadores por obra y subcontratas de outsourcing y la prolongada escasez de desarrolladores de software.

Los ingenieros del proceso deben equilibrar estos cuatro conjuntos de circunstancias. Además, el equilibrio debe estar presente no solo ahora, si no también en el futuro. El creador del proceso debe diseñar el proceso de forma que pueda evolucionar, de igual forma que el desarrollador de software intenta desarrollar un sistema que no solo funciona este año, si no que evoluciona con éxito en los años venideros. Un proceso debe madurar durante varios años antes de alcanzar el nivel de estabilidad y madurez que le permitirá resistir a los rigores del desarrollo de productos comerciales, manteniendo a la vez un nivel razonable de riesgo en su utilización. En estas circunstancias, un proceso puede ser estable.

Sin este equilibrio de tecnologías, herramientas, personas y organización, el uso del proceso sería bastante arriesgado.

El proceso de desarrollo del software contiene las tres fases genéricas: Definición, desarrollo y mantenimiento, las cuales se encuentran en todos los desarrollos de software, independientemente del área de aplicación, del tamaño del proyecto o de la complejidad.

Definición

La fase de definición se centra sobre el *"que"*. Esto es, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, que interfaces han de establecerse, qué restricciones de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto. Por tanto, han de identificarse los requisitos clave del sistema y del software. Aunque los métodos aplicados durante la fase de definición varían, de alguna forma se producirán tres pasos específicos:

1. *Análisis del sistema.*
   El análisis del sistema define el papel de cada elemento de un sistema informático, asignando finalmente al software el papel que va a desempeñar.
2. *Planificación del proyecto de software.*
   Una vez establecido el ámbito del software, se analizan los riesgos, se asignan los recursos, se estiman los costes, se definen las tareas y se planifica el trabajo.
3. *Análisis de requisitos.*
   El ámbito establecido para el software proporciona la dirección a seguir, pero antes de comenzar a trabajar es necesario disponer de una información más detallada del ámbito de información y de función del software.

Desarrollo

La fase de desarrollo se centra en el *"como"*. Esto es, durante esta fase, el que desarrolla el software intenta descubrir cómo han de diseñarse las estructuras de datos y la arquitectura del software, cómo han de implementarse los detalles

procedimentales, cómo ha de traducirse el diseño a un lenguaje de programación y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo varían, pero de alguna forma se producirán tres pasos concretos:

1. *Diseño del software.*
   El diseño traduce los requisitos del software a un conjunto de representaciones que describen la estructura de los datos, la arquitectura, el procedimiento algorítmico y las características de la interfaz.
2. *Codificación.*
   Las representaciones del diseño deben ser traducidas a un lenguaje artificial, dando como resultado unas instrucciones ejecutables por la computadora. El paso de la codificación es el que lleva a cabo esa traducción.
3. *Prueba del software.*
   Una vez que el software ha sido implementado en una forma ejecutable por la máquina, debe ser probado para descubrir los defectos que puedan existir en la función, en la lógica y en la implementación.

Mantenimiento

La fase de mantenimiento se centra en el *"cambio"* que va asociado a la corrección de errores, a las adaptaciones requeridas por la evolución del entorno del software y a las modificaciones debidas a los cambios de los requisitos del cliente dirigidos a reforzar o a ampliar el sistema. La fase de mantenimiento vuelve a aplicar los pasos de las fases de definición y de desarrollo, pero en el contexto del software ya existente. Durante la fase de mantenimiento se encuentran tres tipos de cambios:

1. *Corrección.*
   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.
2. *Adaptación.*
   Con el paso del tiempo es probable que cambie el entorno original para el que se desarrolló el software. El mantenimiento adaptativo consiste en modificar el software para acomodarlo a los cambios de su entorno externo.
3. *Mejora.*
   Conforme utilice el software, el cliente / usuario puede descubrir funciones adicionales que podría interesar que estuvieran incorporadas en el software. El mantenimiento perfectivo amplía el software más allá de sus requisitos funcionales originales.

Importancia del diseño

Función del diseño y de los diseñadores:

- ✓ Pensar a largo plazo nunca viene mal.
- ✓ No se puede añadir calidad al final del proceso: hay que contrastar confiando en las pruebas; resulta más efectivo y mucho menos costoso.
- ✓ Hacer posible la delegación de tareas y el trabajo en equipo.
- ✓ Un diseño defectuoso perjudica al usuario: software difícil de utilizar, incoherente y poco flexible.
- ✓ Un diseño defectuoso también afecta al programador: interfaces pobres, proliferación de errores y dificultades para añadir nueva funcionalidad.

Existe un mito acerca del software, según el cual el diseño carece de importancia, ya que lo único que importa en realidad es el tiempo que se tarda en lanzar el producto al mercado.

Uno de los aspectos básicos del diseño de software es cómo descomponer un programa en partes. Es erróneo pensar que las especificaciones no son más que documentación tediosa. Por el contrario, resultan esenciales para el desacoplamiento y, por lo tanto, para el diseño de alta calidad.

Descomposición
Un programa se construye a partir de un conjunto de partes. El problema de la descomposición consiste en descubrir qué partes integran ese conjunto y qué relación guardan entre ellas.

Ventajas que se derivan de la división de un programa en partes pequeñas

- ✓ División del trabajo. Un programa no surge de la nada: tiene que construirse gradualmente. Si se divide en partes, el proceso de construcción se agiliza, ya que ello permite que varias personas se dediquen a trabajar en las distintas partes.
- ✓ Reusabilidad. Algunas veces es posible aislar las partes que son comunes a diferentes programas, de modo que se puedan crear una sola vez y utilizar muchas veces.
- ✓ Análisis modular. Incluso si un programa ha sido construido por una única persona, resulta conveniente construirlo por partes pequeñas. De este modo, cada vez que se completa una parte puede analizarse para comprobar su exactitud Si funciona, otra parte podrá utilizarla sin necesidad de volver sobre ella. Además de la satisfacción que supone poder progresar con rapidez, el análisis modular proporciona una ventaja más sutil. Analizar una parte que es doblemente extensa supone el doble de esfuerzo, así que analizar un programa que está descompuesto en partes pequeñas reduce drásticamente el coste global del análisis.
- ✓ Cambio localizado. Todo programa útil necesita de adaptaciones y ampliaciones a lo largo de su existencia. La posibilidad de localizar un

cambio en unas cuantas partes permite que sólo haya que tener en cuenta una porción mucho más pequeña del total del programa a la hora de llevar a cabo dicho cambio y validarlo. Las partes de un programa son descripciones: de hecho, el desarrollo de software se centra en realidad en producir, analizar y ejecutar descripciones.

En la década de los años 70 existía un enfoque generalizado sobre el desarrollo del software, llamado diseño descendente, o diseño *"top down"*. La idea de la que parte este diseño consiste simplemente en aplicar de manera recursiva el siguiente paso:

- ✓ Si la parte que se quiere construir se halla ya disponible (como, por ejemplo, las instrucciones de un aparato), el proceso ya está terminado.
- ✓ Si la parte no está disponible, se divide en sub partes, que se desarrollan y combinan entre sí.

La división en sub partes se llevaba a cabo mediante la *"descomposición funcional"*: se piensa en la función que debe tener la parte y se desglosa esa función en pasos más pequeños. La idea resultó atractiva en su momento, y tiene aún hoy en día sus defensores. Sin embargo, se trata de un enfoque que fracasa rotundamente, por la razón siguiente: la primera descomposición que se hace es la más decisiva, y aún así, no se descubre si se ha hecho correctamente hasta que no se llega al nivel más bajo del árbol de descomposición. No es posible hacer muchas evaluaciones durante el desarrollo del proceso, puesto que no se puede probar una descomposición en dos partes que no se han implementado, y una vez que se ha llegado al final, es demasiado tarde para actuar sobre las descomposiciones realizadas en niveles superiores. Por lo tanto, desde el punto de vista del riesgo (es decir, tomar decisiones sólo cuando se dispone de la información necesaria y minimizar la probabilidad de incurrir en errores y el coste de los mismos), se trata de una estrategia totalmente inadecuada.

Esto no quiere decir, por supuesto, que examinar un sistema de forma jerárquica sea una mala idea. Simplemente, no es posible desarrollarlo de ese modo.

Una estrategia mucho mejor consiste en desarrollar una estructura de sistema de múltiples partes a partir de niveles similares de abstracción. Para ello se perfecciona la descripción de cada parte de una sola vez y se analiza si las partes encajan y consiguen la funcionalidad deseada antes de empezar a implementar cualquiera de ellas. Parece, asimismo, mucho más conveniente organizar un sistema en torno a datos que en torno a funciones.

Quizás el factor más importante que hay que tener en cuenta a la hora de evaluar la descomposición en partes sea el modo en que esas partes están acopladas unas con otras. Queremos minimizar el acoplamiento (desacoplar las partes) para poder trabajar en cada una de ellas con independencia de las demás.

2.- Análisis, diseño y programación orientada a objetos

Para la realización de un sistema se requiere de paradigmas que ayuden a los creadores de ese sistema a analizarlo y diseñarlo pero, ¿Qué es un paradigma? Se tienen varias definiciones "informales":

Un paradigma es "... *una constelación de conceptos, valores, percepciones y prácticas compartidas por una comunidad que se forma una visión particular de la realidad y que es la base de la manera en que la comunidad se organiza a sí misma*" .

En el ámbito de la Ingeniería de Software, los principales paradigmas que predominan, y en los cuales se pensó para construir este sistema, son:

- El Paradigma Estructurado Procedural.
- El Paradigma Orientado a Objetos.

Comparación entre el Paradigma Estructurado y el Orientado a Objetos

Paradigma Estructurado Procedural	Paradigma Orientado a Objetos
El mundo real es representado por entidades lógicas y control de flujo.	El mundo real es representado por objetos que imitan a las entidades externas.
Utiliza abstracción procedural.	Utiliza abstracción de clases y objetos.
Unidad de estructura: sentencia o expresión.	Unidad de estructura: objeto tratado como componente de software.
Los sistemas se construyen basándose en el fundamento de las funciones.	Los sistemas se construyen basándose en el principio de las estructuras de datos (objetos).
Utiliza descomposición funcional.	Utiliza descomposición orientada a objetos.
Estructuras de datos pasivas y tontas.	Estructuras de datos activas e inteligentes (objetos) encapsulan todos los procedimientos pasivos.
En un programa, los datos y los procedimientos están separados.	En un programa, el estado de los objetos (tipos de datos) y el comportamiento (operaciones) están encapsulados.
Los programas son ligados a través del mecanismo de paso de parámetros y el sistema operativo.	Los programas son partes integradas del programa completo. Un programa es una colección de objetos que interactúan.
Los programadores son responsables de llamar los procedimientos activos para el paso de parámetros.	Los objetos activos se comunican con otros pasando mensajes para activar sus operaciones.

El programador debe estar seguro de que el procedimiento trabajará correctamente sobre el tipo de dato al cual se aplica.	El paso de mensajes asegura que se puede tener acceso al estado interno de un objeto solamente si es permitido, así como la encapsulación previene el acceso no autorizado.
Utiliza lenguajes orientados a procedimientos tales como C o Pascal.	Utiliza lenguajes orientados a objetos tales como C++, SmallTalk, Java, etc.

El paradigma orientado a objetos

El Paradigma Orientado a Objetos presenta características que lo hacen idóneo para el análisis, diseño y programación de sistemas.

El mundo real es un lugar complejo en el sentido de que todas las cosas, tangibles e intangibles, son complejas si las analizamos a partir de sus componentes más básicos. Se dice que el Paradigma Orientado a Objetos es más natural que el tradicional, el Paradigma Estructurado, y es cierto en dos sentidos. En un sentido, dicho paradigma es más natural porque nos permite organizar la información de una forma similar para nosotros. En el otro sentido es más natural porque refleja las técnicas de la naturaleza para manejar la complejidad

Para entender mejor lo anterior, veamos la manera, de forma más general, en que la naturaleza le da poder al concepto de objeto.

El bloque básico de construcción en la naturaleza, y de la cual está compuesta toda la variedad de seres vivos que existimos sobre la Tierra, es la célula. Si observamos la estructura interna de la misma en la siguiente Figura, podemos hacer las siguientes conclusiones:

1. La célula está protegida por una membrana que controla el acceso a su interior. De hecho, la membrana encapsula a la célula, protegiendo su forma interna de trabajar de la intromisión externa.
2. La célula contiene formas de manejar información y comportamiento. La mayoría de la información que define su comportamiento se encuentra en el núcleo, en el centro de la célula (DNA y RNA). El comportamiento, por ejemplo la síntesis de proteínas y la conversión de energía, se lleva a cabo en el cuerpo medio de la célula.

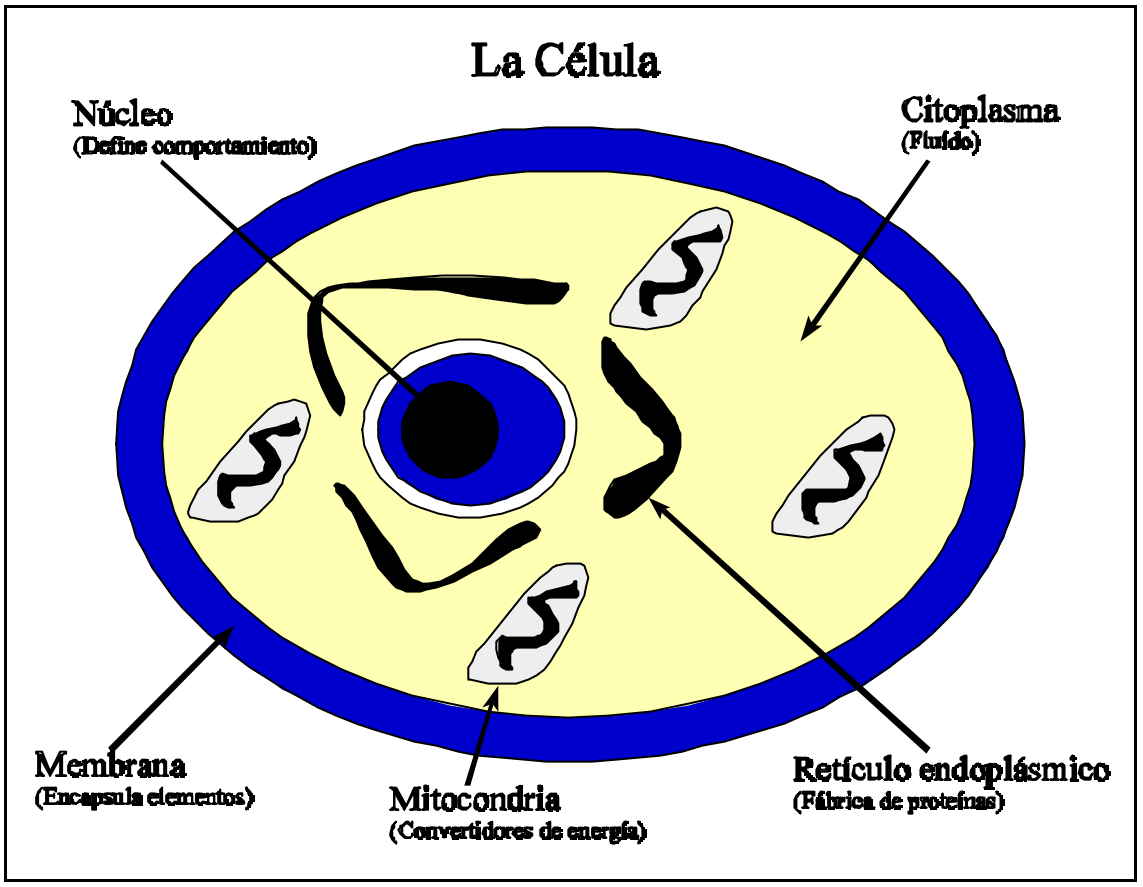


Figura 4 La célula

3. La interacción de las células se realiza a través de la membrana ya que es ésta la que permite intercambiar mensajes químicos con otras células. Una célula puede o no permitir el intercambio del mensaje químico. La membrana representa, entonces, su interfaz con el exterior. Esta comunicación basada en mensajes simplifica enormemente la forma en que las células realizan su función. No es necesario que una célula conozca el funcionamiento interno de otra; lo único que tiene que hacer es enviarle el mensaje apropiado para que la célula receptora ejecute esa acción.

4. Cuando una célula muere, por ejemplo, se liberan sustancias (mensajes) con las que otras células se dan cuenta; entonces una de ellas se divide en dos nuevas células para sustituir a la que murió, regenerándose así los tejidos.

Como podemos apreciar, la interacción entre células está basada en mensajes. Esto quiere decir que cuando una célula quiere afectar el comportamiento de otras células, aquélla les envía un mensaje químico que provoca que éstas modifiquen su comportamiento al realizar cierta acción. Cuando una célula recibe un mensaje

que tiene significado para ella, responde al estímulo de acuerdo con la información que tiene codificada en su núcleo.

En pocas palabras, las células no interactúan entre sí interfiriendo con el DNA de las otras. Las células se ocupan de sus propias actividades y hacen requerimientos de una manera ordenada, en lugar de tratar de controlar la operación interna de las otras células directamente. Esta situación ofrece dos formas diferentes de protección:

1. Protege el interior de cada célula de ser modificado por otras. Cualquier célula viviente es mucho más compleja que la mayoría de los sistemas de cómputo jamás creados. Al proteger el interior de la célula del mundo externo, mantiene su integridad.
2. Protege a otras células al no conocer el manejo interno de dichas células.

La segunda ventaja es un ejemplo vivo de lo que es el ocultamiento de la información. De hecho, las células funcionan tan bien como módulos en sistemas complejos como los seres vivos, gracias a que protegen al resto del sistema del tener que lidiar con la complejidad interna. Cuando no se respeta esta independencia entre células y la puesta en práctica interna, la célula enferma y muere. Veamos el problema con un virus; el virus se interna en la célula y la ataca desde adentro. Dado que el virus está arraigado en la célula, no puede ser destruido por los glóbulos blancos porque destruiría también a la célula.

Así como la célula del ejemplo anterior, de igual manera es de complejo el mundo que rodea al hombre. Típicamente, el ser humano entiende el mundo que lo rodea por la construcción de modelos mentales, de porciones o fragmentos de ese mundo, para tratar de entender las cosas con las que él interactúa. En esencia, este proceso de construcción de modelos es semejante al de diseño de software.

Además, el modelo mental debe ser más simple que el sistema al que está modelando o dicho modelo será inútil. El ser humano debe abstraerse para poder entender el sistema y realizar una migración, de éste, al modelo mental considerando sólo información cuidadosamente seleccionada e ignorando características irrelevantes. Este proceso de *abstracción* es psicológicamente natural y necesario para que nosotros entendamos el mundo que nos rodea.

La Abstracción como herramienta en el Paradigma Orientado a Objetos.

La abstracción es esencial para el funcionamiento de la mente humana, una poderosa herramienta para tratar la complejidad y la llave para diseñar buen software. El *Diseño Orientado a Objetos* (*DOO*) descompone un sistema en objetos, el componente básico del diseño. Este uso de objetos permite la adición de otros mecanismos de abstracción.

Algunos conceptos del Paradigma Orientado a Objetos

De todo lo anterior se puede concluir que el Paradigma Orientado a Objetos ha derivado de los paradigmas anteriores a éste. Así como los métodos de diseño estructurado realizados guían a los desarrolladores que tratan de construir sistemas complejos utilizando algoritmos como sus bloques fundamentales de construcción, similarmente los métodos de diseño orientado a objetos han evolucionado para ayudar a los desarrolladores a explotar el poder de los lenguajes de programación basados en objetos y orientados a objetos, utilizando las clases y objetos como bloques de construcción básicos.

Actualmente el modelo de objetos ha sido influenciado por un número de factores no sólo de la *Programación Orientada a Objetos*, *POO* (*Object Oriented Programming*, *OOP* por sus siglas en inglés). Además, el modelo de objetos ha probado ser un concepto uniforme en las ciencias de la computación, aplicable no sólo a los lenguajes de programación sino también al diseño de interfaces de usuario, bases de datos y arquitectura de computadoras por completo. La razón de ello es, simplemente, que una orientación a objetos nos ayuda a hacer frente a la inherente complejidad de muchos tipos de sistemas.

El Paradigma Orientado a Objetos representa, entonces, un desarrollo evolucionario y no revolucionario. No rompe con los avances del pasado pero se construye sobre éstos. Desafortunadamente, muchos programadores hoy en día están utilizando, formalmente e informalmente, los principios de diseño estructurado por lo que, hasta la fecha, ambos tipos de diseño de sistemas han coexistido.

Debido a que el modelo de objetos deriva de diversas fuentes, ha sido acompañado de una confusión de terminologías, por lo que se tratará de unificar conceptos.

Se define a un objeto como "*una entidad tangible que muestra alguna conducta bien definida*". Un objeto "*es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos*" .

Los objetos tienen una cierta *"integridad"* la cual no deberá ser violada. En particular, un objeto puede solamente cambiar estado, conducta, ser manipulado o estar en relación con otros objetos de manera apropiada a este objeto.

Actualmente, el *Análisis Orientado a Objetos* (*AOO*) va progresando como método de análisis de requisitos por derecho propio y como complemento de otros métodos de análisis. En lugar de examinar un problema mediante el modelo clásico de entrada-proceso-salida (flujo de información) o mediante un modelo derivado exclusivamente de estructuras jerárquicas de información, el AOO introduce varios conceptos nuevos. Estos conceptos nuevos le parecen inusuales a mucha gente, pero son bastante naturales.

Considerando este hecho, Coad y Yourdon escriben: "*El AOO —Análisis Orientado a Objetos— se basa en conceptos que una vez aprendimos en la guardería: objetos y atributos, clases y miembros, todo y parte. Nadie sabe por qué hemos tardado tanto tiempo en aplicar estos conceptos en el análisis y la especificación de los sistemas de información —quizás hemos estado demasiado ocupados 'siguiendo el flujo' durante nuestros análisis estructurados diarios, para ponernos a considerar otras alternativas*" .

Algunos conceptos que se utilizan en el Paradigma Orientado a Objetos son los siguientes:

Una *clase* es una plantilla para objetos múltiples con características similares. Las clases comprenden todas esas características de un conjunto particular de objetos. Cuando se escribe un programa en lenguaje orientado a objetos, no se definen objetos verdaderos sino se definen clases de objetos.

Una *instancia* de un objeto es otro término para un objeto real. Si la clase es la representación general de un objeto, una instancia es su representación concreta. A menudo se utiliza indistintamente la palabra objeto o instancia para referirse, precisamente, a un objeto.

En los lenguajes orientados a objetos, cada clase está compuesta de dos cualidades: *atributos* (estado) y *métodos* (comportamiento o conducta) como se observa en la figura siguiente en la que dichos atributos están en la parte interna de la clase y los métodos se encuentran en la parte externa "cubriendo" los atributos.

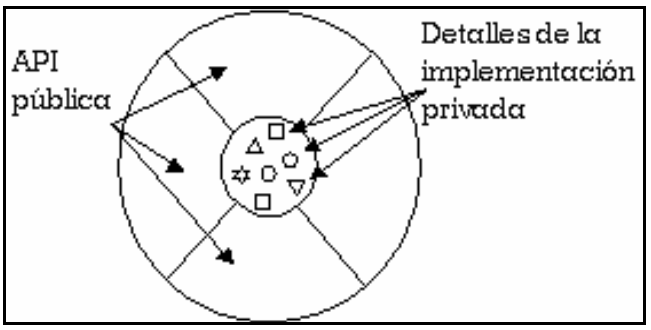


Figura 5 Representación visual de una clase como componente de software

Los atributos son las características individuales que diferencian a un objeto de otro y determinan la apariencia, estado u otras cualidades de ese objeto. Los atributos de un objeto incluyen información sobre su estado.

Los métodos de una clase determinan el comportamiento o conducta que requiere esa clase para que sus instancias puedan cambiar su estado interno o cuando dichas instancias son llamadas para realizar algo por otra clase o instancia. El comportamiento es la única manera en que las instancias pueden hacerse algo a

sí mismas o tener que hacerles algo (Figura 6). Los atributos se encuentran en la parte interna mientras que los métodos se encuentran en la parte externa del objeto.

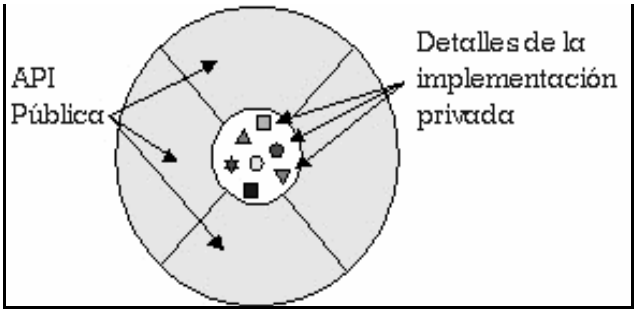


Figura 6 Representación visual de un objeto como componente de software

Para definir el comportamiento de un objeto, se crean métodos, los cuales tienen una apariencia y un comportamiento igual al de las funciones en otros lenguajes de programación, los lenguajes estructurados, pero se definen dentro de una clase. Los métodos no siempre afectan a un solo objeto; los objetos también se comunican entre sí mediante el uso de métodos. Una clase u objeto puede llamar métodos en otra clase u objeto para avisar sobre los cambios en el ambiente o para solicitarle a ese objeto que cambie su estado.

Cualquier cosa que un objeto no sabe, o no puede hacer, es excluida del objeto. Además, las variables del objeto se localizan en el centro o núcleo del objeto. Los métodos rodean y esconden el núcleo del objeto de otros objetos en el programa. Al empaquetamiento de las variables de un objeto con la protección de sus métodos se le llama *encapsulamiento*.

Típicamente, el encapsulamiento es utilizado para esconder detalles de la puesta en práctica no importantes de otros objetos. Entonces, los detalles de la puesta en práctica pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

Esta imagen conceptual de un objeto (un núcleo de variables empaquetadas en una membrana protectora de métodos) es una representación ideal de un objeto y es el ideal por el que los diseñadores de sistemas orientados a objetos luchan. Sin embargo, no lo es todo: a menudo, por razones de eficiencia o la puesta en práctica, un objeto puede querer exponer algunas de sus variables o esconder algunos de sus métodos.

El encapsulamiento de variables y métodos en un componente de software ordenado es, todavía, una simple idea poderosa que provee dos principales beneficios a los desarrolladores de software:

- *Modularidad*, esto es, el código fuente de un objeto puede ser escrito, así como darle mantenimiento, independientemente del código fuente de otros objetos. Así mismo, un objeto puede ser transferido alrededor del sistema sin alterar su estado y conducta.
- *Ocultamiento de la información*, es decir, un objeto tiene una *"interfaz publica"* que otros objetos pueden utilizar para comunicarse con él. Pero el objeto puede mantener información y métodos privados que pueden ser cambiados en cualquier tiempo sin afectar a los otros objetos que dependan de ello.

En las ilustraciones anteriores de una clase (figura 5) y un objeto (figura 6), se observa que son muy similares una de la otra y puede prestarse a una seria confusión. En el mundo real, es obvio que las clases mismas no son los objetos que ellas describen o contienen. Sin embargo, es un poco más difícil diferenciar clases y objetos en software porque los objetos de software son, simplemente, modelos electrónicos de objetos del mundo real o conceptos abstractos en primer lugar. Lo anterior sucede porque mucha gente utiliza el término *"objeto"* irregularmente y lo hace para referirse tanto a clases como instancias.

La clase (figura 5), no está "sombreada" porque representa un *"proyecto"*, general y detallado, de un objeto en lugar de un objeto mismo. En comparación, un objeto (figura 6) está sombreado indicando que ese objeto actualmente existe y puede ser utilizado.

Los objetos proveen el beneficio de la modularidad y el ocultamiento de la información. Las clases proveen el beneficio de la reutilización. Los programadores de software utilizan la misma clase, y por lo tanto el mismo código, una y otra vez para crear muchos objetos.

En las implantaciones orientadas a objetos se percibe un objeto como un paquete de datos y procedimientos que se pueden llevar a cabo con estos datos. Esto encapsula los datos y los procedimientos. La realidad es diferente: los atributos se relacionan al objeto o instancia y los métodos a la clase. ¿Por qué se hace así? Los atributos son variables comunes en cada objeto de una clase y cada uno de ellos puede tener un valor asociado, para cada variable, diferente al que tienen para esa misma variable los demás objetos. Los métodos, por su parte, pertenecen a la clase y no se almacenan en cada objeto, puesto que sería un desperdicio almacenar el mismo procedimiento varias veces y ello va contra el principio de reutilización de código.

Otro concepto muy importante en el Paradigma Orientado a Objetos es el de *herencia*. La herencia es un mecanismo poderoso con el cual se puede definir una clase en términos de otra clase; lo que significa que cuando se escribe una clase, sólo se tiene que especificar la diferencia de esa clase con otra, con lo cual, la herencia dará acceso automático a la información contenida en esa otra clase.

Con la herencia, todas las clases están arregladas dentro de una jerarquía estricta. Cada clase tiene una superclase (la clase superior en la jerarquía) y puede tener una o más subclases (las clases que se encuentran debajo de esa clase en la jerarquía). Se dice que las clases inferiores en la jerarquía, las clases hijas, heredan de las clases más altas, las clases padres.

Las subclases heredan todos los métodos y variables de las superclases. Es decir, en alguna clase, si la superclase define un comportamiento que la clase hija necesita, no se tendrá que redefinir o copiar ese código de la clase padre.

De esta manera, se puede pensar en una jerarquía de clase como la definición de conceptos demasiado abstractos en lo alto de la jerarquía y esas ideas se convierten en algo más concreto conforme se desciende por la cadena de la superclase.

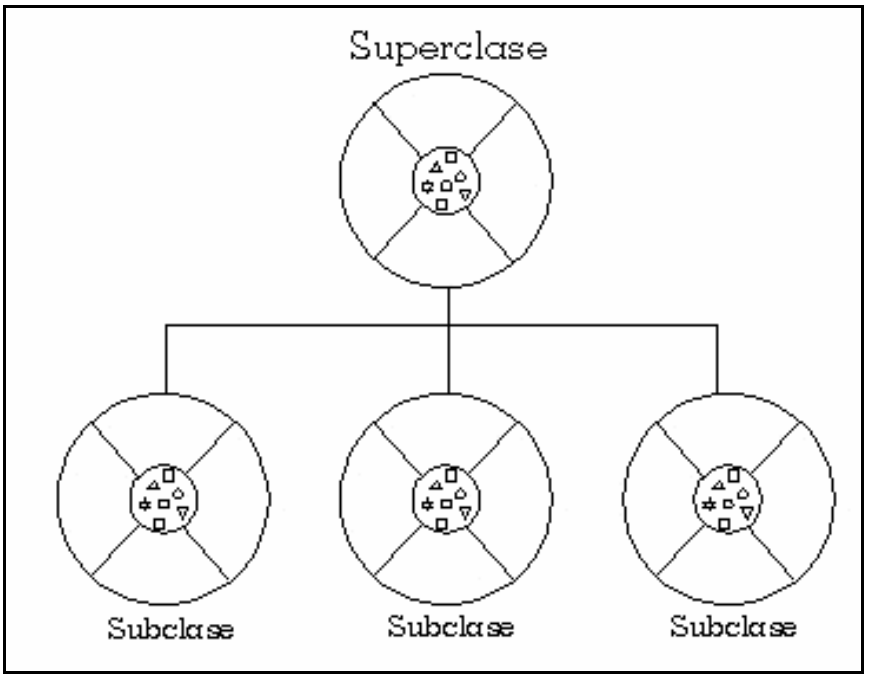


Figura 7 Herencia

Sin embargo, las clases hijas no están limitadas al estado y conducta provistos por sus superclases; pueden agregar variables y métodos además de los que ya heredan de sus clases padres. Las clases hijas pueden, también, sobrescribir los métodos que heredan por implementaciones especializadas para esos métodos.

De igual manera, no hay limitación a un sólo nivel de herencia por lo que se tiene un árbol de herencia en el que se puede heredar varios niveles hacia abajo y mientras más niveles descienda una clase, más especializada será su conducta.

La herencia presenta los siguientes beneficios:

- Las subclases proveen conductas especializadas sobre la base de elementos comunes provistos por la superclase. A través del uso de herencia, los programadores pueden reutilizar el código de la superclase muchas veces.
- Los programadores pueden implementar superclases llamadas clases abstractas que definen conductas *"genéricas"*. Las superclases abstractas definen, y pueden implementar parcialmente la conducta pero gran parte de la clase no está definida ni implementada. Otros programadores concluirán esos detalles con subclases especializadas.

Ventajas del Paradigma Orientado a Objetos

En síntesis, algunas ventajas que presenta el Paradigma Orientado a Objetos son:

- *Reutilización*. Las clases están diseñadas para que se reutilicen en muchos sistemas. Para maximizar la reutilización, las clases se construyen de manera que se puedan adaptar a los otros sistemas. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir el software.

- *Estabilidad*. Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips se hacen estables.

- *El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel*. El encapsulamiento oculta los detalles y hace que las clases complejas sean fáciles de utilizar.

- *Se construyen clases cada vez más complejas*. Se construyen clases a partir de otras clases, las cuales a su vez se integran mediante clases. Esto permite construir componentes de software complejos, que a su vez se convierten en bloques de construcción de software más complejo.

- *Calidad*. Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido verificados y pulidos varias veces.

- *Un diseño más rápido*. Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están construidos de modo que se pueden adaptar para un diseño particular.

- *Integridad*. Las estructuras de datos (los objetos) sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-servidor y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.

- *Mantenimiento más sencillo*. El programador encargado del mantenimiento cambia un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás.

- *Una interfaz de pantalla sugestiva para el usuario*. Hay que utilizar una interfaz de usuario gráfica de modo que el usuario apunte a iconos o elementos de un menú desplegado, relacionados con los objetos. En determinadas ocasiones, el usuario puede ver un objeto en la pantalla. Ver y apuntar es más fácil que recordar y escribir.

- *Independencia del diseño*. Las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software. Utilizan solicitudes y respuestas con formato estándar. Esto les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de red, interfaces de usuario gráficas, etc. El creador del software no tiene que preocuparse por el ambiente o esperar a que éste se especifique.

- *Interacción*. El software de varios proveedores puede funcionar como conjunto. Un proveedor utiliza clases de otros. Existe una forma estándar de localizar clases e interactuar con ellas. El software desarrollado de manera independiente en lugares ajenos debe poder funcionar en forma conjunta y aparecer como una sola unidad ante el usuario.

- *Computación Cliente-Servidor*. En los sistemas cliente-servidor, las clases en el software cliente deben enviar solicitudes a las clases en el software servidor y recibir respuestas. Una clase servidor puede ser utilizada por clientes diferentes. Estos clientes sólo pueden tener acceso a los datos del servidor a través de los métodos de la clase. Por lo tanto los datos están protegidos contra su corrupción.

- *Computación de distribución masiva*. Las redes a nivel mundial utilizarán directorios de software de objetos accesibles. El diseño orientado a objetos es la clave para la computación de distribución masiva. Las clases de una máquina interactúan con las de algún otro lugar sin saber donde residen tales clases. Ellas reciben y envían mensajes orientados a objetos en formato estándar.

- *Mayor nivel de automatización de las bases de datos*. Las estructuras de datos (los objetos) en las bases de datos orientadas a objetos están ligadas a métodos que llevan a cabo acciones automáticas. Una base de datos OO tiene integrada una *inteligencia*, en forma de métodos, en tanto que una base de datos relacional básica carece de ello.

- *Migración*. Las aplicaciones ya existentes, sean orientadas a objetos o no, pueden preservarse si se ajustan a un contenedor orientado a objetos, de

modo que la comunicación con ella sea a través de mensajes estándar orientados a objetos.

- *Mejores herramientas CASE*. Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) utilizarán las técnicas gráficas para el diseño de las clases y de la interacción entre ellas, para el uso de los objetos existentes adaptados a nuevas aplicaciones. Las herramientas deben facilitar el modelado en términos de eventos, formas de activación, estados de objetos, etc. Las herramientas OO del CASE deben generar un código tan pronto se definan las clases y permitir al diseñador utilizar y probar los métodos recién creados. Las herramientas se deben diseñar de manera que apoyen el máximo de creatividad y una continua afinación del diseño durante la construcción.

Dentro del Paradigma Orientado a Objetos existen muchos métodos de modelado de sistemas, de los cuales el predominante es *OMT* (*Object Modeling Technique*, *Técnica de Modelado de Objetos*) pero en 1995 surgió no un método sino un lenguaje de modelado de sistemas llamado *UML* (*Unified Modeling Language*, *Lenguaje Unificado de Modelado*).

3. -Lenguaje Unificado de Modelado (UML)

UML es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

UML sirve para hacer modelos que permitan:

- Visualizar cómo es un sistema o cómo queremos que sea.
- Especificar la estructura o comportamiento de un sistema.
- Hacer una plantilla que guíe la construcción de los sistemas.
- Documentar las decisiones que se han tomado.

Con UML se capturan las características estáticas y dinámicas de un sistema, que en conjunto presentan los objetos importantes de un sistema, así como su interacción y su comportamiento a través del tiempo.

UML prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan.

UML ofrece nueve diagramas en los cuales modelar sistemas.

- Diagramas de Casos de Uso para modelar los procesos de negocio.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

Los beneficios que UML aporta son:

- Mejores tiempos totales de desarrollo (de 50 % o más).
- Mejor calidad.
- Mejor soporte a la planeación y al control de proyectos.
- Mayor independencia del personal de desarrollo.
- Mayor soporte al cambio organizacional, comercial y tecnológico.
- Alta reutilización.
- Minimización de costos.

UML soporta prácticamente todas las fases de un ciclo de desarrollo de un sistema: Recopilación de requerimientos, Análisis, Diseño de sistemas, Pruebas, Puesta en práctica o Instrumentación, en Reingeniería y en cualquier actividad de desarrollo que sea susceptible de ser modelada.

Cada diagrama puede ser usado con énfasis distinto en cada fase de desarrollo. Un diagrama cualquiera en una fase de análisis tendrá un énfasis lógico y mientras más se acerque al diseño y a la puesta en práctica, mayor será su énfasis físico y tecnológico.

El lenguaje es muy amplio y abarca muchas de las etapas del ciclo de vida del software, sin embargo este documento solo abarca los conceptos que se utilizarán para el desarrollo del proyecto.

Vista estática.

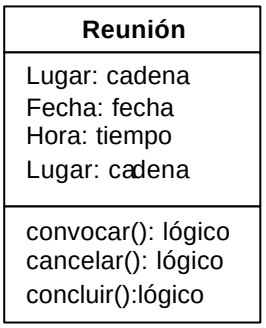
La vista estática presenta y define los objetos que constituyen el sistema. Estos objetos reflejan conceptos del mundo que el sistema modelará, incluyendo objetos del mundo real, conceptos abstractos, conceptos de computo, etc.

En la vista estática se definen los elementos que otras vistas usan para describir procesos. Esta vista captura la estructura de los objetos, sus propiedades que están representadas por datos, así como la organización de las operaciones que se realizan sobre esos datos.

Los conceptos base de esta vista son los clasificadores y sus operaciones. Un clasificador es un elemento del lenguaje usado para descripción. Desde el punto de vista del paradigma orientado a objetos se tienen los clasificadores clase, interfase y tipo de dato para descripción de entidades que participarán en el sistema. Los conceptos relacionados con interacciones y comportamiento son representados con clasificadores como los casos de uso y las señales.

Clase: Una clase es una forma de representar objetos del mundo real, conceptos de negocios, conceptos de computación, etc. En la que se especifican los elementos que la definen: atributos y operaciones.

Por ejemplo, una reunión esta definida por sus atributos que son la hora y fecha a la que esta convocada, el lugar en el que se realizara y el tema a tratar. Las operaciones que una reunión puede realizar son ser convocada, cancelada, llevada a cabo, etc. La figura siguiente presenta la clase Reunión usando la notación para definición de clases.



Nombre de la case: identifica de manera única a esta clase.

Atributos: representan las características de la clase, los valores que tomen estos atributos en cierto tiempo define el estado del objeto en ese momento. Su notación es

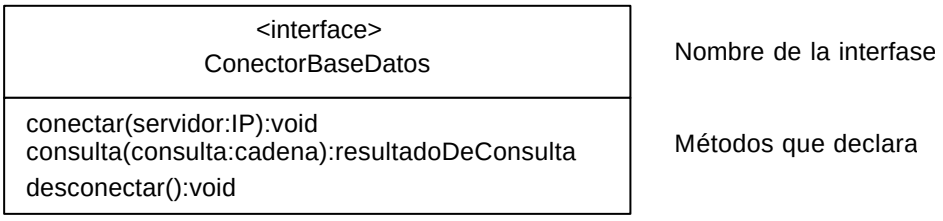
nombreDelAtributo: tipoDeDato

Operaciones: son las acciones que puede realizar la clase, puede implicar la modificación o uso de sus propiedades. Su notación es:

nombreDeLaOperacion(parametro1:tipo,parametro2:tipo,...):tipoDeDatoDeRegreso

Interfaz: Es un conjunto de operaciones que confieren cierto comportamiento a una clase. Una clase que implementa una interfaz esta obligada a cumplir (implementando) las operaciones definidas por esta. Así cualquier observador externo a la clase sabe que puede utilizar las acciones que una interfaz define a través de una clase que lo implementa.

Por ejemplo una interfaz de conexión a una base de datos proporcionara la definición de las acciones que un clase debe cumplir para servir como conector, tales como establecer una conexión, realizar una consulta, desconectarse. Al momento de la implementación la clase que lo haga deberá proporcionar dichos métodos. La siguiente figura presenta la definición de una interfaz de conexión a base de datos utilizando la notación para definición de interfases:



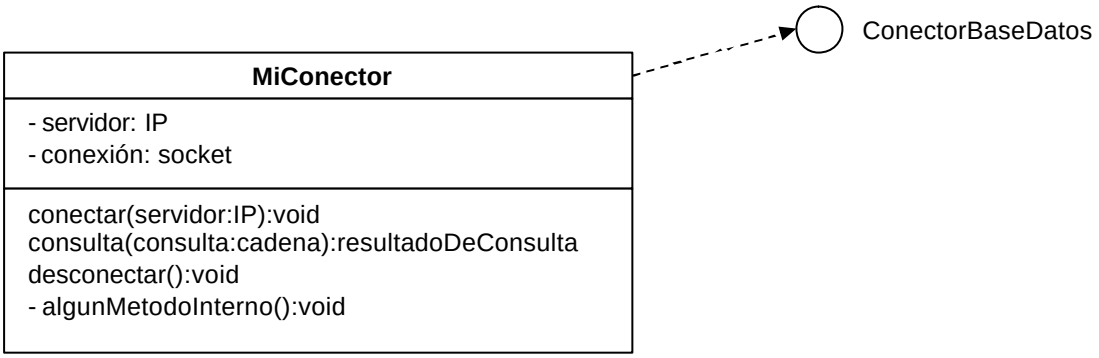
Nombre de la clase: identifica de manera única la interfase. Se usa la palabra reservada <interface>.

Métodos que declara: es el listado de métodos que la implementación debe de cumplir, respetando estrictamente el nombre, tipo, orden y número de argumentos y el tipo de dato de regreso

nombreDeLaOperacion(parametro1:tipo,parametro2:tipo,...):tipoDeDatoDeRegreso

Una interfaz no define atributos porque simplemente denota comportamiento, los atributos o detalles necesarios para presentar dicho comportamiento son agregados por la clase que implementa la interfaz.

Una clase que implemente una interfaz de conexión a base de datos debe tener un lugar para guardar la dirección del servidor al que se esta conectando, un socket para mantener la conexión al servidor, etc. Estos atributos son parte de la clase y son necesarios para cumplir con los métodos definidos por la interfaz.

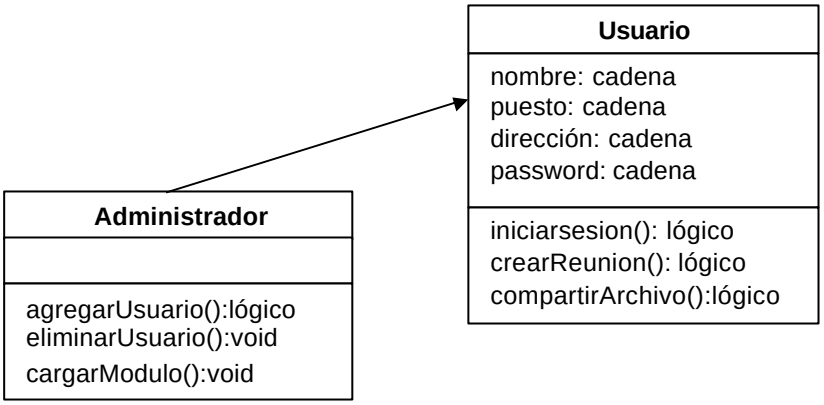


El circulo es la notación simplificada de una interfaz, la flecha punteada representa la operación *"realización"* que sirve para denotar cuando una clase implementa una interfaz. Como se puede ver en la figura anterior, la clase que implementa puede agregar todos los atributos y métodos necesarios para cumplir con la interfaz, normalmente estos elementos son especificados como privados. El carácter privado se denota con el símbolo " – ".

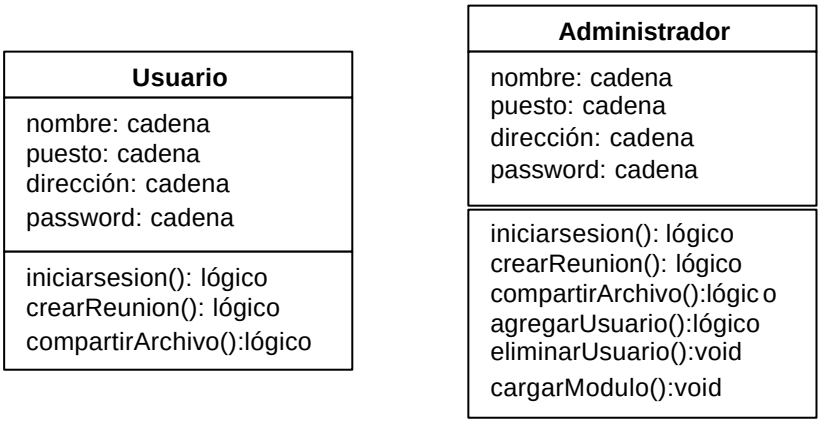
Herencia

La herencia es un mecanismo de propagación de características entre clasificadores, se usa herencia cuando se quiere que un clasificador presente las mismas características de otro mas nuevos elementos. Este mecanismo permite reducir la especificación de clasificadores basándose en otros. Traduciendo lo anterior a líneas de código: una clase basada en otra no necesita rescribir las líneas que ya tiene la clase de la que hereda.

Se puede crear una clase Usuario que representará un usuario del sistema, en ella se guardaran los datos y acciones básicas, un Administrador del sistema es un Usuario al que se le han agregado acciones, para no tener que redefinir los atributos de un usuario en la clase Administrador, se hace una relación de herencia.



La herencia se representa por medio de la operación de generalización, gráficamente es una flecha sólida que parte de la clase que hereda (llamada clase hija) hacia la clase heredada (clase padre o superclase). En la figura anterior, la clase hija Administrador hereda las características de la clase padre Usuario. Si esto no se hiciera por medio de herencia tendríamos el siguiente esquema, que como se puede ver es redundante.



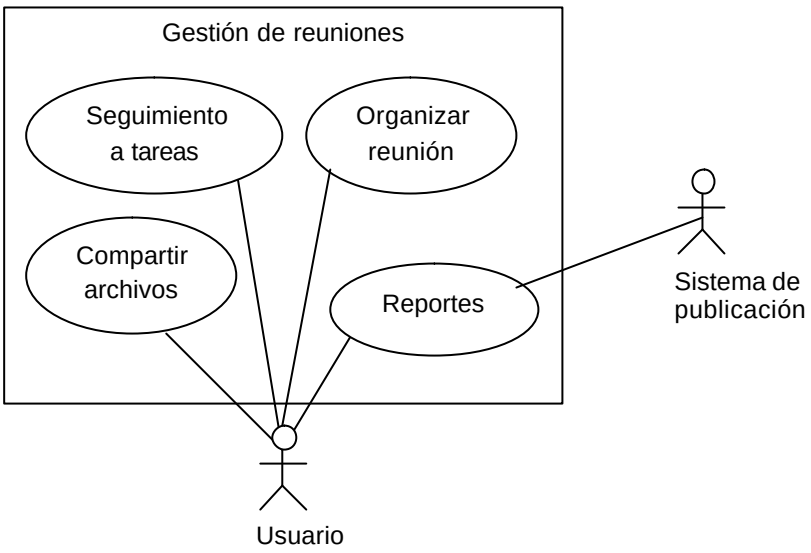
Vista de casos de uso.

El comportamiento que los clasificadores presentan al usuario es modelado a través de esta vista. La funcionalidad del sistema se separa en acciones que los entes que interactúan con el sistema, en adelante actores, pueden realizar.

Un actor es la abstracción de un usuario del sistema, sin restringir esta definición a personas, pues el sistema puede ser *"usado"* por otros sistemas que se comunican a través de protocolos específicos.

Las funcionalidades que el sistema presenta a los actores se representa con casos de uso, que es una unidad de funcionalidad visible desde fuera del sistema. Un caso de uso engloba todo un comportamiento del sistema incluyendo sus distintas variantes, manejo de errores, casos excepcionales, etc.

En este caso, se tienen algunos ejemplos de casos de uso básicos, como se puede ver muchos casos de uso representan requerimientos del sistema, otros son acciones derivadas de dichos requerimientos.



Un caso de uso se representa gráficamente por un elipse en donde se anota el nombre del caso de uso, los actores son representados por el dibujo de una persona. Los casos de uso se pueden agrupar en sistemas que son representados por rectángulo que engloban los casos de uso.

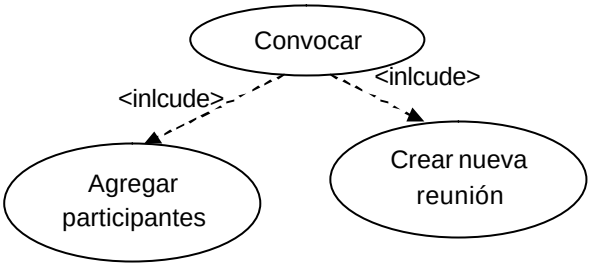
La figura anterior puede ser el sistema (simplificado) con los casos de uso básicos, además de dos actores, uno representando a un usuario del sistema que será una persona física y el otro es un hipotético sistema de publicación para presentar los reportes.

El usuario puede ver todas las funcionalidades del sistema, pues tiene una línea que lo une con cada uno de los casos de uso. Sin embargo el sistema de publicación solo puede ver la funcionalidad de generación de reportes. De lo anterior se desprende que el caso de uso Reportes debe de tener dos formas de comportamiento, uno para presentar reportes destinado a una persona, como HTML, PDF, etc. El otro comportamiento debe de generar los reportes en el formato acordado para la comunicación con el sistema de publicación por ejemplo XML.

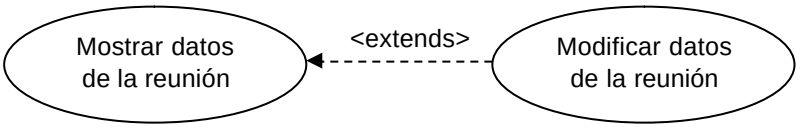
Los casos de uso son unidades de comportamiento independientes, ortogonales, una de la otra, sin embargo la descripción de un caso de uso puede estar compuesta por casos de uso mas simples.

Cuando un caso de uso incluye el comportamiento de caos de uso mas simple se tiene la relación de inclusión, la cual indica que el nuevo caso de uso simplemente

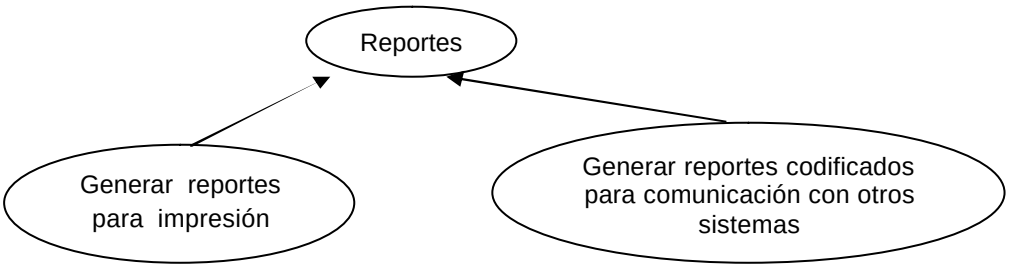
incluye a otros como fragmentos de un comportamiento global. Gráficamente se representa por una flecha punteada con la palabra reservada <include> desde el nuevo caso de uso hacia los que incluye.



Si un caso de uso es muy parecido a otro pero con mas funcionalidad, se puede usar una relación de extensión, la cual indica que el nuevo caso de uso presenta el mismo comportamiento que el caso base además de extensiones. Su símbolo es una flecha punteada del nuevo caso al caso de uso base, con la palabra <extends>.



Cuando se tiene un caso de uso muy general se pude hacer mas especifico utilizando el mecanismo de herencia, así los casos de uso hijos son versiones mas especificas del caso de uso padre. Gráficamente es una flecha sólida desde el caso hijo al padre.



Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

4.- Bases de Datos

Tan importante como el sistema, son los datos que éste va a manipular. Dichos datos nos permitirán el buen desempeño de nuestro sistema, ya que servirán como entrada y salida del mismo. De igual manera, uno de los objetivos del sistema es la explotación de los datos. El papel del ingeniero es el de definir la información que va a contener la base de datos, los tipos de peticiones que se podrán procesar, la manera en que se accederá a los datos y la capacidad de la base de datos. El análisis y el diseño de los datos son actividades fundamentales de la ingeniería de software.

Por todo esto, es necesario entrar en detalles de conceptos propios de bases de datos.

Una *base de datos* es un conjunto de datos interrelacionados entre si. Comúnmente la base de datos contiene información interrelacionada y referente a una misma entidad o empresa.

Diseño de bases de datos

El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo, se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios.

Debido a la creciente aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información en la mayoría de las organizaciones.

Las bases de datos son componentes esenciales de los sistemas de información, usadas rutinariamente en todas las computadoras. El diseño de bases de datos se ha convertido en una actividad popular, desarrollada no sólo por profesionales sino también por no especialistas. A finales de la década de 1960, cuando las bases de datos entraron por primera vez en el mercado del software, los diseñadores de bases de datos actuaban como artesanos, con herramientas muy primitivas: diagramas de bloques y estructuras de registros eran los formatos comunes para las especificaciones, y el diseño de bases de datos se confundía frecuentemente con la implantación de las bases de datos.

Esta situación ahora ha cambiado: los métodos y modelos de diseño de bases de datos han evolucionado paralelamente con el progreso de la tecnología en los sistemas de bases de datos. Se ha entrado en la era de los sistemas relacionales de bases de datos, que ofrecen poderosos lenguajes de consulta, herramientas para el desarrollo de aplicaciones e interfaces amables con los usuarios. La

tecnología de bases de datos cuenta ya con un marco teórico, que incluye la teoría relacional de datos, procesamiento y optimización de consultas, control de concurrencia, gestión de transacciones y recuperación, etc.

Desafortunadamente, las metodologías de diseño de bases de datos no son muy populares; la mayoría de las organizaciones y de los diseñadores individuales confía muy poco en las metodologías para llevar a cabo el diseño y esto se considera, con frecuencia, una de las principales causas de fracaso en el desarrollo de los sistemas de información. Debido a la falta de enfoques estructurados para el diseño de bases de datos, a menudo se subestiman el tiempo o los recursos necesarios para un proyecto de bases de datos, las bases de datos son inadecuadas o ineficientes en relación a las demandas de la aplicación, la documentación es limitada y el mantenimiento es difícil.

Metodología de diseño de bases de datos

El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles. La complejidad se controla mejor si se descompone el problema en sub problemas y se resuelve cada uno de estos sub problemas independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

El diseño conceptual parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos. Un *esquema conceptual* es una descripción de alto nivel de la estructura de la base de datos, independientemente del Sistema de Gestión de Bases de Datos (SGBD) que se vaya a utilizar para manipularla.

Un *modelo conceptual* es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar esta información.

El diseño lógico parte del esquema conceptual y da como resultado un esquema lógico. Un *esquema lógico* es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de SGBD. Un *modelo lógico* es un lenguaje usado para especificar esquemas lógicos (modelo relacional, modelo de red, etc.). El diseño lógico depende del tipo de SGBD que se vaya a utilizar, no depende del producto concreto.

El diseño físico parte del esquema lógico y da como resultado un esquema físico. Un *esquema físico* es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del SGBD concreto y el esquema físico se expresa mediante su lenguaje de definición de datos.

Modelos de datos

Un *modelo de datos* es una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los *modelos conceptuales* se utilizan para representar la realidad a un alto nivel de abstracción. Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los *modelos lógicos*, las descripciones de los datos tienen una correspondencia sencilla con la estructura física de la base de datos.

En el diseño de bases de datos se usan primero los modelos conceptuales para lograr una descripción de alto nivel de la realidad, y luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad. Un *esquema* es un conjunto de representaciones lingüísticas o gráficas que describen la estructura de los datos de interés.

Los modelos conceptuales deben ser buenas herramientas para representar la realidad, por lo que deben poseer las siguientes cualidades:

- *Expresividad*: deben tener suficientes conceptos para expresar perfectamente la realidad.
- *Simplicidad*: deben ser simples para que los esquemas sean fáciles de entender.
- *Minimalidad*: cada concepto debe tener un significado distinto.
- *Formalidad*: todos los conceptos deben tener una interpretación única, precisa y bien definida.

El diseño de bases de datos se descompone en tres etapas: diseño conceptual, diseño lógico y diseño físico. El diseño conceptual es el proceso por el cual se construye un modelo de la información que se utiliza en una empresa u organización, independientemente del SGBD que se vaya a utilizar para implementar el sistema y de los equipos informáticos o cualquier otra consideración física.

Un modelo conceptual es un conjunto de conceptos que permiten describir la realidad mediante representaciones lingüísticas y gráficas. Los modelos conceptuales deben poseer una serie de propiedades: expresividad, simplicidad, minimalidad y formalidad.

El modelo conceptual más utilizado es el modelo entidad-relación, que posee los siguientes conceptos: entidades, relaciones, atributos, dominios de atributos, identificadores y jerarquías de generalización.

En la metodología del diseño conceptual se construye un esquema conceptual local para cada vista de cada usuario o grupo de usuarios. En el diseño lógico se

obtiene un esquema lógico local para cada esquema conceptual local. Estos esquemas lógicos se integran después para formar un esquema lógico global que represente todas las vistas de los distintos usuarios de la empresa. Por último, en el diseño físico, se construye la implementación de la base de datos sobre un SGBD determinado. Ya que este diseño debe adaptarse al SGBD, es posible que haya que introducir cambios en el esquema lógico para mejorar las prestaciones a nivel físico.

Cada vista de usuario comprende los datos que un usuario maneja para llevar a cabo una determinada tarea. Normalmente, estas vistas corresponden a las distintas áreas funcionales de la empresa, y se pueden identificar examinando los diagramas de flujo de datos o entrevistando a los usuarios, examinando los procedimientos, informes y formularios, y observando el funcionamiento de la empresa.

Cada esquema conceptual local está formado por entidades, relaciones, atributos, dominios de atributos, identificadores y puede haber también jerarquías de generalización. Además, estos esquemas se completan documentándolos en el diccionario de datos.

Metodología de diseño conceptual

El primer paso en el diseño de una base de datos es la producción del esquema conceptual. Normalmente, se construyen varios esquemas conceptuales, cada uno para representar las distintas visiones que los usuarios tienen de la información.

Estas visiones de la información, denominadas *vistas*, se pueden identificar de varias formas. Una opción consiste en examinar los diagramas de flujo de datos, que se pueden haber producido previamente, para identificar cada una de las áreas funcionales. La otra opción consiste en entrevistar a los usuarios, examinar los procedimientos, los informes y los formularios, etc.

A los esquemas conceptuales correspondientes a cada vista de usuario se les denomina *esquemas conceptuales locales*. Cada uno de estos esquemas se compone de entidades, relaciones, atributos, dominios de atributos e identificadores. El esquema conceptual también tendrá una documentación, que se irá produciendo durante su desarrollo. Las tareas a realizar en el diseño conceptual son las siguientes:

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).

7. Dibujar el diagrama entidad-relación.
8. Revisar el esquema conceptual local con el usuario.

1. Identificar las entidades
En primer lugar hay que definir los principales objetos que interesan al usuario. Estos objetos serán las entidades. Una forma de identificar las entidades es examinar las especificaciones de requisitos de usuario. En estas especificaciones se buscan los nombres o los sintagmas nominales que se mencionan (por ejemplo: nombre de la persona, titulo de la reunión, dirección de la empresa, fecha del acuerdo, etc.). También se buscan objetos importantes como personas, lugares o conceptos de interés, excluyendo aquellos nombres que sólo son propiedades de otros objetos. Por ejemplo, se pueden agrupar el nombre y el puesto en una entidad denominada *persona*, y agrupar titulo, hora, lugar, etc en otra entidad denominada *reunión*.

Otra forma de identificar las entidades es buscar aquellos objetos que existen por sí mismos. Por ejemplo, *persona* es una entidad porque las personas existen, sepamos o no sus nombres, direcciones y teléfonos. Siempre que sea posible, el usuario debe colaborar en la identificación de las entidades.

A veces, es difícil identificar las entidades por la forma en que aparecen en las especificaciones de requisitos. Los usuarios, a veces, hablan utilizando ejemplos o analogías. En lugar de hablar de personas en general, hablan de personas concretas, o bien, hablan de los puestos que ocupan esas personas.

Para confundir aún más, los usuarios usan, muchas veces, sinónimos y homónimos. Dos palabras son sinónimos cuando tienen el mismo significado. Los homónimos ocurren cuando la misma palabra puede tener distintos significados dependiendo del contexto.

No siempre es obvio saber si un objeto es una entidad, una relación o un atributo. Los diseñadores de bases de datos deben tener una visión selectiva y clasificar las cosas que observan dentro del contexto de la empresa u organización. A partir de unas especificaciones de usuario es posible que no se pueda deducir un conjunto único de entidades, pero después de varias iteraciones del proceso de análisis, se llegará a obtener un conjunto de entidades que sean adecuadas para el sistema que se ha de construir.

Conforme se van identificando las entidades, se les dan nombres que tengan un significado y que sean obvias para el usuario. Los nombres de las entidades y sus descripciones se anotan en el diccionario de datos. Cuando sea posible, se debe anotar también el número aproximado de ocurrencias de cada entidad. Si una entidad se conoce por varios nombres, éstos se deben anotar en el diccionario de datos como alias o sinónimos.

2. Identificar las relaciones
Una vez definidas las entidades, se deben definir las relaciones existentes entre ellas. Del mismo modo que para identificar las entidades se buscaban nombres en las especificaciones de requisitos, para identificar las relaciones se suelen buscar las expresiones verbales (por ejemplo: reunión tiene puntos, persona planea reunión, etc.). Si las especificaciones de requisitos reflejan estas relaciones es porque son importantes para la empresa y, por lo tanto, se deben reflejar en el esquema conceptual.

La mayoría de las relaciones son binarias (entre dos entidades), pero no hay que olvidar que también puede haber relaciones en las que participen más de dos entidades, así como relaciones recursivas. Es muy importante repasar las especificaciones para comprobar que todas las relaciones, explícitas o implícitas, se han encontrado. Si se tienen pocas entidades, se puede comprobar por parejas si hay alguna relación entre ellas. De todos modos, las relaciones que no se identifican ahora se suelen encontrar cuando se valida el esquema con las transacciones que debe soportar.

Una vez identificadas todas las relaciones, hay que determinar la cardinalidad mínima y máxima con la que participa cada entidad en cada una de ellas. De este modo, el esquema representa de un modo más explícito la semántica de las relaciones. La cardinalidad es un tipo de restricción que se utiliza para comprobar y mantener la calidad de los datos. Estas restricciones son aserciones sobre las entidades que se pueden aplicar cuando se actualiza la base de datos para determinar si las actualizaciones violan o no las reglas establecidas sobre la semántica de los datos.

Conforme se van identificando las relaciones, se les van asignando nombres que tengan significado para el usuario. En el diccionario de datos se anotan los nombres de las relaciones, su descripción y las cardinalidades con las que participan las entidades en ellas.

3. Identificar los atributos y asociarlos a entidades y relaciones
Al igual que con las entidades, se buscan nombres en las especificaciones de requisitos. Son atributos los nombres que identifican propiedades, cualidades, identificadores o características de entidades o relaciones.

Lo más sencillo es preguntarse, para cada entidad y cada relación, ¿qué información se quiere saber de ...? La respuesta a esta pregunta se debe encontrar en las especificaciones de requisitos. Pero, en ocasiones, será necesario preguntar a los usuarios para que aclaren los requisitos.

Desgraciadamente, los usuarios pueden dar respuestas a esta pregunta que también contengan otros conceptos, por lo que hay que considerar sus respuestas con mucho cuidado. Al identificar los atributos, hay que tener en cuenta si son simples o compuestos.

Por ejemplo, el atributo *dirección* puede ser simple, teniendo la dirección completa como un solo valor: `San Rafael 45, Almazora'; o puede ser un atributo compuesto, formado por la *calle* (`San Rafael'), el *número* (`45') y la *población* (`Almazora'). El escoger entre atributo simple o compuesto depende de los requisitos del usuario. Si el usuario no necesita acceder a cada uno de los componentes de la dirección por separado, se puede representar como un atributo simple. Pero si el usuario quiere acceder a los componentes de forma individual, entonces se debe representar como un atributo compuesto.

También se deben identificar los atributos derivados o calculados, que son aquellos cuyo valor se puede calcular a partir de los valores de otros atributos. Por ejemplo, el número de personas en una reunión, la edad de las personas, etc.

Algunos diseñadores no representan los atributos derivados en los esquemas conceptuales. Si se hace, se debe indicar claramente que el atributo es derivado y a partir de qué atributos se obtiene su valor. Donde hay que considerar los atributos derivados es en el diseño físico. Cuando se están identificando los atributos, se puede descubrir alguna entidad que no se ha identificado previamente, por lo que hay que volver al principio introduciendo esta entidad y viendo si se relaciona con otras entidades.

Es muy útil elaborar una lista de atributos e ir eliminándolos de la lista conforme se vayan asociando a una entidad o relación. De este modo, uno se puede asegurar de que cada atributo se asocia a una sola entidad o relación, y que cuando la lista se ha acabado, se han asociado todos los atributos. Hay que tener mucho cuidado cuando parece que un mismo atributo se debe asociar a varias entidades. Esto puede ser por una de las siguientes causas:

- Se han identificado varias entidades, como *administrador*, usuario y *contacto*, cuando, de hecho, pueden representarse como una sola entidad denominada *persona*. En este caso, se puede escoger entre introducir una jerarquía de generalización, o dejar las entidades que representan cada uno de los roles de la persona.
- Se ha identificado una relación entre entidades. En este caso, se debe asociar el atributo a una sola de las entidades y hay que asegurarse de que la relación ya se había identificado previamente. Si no es así, se debe actualizar la documentación para recoger la nueva relación.

Conforme se van identificando los atributos, se les asignan nombres que tengan significado para el usuario. De cada atributo se debe anotar la siguiente información:

- Nombre y descripción del atributo.
- Alias o sinónimos por los que se conoce al atributo.
- Tipo de dato y longitud.
- Valores por defecto del atributo (si se especifican).

- Si el atributo siempre va a tener un valor (si admite o no nulos).
- Si el atributo es compuesto y, en su caso, qué atributos simples lo forman.
- Si el atributo es derivado y, en su caso, cómo se calcula su valor.
- Si el atributo es multievaluado.

4. Determinar los dominios de los atributos
El dominio de un atributo es el conjunto de valores que puede tomar el atributo. Por ejemplo el dominio de los números de teléfono y los números de fax son las tiras de 9 dígitos.

Un esquema conceptual está completo si incluye los dominios de cada atributo: los valores permitidos para cada atributo, su tamaño y su formato. También se puede incluir información adicional sobre los dominios como, por ejemplo, las operaciones que se pueden realizar sobre cada atributo, qué atributos pueden compararse entre sí o qué atributos pueden combinarse con otros. Aunque sería muy interesante que el sistema final respetara todas estas indicaciones sobre los dominios, esto es todavía una línea abierta de investigación. Toda la información sobre los dominios se debe anotar también en el diccionario de datos.

5. Determinar los identificadores
Cada entidad tiene al menos un identificador. En este paso, se trata de encontrar todos los identificadores de cada una de las entidades. Los identificadores pueden ser simples o compuestos. De cada entidad se escogerá uno de los identificadores como clave primaria en la fase del diseño lógico.

Cuando se determinan los identificadores es fácil darse cuenta de si una entidad es fuerte o débil. Si una entidad tiene al menos un identificador, es *fuerte* (otras denominaciones son *padre*, *propietaria* o *dominante*). Si una entidad no tiene atributos que le sirvan de identificador, es *débil* (otras denominaciones son *hijo*, *dependiente* o *subordinada*). Todos los identificadores de las entidades se deben anotar en el diccionario de datos.

6. Determinar las jerarquías de generalización

En este paso hay que observar las entidades que se han identificado hasta el momento. Hay que ver si es necesario reflejar las diferencias entre distintas ocurrencias de una entidad, con lo que surgirán nuevas subentidades de esta entidad genérica; o bien, si hay entidades que tienen características en común y que realmente son subentidades de una nueva entidad genérica. En cada jerarquía hay que determinar si es total o parcial y exclusiva o superpuesta.

7. Dibujar el diagrama entidad-relación
Una vez identificados todos los conceptos, se puede dibujar el diagrama entidad-relación correspondiente a una de las vistas de los usuarios. Se obtiene así un esquema conceptual local.

8. Revisar el esquema conceptual local con el usuario

Antes de dar por finalizada la fase del diseño conceptual, se debe revisar el esquema conceptual local con el usuario. Este esquema está formado por el diagrama entidad-relación y toda la documentación que describe el esquema. Si se encuentra alguna anomalía, hay que corregirla haciendo los cambios oportunos, por lo que posiblemente haya que repetir alguno de los pasos anteriores. Este proceso debe repetirse hasta que se esté seguro de que el esquema conceptual es una fiel representación de la parte de la empresa que se está tratando de modelar.

El modelo entidad-relación

El modelo entidad-relación es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976. El modelo entidad-relación está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.

Originalmente, el modelo entidad-relación sólo incluía los conceptos de entidad, relación y atributo. Más tarde, se añadieron otros conceptos, como los atributos compuestos y las jerarquías de generalización, en lo que se ha denominado *modelo entidad-relación extendido*.

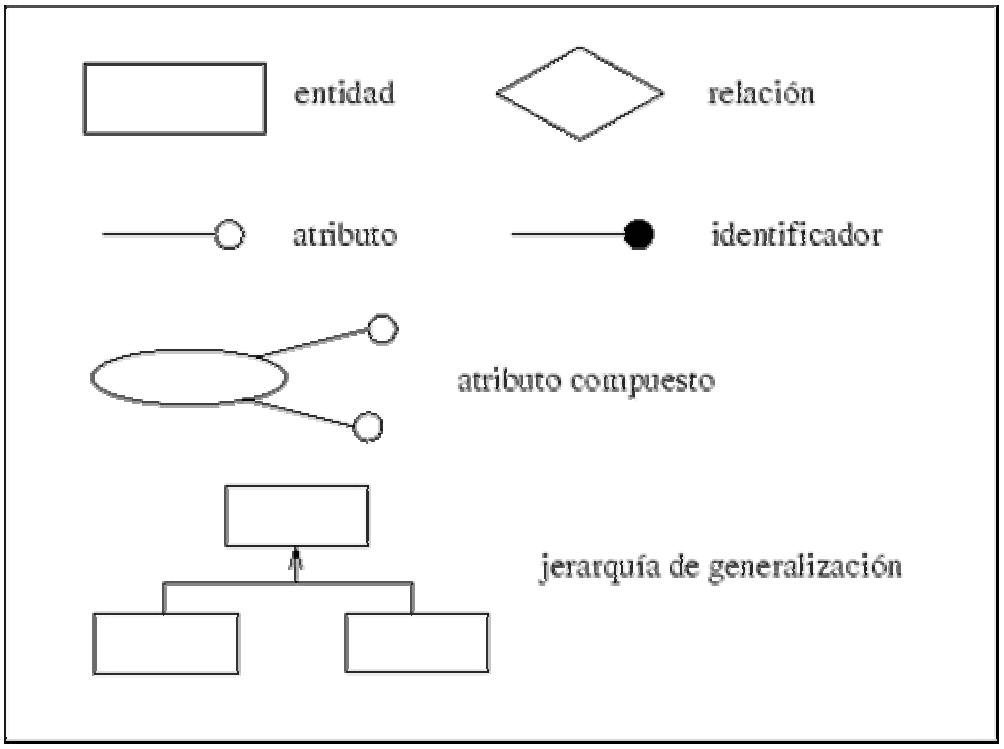


Figura 8 Conceptos del modelo entidad-relación extendido

Entidad
Cualquier tipo de objeto o concepto sobre el que se recoge información: cosa, persona, concepto abstracto o suceso. Por ejemplo: coches, casas, empleados, clientes, empresas, oficios, diseños de productos, conciertos, excursiones, etc. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual.

Hay dos tipos de entidades: fuertes y débiles. Una *entidad débil* es una entidad cuya existencia depende de la existencia de otra entidad. Una *entidad fuerte* es una entidad que no es débil.

Relación
Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior.

Las entidades que están involucradas en una determinada relación se denominan *entidades participantes*. El número de participantes en una relación es lo que se denomina *grado* de la relación. Por lo tanto, una relación en la que participan dos entidades es una relación *binaria*; si son tres las entidades participantes, la relación es *ternaria*; etc.

Una *relación recursiva* es una relación donde la misma entidad participa más de una vez en la relación con distintos papeles. El nombre de estos papeles es importante para determinar la función de cada participación.

La *cardinalidad* con la que una entidad participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad. La participación de una entidad en una relación es *obligatoria (total)* si la existencia de cada una de sus ocurrencias requiere la existencia de, al menos, una ocurrencia de la otra entidad participante. Si no, la participación es *opcional (parcial)*. Las reglas que definen la cardinalidad de las relaciones son las *reglas de negocio*.

Atributo
Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

Cada atributo tiene un conjunto de valores asociados denominado *dominio*. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio.

Los atributos pueden ser simples o compuestos. Un *atributo simple* es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio. Un *atributo compuesto* es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo.

Los atributos también pueden clasificarse en monovalentes o polivalentes. Un *atributo monovalente* es aquel que tiene un solo valor para cada ocurrencia de la entidad o relación a la que pertenece. Un *atributo polivalente* es aquel que tiene varios valores para cada ocurrencia de la entidad o relación a la que pertenece. A estos atributos también se les denomina *multivaluados*, y pueden tener un número máximo y un número mínimo de valores.

La *cardinalidad* de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. El valor por omisión es (1,1).

Por último, los atributos pueden ser derivados. Un *atributo derivado* es aquel que representa un valor que se puede obtener a partir del vabr de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación.

Identificador
Un identificador de una entidad es un atributo o conjunto de atributos que determina de modo único cada ocurrencia de esa entidad. Un identificador de una entidad debe cumplir dos condiciones:

1. No pueden existir dos ocurrencias de la entidad con el mismo valor del identificador.
2. Si se omite cualquier atributo del identificador, la condición anterior deja de cumplirse.

Toda entidad tiene al menos un identificador y puede tener varios identificadores alternativos. Las relaciones no tienen identificadores.

5.- Plataforma WEB

La WEB o World Wide WEB (WWW)

La Web es un sistema de información hipermedial altamente descentralizado. Para comprenderlo y utilizarlo de la mejor manera posible debemos comenzar con entender sus algunos pilares: la arquitectura cliente-servidor, el HTTP (el medio) y el HTML (el mensaje).

La WWW fue creada en 1989-90 por un grupo de científicos en el CERN (Consejo Europeo para la Investigación Nuclear en Geneva, *Conseil Européen pour la Recherche Nucléaire*) con un objetivo muy concreto: compartir información de manera flexible, sencilla y distribuida.

Uno de los principales responsables es *Tim Berners Lee* para quien la WWW es *"un sistema de información multimedial distribuido, heterogéneo y colaborativo"*.

Es multimedial ya que maneja textos, imágenes estáticas y dinámicas (video o animación) y sonido.

Es *distribuida* por que no hay un gran punto central donde la información está concentrada , si no que la WEB cubre todo el globo con miles de servidores, cada uno conteniendo algo de información organizada como un gran hipertexto.

Es *colaborativo* porque permite que cualquiera pueda agregar nueva información y la misma esté disponible rápidamente para todos los demás.

Algunos de los conceptos fundamentales de la World Wide Web son:

- La *arquitectura cliente-servidor*: permite este tipo de operación descentralizada.
- El *HiperText Transfer Protocol* (protocolo de transferencia de hipertextos): permite pasar hipertexto entre clientes y servidores.
- El *HiperText Markup Language* (lenguaje de marcado de hipertextos): permite expresar toda esta información multimedial y que el cliente la interprete.
- Los *estándares*: permiten el avance gracias a los acuerdos en común.

Arquitectura cliente-servidor

Este tipo de estrategia de diseño de programas, separa las funciones en dos programas distintos. Uno de ellos el cliente, se especializa en pedir. Es el que normalmente tenemos en nuestra computadora y aquel con el que interactuamos. Debido a que no tiene todas las capacidades necesarias para completar un servicio (sólo se especializa en pedir), es más pequeño y posee menos requerimientos para correr satisfactoriamente, es decir, necesita menos memoria RAM, menos potencia de procesador, etc.

Del otro lado de la conexión se encuentra la otra mitad, el servidor. Lo único que hace es esperar pedidos de sus clientes y satisfacerlos (ambos programas pueden residir en la misma máquina). Eso si, como en la vida real, se deben pedir correctamente. Para que cliente y servidor se entiendan deben tener un lenguaje en común, un protocolo.

HTTP: HiperText Transfer Protocol

El protocolo utilizado por los clientes y servidores de la Web se denomina *HiperText Transfer Protocol* (protocolo de transferencia de hipertextos). Así como nosotros utilizamos el español en nuestro país, los servidores y clientes de la World Wide Web utilizan el HTTP para entenderse.

Este no es el único protocolo utilizado en la WEB. Una de las características más importantes de la WEB, es que reúne a todos los otros servicios *"bajo el mismo sombrero"*. Pero el HTTP es el protocolo primario, y es el que fue específicamente diseñado para transferir documentos u objetos hipermediales.

El protocolo en sí define una transacción simple de cuatro pasos:

1. El cliente establece una conexión con el servidor.
2. El cliente hace un pedido al servidor (en general especificando un objeto o documento en particular).
3. El servidor devuelve una respuesta conteniendo el status y el contenido de la respuesta (el objeto requerido por el cliente).
4. Se termina (corta) la conexión.

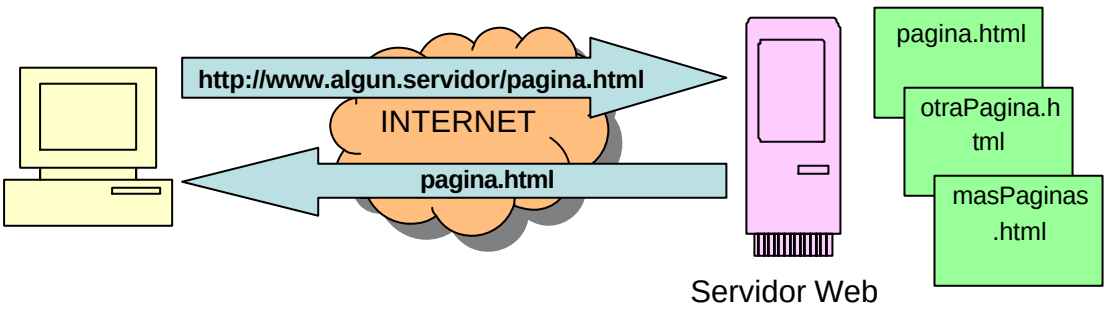


Figura 9 Flujo de una petición en Internet

Algunas de las características del HTTP son:

1. *Simplicidad*. Permite que el servidor maneje poca *"carga"* por cada pedido, de manera de puede atender más pedidos simultáneamente.
2. *Flexibilidad.* Permite tipificar y transmitir cualquier tipo de dato.
3. *Sin conexión (conectionless).* Puede haber sólo un pedido por conexión y luego se corta la misma, de manera que la utilización de recursos es la mínima necesaria.

4. *Sin estado (stateless).* Esto significa que no guarda información sobre transacciones previas. Si bien esto permite agilizar mucho el mecanismo, también es una falencia porque muchas veces es necesario manejar información acerca por ejemplo, de los pasos previos de un usuario y esta información se debe manejar forzando los protocolos.
5. *Permite manejar metainformación (es información acerca de la información).* Esto permite que el agente HTML (el browser en este caso) decida la mejor manera de aprovechar la misma. Por ejemplo, podría mandarse un objeto y especificar el idioma de manera de poder elegir si se desea hacer el *"download"* o no, o la fecha de expiración.

Arquitectura cliente-servidor

En esta arquitectura la computadora de cada uno de los usuarios, llamada cliente, produce una demanda de información a cualquiera de las computadoras que proporcionan información, conocidas como servidores, estos últimos responden a la demanda del cliente que la produjo.

Los clientes y los servidores pueden estar conectados a una red local o una red amplia, como la que se puede implementar en una empresa o a una red mundial como lo es la Internet. Bajo este modelo cada usuario tiene la libertad de obtener la información que requiera en un momento dado proveniente de una o varias fuentes locales o distantes y de procesarla como según le convenga. Los distintos servidores también pueden intercambiar información dentro de esta arquitectura.

Una *arquitectura* es un entramado de componentes funcionales que aprovechando diferentes estándares, convenciones, reglas y procesos, permite integrar una amplia gama de productos y servicios informáticos, de manera que pueden ser utilizados eficazmente dentro de la organización.

Debemos señalar que para seleccionar el modelo de una arquitectura, hay que partir del contexto tecnológico y organizativo del momento y, que la arquitectura Cliente / Servidor requiere una determinada especialización de cada uno de los diferentes componentes que la integran.

Un *cliente* es el que inicia un requerimiento de servicio. El requerimiento inicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones es totalmente transparente para el cliente.

Un *servidor* es cualquier recurso de cómputo dedicado a responder a los requerimientos del cliente. Los servidores pueden estar conectados a los clientes a través de redes LAN o WAN, para proveer de múltiples servicios a los clientes y ciudadanos tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc.

Características del modelo cliente / servidor

En el modelo CLIENTE / SERVIDOR podemos encontrar las siguientes características:

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Un servidor da servicio a múltiples clientes en forma concurrente.
- Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.
- La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.
- Un sistema de servidores realiza múltiples funciones al mismo tiempo que presenta una imagen de un solo sistema a las estaciones Clientes**.** Esto se logra combinando los recursos de cómputo que se encuentran físicamente separados en un solo sistema lógico, proporcionando de esta manera el servicio más efectivo para el usuario final. También es importante hacer notar que las funciones Cliente / Servidor pueden ser dinámicas. Por ejemplo, un servidor puede convertirse en cliente cuando realiza la solicitud de servicios a otras plataformas dentro de la red.
- Además se constituye como el nexo de unión mas adecuado para reconciliar los sistemas de información basados en mainframes o minicomputadoras, con aquellos otros sustentados en entornos informáticos pequeños y estaciones de trabajo.
- Designa un modelo de construcción de sistemas informáticos de carácter distribuido.

En conclusión, Cliente / Servidor puede incluir múltiples plataformas, bases de datos, redes y sistemas operativos. Estos pueden ser de distintos proveedores, en arquitecturas propietarias y no propietarias y funcionando todos al mismo tiempo.

Tipos de servidores

Servidores de archivos
Servidor donde se almacena archivos y aplicaciones de productividad como por ejemplo procesadores de texto, hojas de cálculo, etc.

Servidores de bases de datos
Servidor donde se almacenan las bases de datos, tablas, índices. Es uno de los servidores que más carga tiene.

Servidores de transacciones
Servidor que cumple o procesa todas las transacciones. Valida primero y genera un pedido al servidor de bases de datos.

Servidores de objetos
Contienen objetos que deben estar fuera del servidor de base de datos. Estos objetos pueden ser videos, imágenes, objetos multimedia en general.

Servidores Web
Se usan como una forma inteligente para comunicación entre empresas a través de Internet. Este servidor permite transacciones con el acondicionamiento de un browser específico.

Páginas Web

Las páginas Web son el componente principal de una aplicación o sitio Web. Los browsers piden páginas (almacenadas o creadas dinámicamente) con información a los servidores Web.

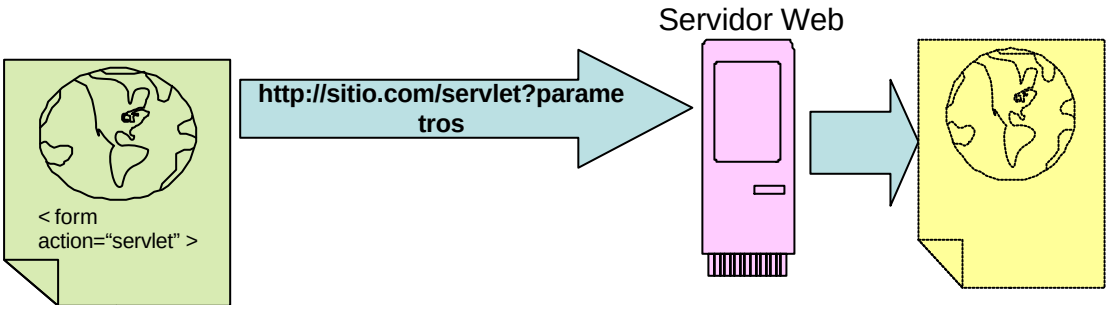


Figura 10 Páginas Web

En algunos ambientes de desarrollo de aplicaciones Web, las páginas contienen código HTML y scripts dinámicos, que son ejecutados por el servidor antes de entregar la página.

Una vez que se entrega una página, la conexión entre el browser y el servidor Web se rompe (a diferencia de otros esquemas tipo cliente / servidor). Es decir

que la lógica del negocio en el servidor solamente se activa por la ejecución de los scripts de las páginas solicitadas por el browser (en el servidor, no en el cliente).

Contenido dinámico

Cuando la WWW se convirtió en canal para prestar servicios apareció la necesidad de contenido dinámico. Para el contenido dinámico del lado del cliente existen: JavaScript, VBScript, Applets y para el contenido dinámico generado en el servidor se tienen: CGI, Servlets, JSP, ASP, PHP.

Para el contenido dinámico del lado del cliente es necesario que el navegador soporte dichas tecnologías por medio de interpretes y ambientes de ejecución de los lenguajes.

Las tecnologías de contenido dinámico en el servidor crean los documentos html *"al vuelo"* tomando parámetros que el usuario envía, configurando la página según las peticiones del usuario,.

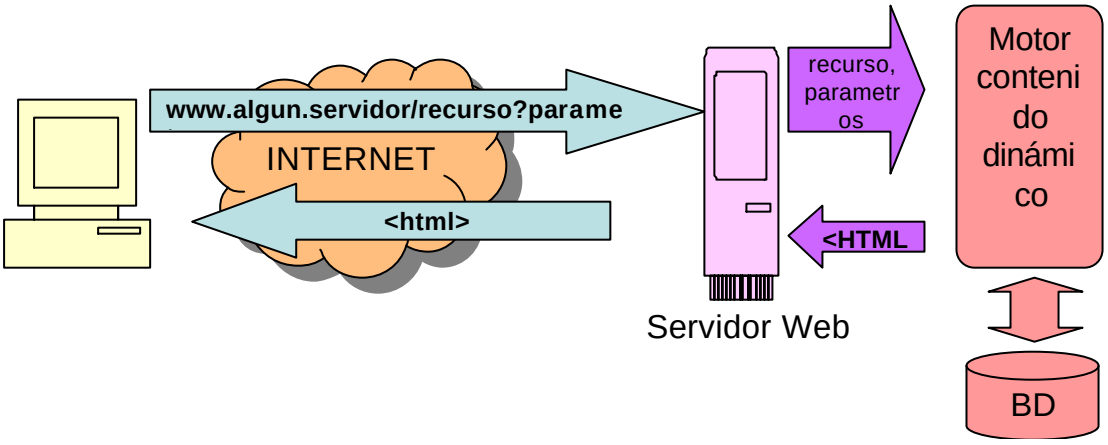


Figura 11 Contenido dinámico generado en el servidor

CAPITULO 2 REQUERIMIENTOS

1.- Comienzo del proyecto de software

Antes de poder empezar a planificar un proyecto, deben establecerse el ámbito y los objetivos, deben considerarse soluciones alternativas y deben identificarse las restricciones técnicas y de gestión. Sin esta información, es imposible obtener unas estimaciones de coste razonables y precisas, una identificación realista de las tareas del proyecto o un plan de trabajo adecuado que proporcione una indicación significativa del progreso.

El desarrollador del software y el cliente deben ponerse de acuerdo para definir el ámbito y los objetivos del proyecto. Los objetivos identifican los fines globales del proyecto sin considerar cómo se llegará a esos fines. El ámbito identifica las funciones primordiales que debe llevar a cabo el software y lo que es más importante, intenta limitar esas funciones de manera cuantitativa.

Una vez entendidos los objetivos y el ámbito del proyecto se han de considerar soluciones alternativas. Aunque se estudien con muy poco detalle, las alternativas han de permitir a los gestores y a los desarrolladores seleccionar el *"mejor"* enfoque, dadas las restricciones impuestas por las fechas de entrega, el presupuesto, etc.

Para esto se debe tener una serie de reuniones ente el cliente y el desarrollador para poder llegar a un entendimiento mutuo, y realizar la captura de los requerimientos, es decir, la comprensión de lo que los clientes y los usuarios esperan que haga el sistema.

2.- Fase de definición

La fase de definición de la ingeniería de software comienza con la etapa de planificación, durante esta etapa se desarrolla una descripción bien delimitada del ámbito del esfuerzo del software, se definen los recursos necesarios para desarrollar el software y se establecen estimaciones.

El propósito de la etapa de planificación del software es proporcionar una indicación preliminar de la viabilidad del proyecto de acuerdo con el coste y con la agenda.

Definición del problema

En la actualidad es complicado el llevar a cabo una reunión de trabajo en la cual se toman decisiones importantes, como comenta David Meador, vicepresidente de la Compañía Eléctrica de Detroit:

"Demasiadas veces comenzamos una reunión y no nos damos cuenta de que, dos tercios de los presentes no tienen idea de que porque están ahí. Por supuesto, no hay tiempo para explicarles, estamos demasiados preocupados por cumplir la agenda. Al final de la reunión asumimos sin verificarlos que todos comprendieron los temas y se comprometieron con las decisiones.

Después, no se cumple lo que acordamos es un desastre. Y nos sorprendemos, ya que pensábamos que la reunión había sido estupenda. De hecho, siempre pensamos que cuanto menos preguntas se hagan, mejor. Si no hay preguntas, inferimos que todo mundo entendió y estuvo de acuerdo. "

Para solucionar este problema, se propone desarrollar un software que asista en la tarea de gestión de reuniones.

Objetivo

Desarrollar un sistema que logre ayudar en todas las etapas que conforman una reunión, y que cumpla con los siguientes objetivos:

- Ser una herramienta que apoye la gestión de actividades y proyectos dentro de las organizaciones.
- Facilitar la administración y seguimiento de todos los acuerdos derivados de reuniones registradas en la herramienta.
- Incrementar la comunicación y colaboración entre los equipos de trabajo.
- Otorgar a las organizaciones la infraestructura para administrar el conocimiento y lecciones aprendidas que surgen durante las reuniones de trabajo.
- Reducir el tiempo de registro y control de cualquier tipo de evento.

Dado que actualmente la mayor parte de las personas tienen acceso a una computadora con Internet, se desea que en el momento se necesite organizar una reunión, simplemente se vaya a la computadora, se acceda a Internet y se comience a documentar la reunión.

3.- Metodología de gestión de reuniones

La reunión de trabajo es una herramienta muy importante para cualquier persona que desee hacer más eficientes sus esfuerzos para lograr un objetivo. El riesgo del fracaso siempre está latente y sólo una acción efectiva de quienes se declaren responsables de la misma pueden alterar los resultados.

La efectividad de las reuniones de trabajo depende de varios factores, por lo que se propone una metodología que considera las siguientes etapas:

- Planeación
- Convocatoria
- Desarrollo
- Seguimiento

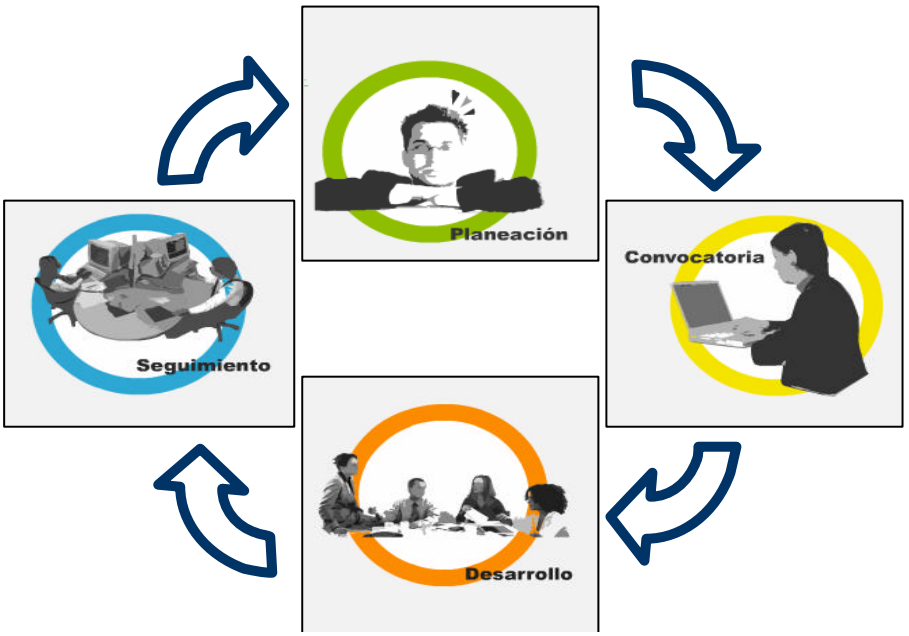


Figura 12 Metodología de gestión de reuniones

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

Planeación

El éxito de las reuniones depende fundamentalmente de su preparación cuidadosa por lo que deben ser planeadas con anticipación. Para que una reunión sea exitosa, es necesario planificarla y prepararla bien, definiendo:

- Título del evento
- Fecha y hora de inicio
- Fecha y hora de término
- Lugar
- Objetivo
- Carácter
- Puntos de la agenda
- Personas convocadas

La persona que planea una reunión toma el rol de “Convocante” de la reunión y los demás asistentes serán los “Convocados”

Convocatoria

La convocatoria es el aviso previo que se hace para dar a conocer la agenda a los participantes y vayan a la reunión con pleno conocimiento de lo que se va a tratar y hayan podido formarse un juicio u opinión personal sobre los temas para considerar. Por ultimo es de gran importancia que la convocatoria se haga con suficiente anticipación para que los participantes prevean el tiempo.

Desarrollo

Es cuando se lleva a cabo la reunión y es aquí donde:

- Se verifica la asistencia con el propósito de tener un registro de las personas que participaron, para cualquier aclaración en cuanto a lo acordado en la reunión.
- Se da lectura a la agenda y se desarrollan los puntos a tratar en ella.
- Se generan acuerdos, fechas de compromiso, responsables e instrucciones a seguir.
- Se genera la minuta de la reunión

Es importante mencionar que la agenda establecida en la etapa de planeación no es definitiva, porque en el momento en que se lleva a cabo la reunión, es muy probable que se agreguen, modifiquen o eliminen puntos, lo mismo sucede con los acuerdos, responsables e instrucciones.

Seguimiento

Para que una reunión cumpla totalmente con su objetivo, es necesario que las decisiones o acuerdos alcanzados en ella se respeten. Para ello es fundamental que el convocante haga llegar, lo más pronto posible, a quienes asistieron a la reunión, la minuta de la misma, en la que se incluyan todos los acuerdos adoptados, responsabilidades asignadas y los plazos establecidos para el cumplimiento de ellas.

La función del convocante de una reunión no termina con la entrega de la minuta, si no con el seguimiento posterior de la forma en que cada participante cumple con las responsabilidades que contrajo. Desafortunadamente, ésta es una práctica muy poco frecuente y es lamentable comprobar que luego de largas reuniones, en que se distribuyen diferentes tareas, éstas no se cumplen por falta de seguimiento posterior. Es por esto que se debe:

- Realizar las actividades necesarias para cumplir en su totalidad con los acuerdos que se nos han asignado.
- Revisar y calificar las actividades de los acuerdos que hemos asignado.
- Generar informes de los avances.

4.- Requerimientos

Un requerimiento es un a característica o una descripción de algo que el sistema es capaz de hacer con el objetivo de satisfacer el propósito del sistema. Con esto se pretende determinar el límite del sistema.

En este caso se desea que el sistema sea accesible desde Internet y que la información se comparta y este actualizada.

Además el sistema debe cumplir con la metodología de gestión de reuniones, es decir, debe seguir el ciclo de una reunión:

- Realizar la planeación de una reunión, definir la fecha, hora, lugar y participantes, así como establecer los objetivos y plantear la agenda. Al finalizar esta etapa se debe poder obtener un documento con la "Orden del Día" o "Agenda".

- En segundo lugar se procede a la convocatoria de los participantes, es decir, la notificación mediante cualquier medio a los involucrados y hacer de su conocimiento la agenda.

- Una vez llegado el momento de la reunión es necesario llevar registro del desarrollo, de las personas que asistieron, los puntos que se trataron y los acuerdos a los que se llegaron. Al finalizar esta etapa se debe obtener la "Minuta"

- Después de celebrada la reunión se le debe dar seguimiento a los asuntos tratados en la misma, es decir, se puede planear otra reunión o serie de reuniones para continuar con los asuntos pendientes o en caso de que ya no se necesiten mas reuniones, vigilar el avance y cumplimiento de los acuerdos.

Se puede asignar tareas sin tener que entrar al ciclo de una reunión y que al igual que a los acuerdos se les pueda dar seguimiento, capturar sus actividades, delegar responsabilidades y adjuntar archivos.

Se comunicará a asignación de tareas y captura de avances mediante correo electrónico, así como también se debe poder obtener los reportes impresos de las tareas pendientes o en seguimiento.

Especificación de requerimientos

La definición de los requerimientos del software es un esfuerzo conjunto llevado a cabo por el desarrollador y el cliente. *La especificación del requerimientos de software* es el documento que se produce como resultado de esta etapa. La fase de definición culmina con una revisión técnica de este documento realizada por el desarrollador del software y el cliente.

Los requerimientos base son los siguientes:

- El sistema debe ser accesible desde Internet
- La información debe ser compartida entre todos los usuarios.
- La información debe estar actualizada.
- El sistema se debe ajustar a la metodología de gestión de reuniones (planear, convocar, desarrollar y dar seguimiento)
- La planeación debe contener los datos generales de fecha, lugar, hora, objetivos, etc.
- Poder agregar asistentes a una reunión.
- Definir e imprimir una orden del día.
- Convocar a los asistentes.
- Llevar el desarrollo de la reunión, capturando puntos tratados y acuerdos asignados.
- Se tendrá control sobre la asistencia de los convocados.
- Generar e imprimir la minuta de la reunión.
- Dar seguimiento a las tareas asignadas en una reunión.
- Planear una o varias reuniones en las que se continúen los asuntos pendientes de una anterior.
- Vigilar el cumplimiento de los acuerdos.
- Asignar tareas sin tener que entrar al ciclo de una reunión.
- Dar seguimiento y vigilar el cumplimiento de las tareas al igual que los acuerdos.
- Compartir archivos relacionados con las tareas o acuerdos.
- Tener comunicación por correo electrónico cuando se asignan tareas o acuerdos y cuando se capturan avances.
- Obtener e imprimir los reportes de las tareas y el seguimiento.

CAPITULO 3 ANÁLISIS

1.-Análisis de requerimientos

Para que un esfuerzo de desarrollo de software tenga éxito, es esencial comprender perfectamente los requerimientos del software. Independientemente de lo bien diseñado o codificado que esté un programa, si se ha analizado y especificado pobremente, decepcionará al usuario y desprestigiará al que lo ha desarrollado.

El análisis de requerimientos es un proceso de descubrimiento, refinamiento, modelado y especificación. Comienza con un refinamiento detallado del ámbito del programa, que ha sido inicialmente establecido durante la ingeniería del sistema y refinado durante la planificación del proyecto de software. Se crean los modelos de flujo de la información y del control, del comportamiento en operación y del contenido de los datos. Se analizan las soluciones alternativas, con la asignación de los distintos elementos del software.

Tanto el desarrollador como el cliente juegan un papel activo en el análisis de requerimientos. El cliente intenta volver a plantear detalladamente el concepto, algo nebuloso, de la función y del comportamiento del software. El desarrollador actúa como interrogador, como consultor y como persona que resuelve problemas.

El análisis de los requerimientos puede parecer una tarea relativamente sencilla, pero las apariencias engañan. El contenido de la comunicación es muy denso. Abundan los casos en que se puede llegar a malas interpretaciones o falta de información. Es probable que se llegue a ambigüedades.

El análisis de requerimientos es la tarea de la ingeniería de software que establece un puente entre la asignación del software a nivel de sistema y el diseño del software, facilita al ingeniero de sistemas la especificación de la función y del rendimiento del software, la descripción de la interfaz con otros elementos del sistema y el establecimiento de las restricciones de diseño que debe considerar el software.

También permite al ingeniero de software refinar la asignación del software y construir modelos de los ámbitos del proceso, de los datos y del comportamiento que serán cubiertos por el software, proporciona al diseñador del software una representación de la información y de las funciones que se puede traducir en un diseño de datos, arquitectónico y procedimental.

2.-Análisis del sistema

Como se desea que el sistema sea accesible desde cualquier computadora con acceso a Internet o a una intranet, será un sistema WEB y para que la información este compartida y actualizada en todo momento, tendrá una base de datos centralizada.

Por ser un sistema WEB, debemos tener un control de quien pueden acceder a él, por lo que se tendrá un registro de las personas que tienen una cuenta para utilizar el sistema y se hará la validación por medio de un nombre de usuario y una contraseña.

Con el objeto de personalizar las actividades a las que cada usuario tiene acceso, dependiendo de la persona que ingrese al sistema, ésta tendrá un perfil específico con sus permisos y restricciones. Esto implica que se contará con un módulo de seguridad.

Un módulo es una pequeña parte del sistema que tiene a su vez funcionalidades específicas como ver una pantalla, agregar un dato, ver un reporte, etc. Un ejemplo puede ser: módulo de seguridad y administración, funcionalidad agregar usuarios.

USUARIOS

En el sistema existirán 3 tipos de personas o usuarios:

Administrador, Normal y Contacto.

1. *Administrador:* es el único que puede dar de alta y hacer cambios de usuarios y solo existirá uno en el sistema.
2. *Normal:* es todo aquél que cuenta con una cuenta para usar el sistema y además puede dar de alta Contactos.
3. *Contacto:* es una persona que solo participa como responsable de algún acuerdo o asistente a una reunión y no tiene acceso al sistema.

MÓDULO DE SEGURIDAD Y ADMINSITRACIÓN

A éste módulo solo el administrador del sistema tendrá acceso, y en él se podrán realizar las siguientes acciones:

Se definirán los usuarios

Aquí el administrador podrá dar de alta a un usuario nuevo, obteniendo la información necesaria como nombre completo, dirección, etc. y le asignará un nombre de usuario mejor conocido como *"Login"* y una contraseña o *"Password"* así como también le asignará un perfil.

También aquí el administrador podrá hacer que un usuario de tipo *"Contacto"* se convierta en *"Normal"* haciendo el cambio de tipo y asignándole un Nombre de Usuario y una Contraseña.

Se definirán los perfiles

Los perfiles darán acceso o restricción a los usuarios *"Normales"* a ciertos módulos y funcionalidades. Existirá un catálogo de módulos y uno de funcionalidades por módulo y habrá una clave para cada funcionalidad.

El administrador definirá los perfiles según las necesidades en base al catalogo módulo-funcionalidad, para esto existirá un catálogo de Perfiles y tendrá relación con el de módulo-funcionalidad.

Se accederá a los catálogos genéricos

El sistema utilizará una serie de catálogos que se irán llenando de acuerdo a las necesidades, pero existen algunos (como el de países, el de estados y empresas) que deben llenarse inicialmente para poder utilizar el módulo de seguridad y administración correctamente, por lo que en este módulo también se tendrá acceso a ellos.

Una vez que el Administrador ha definido los usuarios y asignado los perfiles de cada uno, éstos pueden hacer uso del sistema.

PANTALLA PRINCIPAL

Al entrar al sistema un usuario *"Normal"* independientemente del perfil con el que cuente, podrá ver una pantalla principal que tendrá un resumen de las tareas y reuniones pendientes, el avance de los acuerdos asignados a otras personas, etc. Mediante esta pantalla el usuario podrá entrar a los módulos y funcionalidades del sistema.

Los módulos que existirán en el sistema son:

- Catálogos
- Planeación
- Convocatoria
- Desarrollo
- Seguimiento
- Estadísticas

MÓDULO DE CATALOGOS

En este módulo se podrá llenar la información que será utilizada posteriormente por otras funcionalidades, estos catálogos tendrán un papel importante en el funcionamiento del sistema ya que contendrán datos que serán usados constantemente y evitarán el tener que estar capturando la información una y otra vez.

MÓDULO DE PLANEACIÓN

En este módulo se llevará a cabo la etapa de planeación de una reunión. Esta etapa comenzará desde que el usuario organiza sus reuniones, es decir, generará una estructura de directorios donde irá colocando las reuniones que planea. Una vez hecho esto, el usuario procederá a planear una nueva reunión en el directorio donde será creada la reunión y le asignará un nombre o título, una fecha y hora de inicio y fin, lugar en donde será, observaciones, etc.

Al hacer esto el usuario podrá ahora definir una orden del día para su reunión, es decir, podrá agregar, modificar o eliminar puntos y sub puntos a tratar. También deberá seleccionar a las personas que invitará o convocará a la reunión.

Al terminar esta etapa podrá generar un documento que contendrá la Orden del Día.

Al planear una reunión la persona que planea la reunión, llamada *"Convocante"*, se agregará automáticamente como asistente y no podrá ser eliminado de dicha reunión.

En esta etapa la reunión tendrá un status de PLANEADA y en esta etapa se podrán modificar los datos generales de la reunión.

Si se planea una nueva reunión en el mismo directorio, por default cargará los datos generados en la última reunión, pero dichos datos podrán ser modificados. También se podrán agregar los mismos asistentes y puntos, y en caso de que haya acuerdos pendientes de las reuniones anteriores en el mismo directorio que la reunión que esta siendo planeada, se podrán arrastrar los pendientes a esta reunión, seleccionando cuales se incluirán.

En esta etapa los convocados no tienen conocimiento de que han sido invitados a la reunión.

MÓDULO DE CONVOCATORIA

En este módulo que se encargará de la etapa de Convocatoria, el usuario *"Convocante"*, hará que los demás asistentes a la reunión, o *"Convocados"*, tengan conocimiento de la reunión, es decir, los usuarios que tienen acceso podrán ver la información al entrar al sistema y podrán ver el documento con la orden del día . Al hacer esto, el status de la reunión pasa de PLANEADA a CONVOCADA.

MÓDULO DE DESARROLLO

Este módulo o etapa se da cuando llega el momento de la reunión, para pasar a desarrollo la reunión deberá tener un status de CONVOCADA en otro caso no se podrá entrar a esta etapa.

Cuando el usuario entre al desarrollo, el sistema obtendrá automáticamente la hora actual para iniciar la reunión, esto con el propósito de medir el tiempo real que dura la reunión.

Ahora se verá la orden del día o agenda de la reunión y en ella se podrán agregar, modificar o eliminar puntos o sub puntos y capturarles una breve narrativa. Una vez definida la agenda se podrá empezar a desarrollar la reunión y cada vez que se tenga un acuerdo se capturará debidamente en el punto correspondiente dándole una descripción detallada, la fecha compromiso, la prioridad, el o los responsables que podrán ser personas asistentes o no asistentes a la reunión y una breve descripción de la actividad o instrucción por persona o para todos para cumplir con el acuerdo.

Así se seguirá el proceso de desarrollo de la reunión hasta que todos los acuerdos hayan sido capturados y asignados.

También existirá una opción en donde se verán los asistentes que se tenían considerados en la etapa de planeación, para proceder a "pasar lista de asistencia" a los convocados uno por uno o todos a la vez, así como también se podrán dar de alta suplentes de los convocados, agregar o eliminar convocados.

Al salir del desarrollo, preguntará si quiere darse la reunión por terminada o si se desea continuar en desarrollo, en caso de que se elija terminar la reunión, cambiará su status a TERMINADA y se obtendrá automáticamente la hora de término de la reunión con lo que se calculará el tiempo real de duración de la reunión. En esta etapa se podrá obtener el documento de *"Minuta"* de la reunión que contendrá la orden del día y los acuerdos y responsables de cada uno de los puntos.

MÓDULO DE SEGUIMIENTO

Este módulo o etapa se iniciará cuando se asignen responsables a los acuerdos de una reunión o bien cuando se asignen tareas a una persona aunque no provengan de una reunión. Aquí básicamente el objetivo es calificar el cumplimiento de los acuerdos y tareas.

Aquí existirán dos vertientes, una llamada *"Tareas"* para ver los acuerdos o tareas que le fueron asignadas al usuario y podrá capturar sus avances y adjuntar archivos para que los involucrados los puedan descargar, así como también podrá delegar las tareas que le han sido encargadas.

La otra será llamada *"Seguimiento"* en donde el usuario podrá ver las tareas que asignó a otras personas, sus avances, archivos adjuntos, así como también calificarlas. En esta parte, también podrá dar de alta tareas libres, es decir, que no tendrá que entrar en el ciclo de una reunión para asignar una tarea a una persona.

Un acuerdo se resuelve si todos los responsables están resueltos, es decir sus actividades cumplen con su objetivo. Al calificar un acuerdo, se cambia el status para todos los responsables involucrados en el acuerdo, o si a todos los responsables se les cambia el status a resuelto, el acuerdo también se considerará resuelto. En ambos casos se podrá obtener esta información en un reporte.

MÓDULO DE ESTADÍSTICAS

En este módulo se verán las estadísticas que mostrarán de manera gráfica los avances de las personas en algunos periodos de tiempo. Servirá para tener una comparativa de rendimiento de los usuarios.

STATUS

Los distintos status que se manejarán en el sistema son los siguientes:

- RESPONSABLES
  - NO RESUELTO
  - RESUELTO
- ACUERDOS
  - NO RESUELTO
  - CANCELADO
  - RESUELTO
- CATALOGOS, USUARIOS Y PERFILES
  - ACTIVO
  - INACTIVO
- REUNION
  - PLANEADA
  - CONVOCADA
  - DESARROLLO
  - TERMINADA

3.-Casos de uso

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Los *Casos de Uso* describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario, permiten definir los límites del sistema y las relaciones entre el sistema y el entorno, son descripciones de la funcionalidad del sistema independientes de la implementación, cubren la carencia existente en métodos previos en cuanto a la determinación de requisitos, están basados en el lenguaje natural, es decir, es accesible por los usuarios, entre otras cosas.

Existen los siguientes tipos de actores:

Principales : personas que usan el sistema.
Secundarios : personas que mantienen o administran el sistema.
Material externo : dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados.
Otros sistemas : sistemas con los que el sistema interactúa.

La misma persona física puede interpretar varios papeles como actores distintos, el nombre del actor describe el papel desempeñado.

Los Casos de Uso se determinan observando y precisando, actor por actor, las secuencias de interacción, los escenarios, desde el punto de vista del usuario. Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso. Un escenario es una instancia de un caso de uso.

Como ejemplo de los casos de uso del sistema se muestra el caso de uso *Acuerdos*, el resto de los casos se encuentran en la sección de anexos.

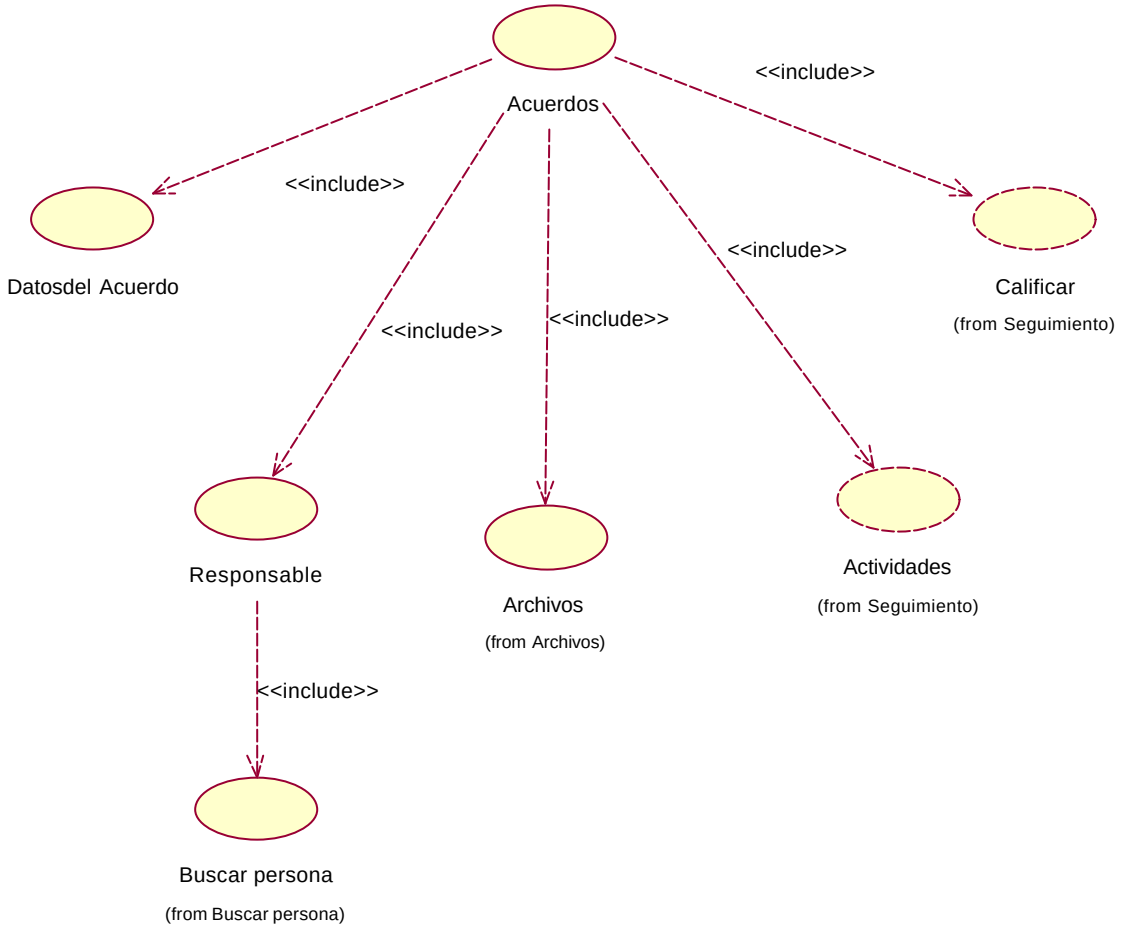


Figura 13 Caso de uso "Acuerdo"

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

4.-Flujo del sistema

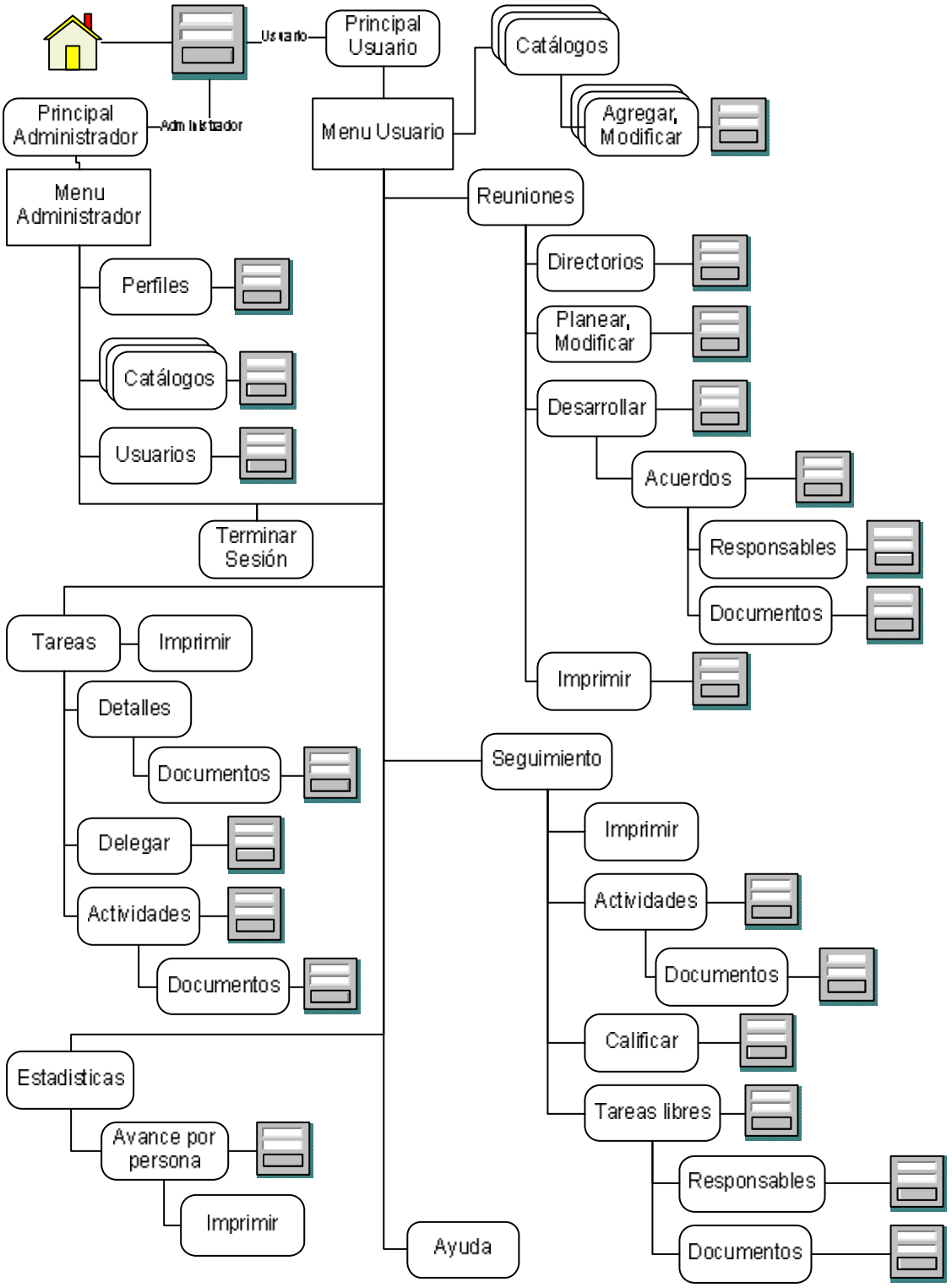


Figura 14 Flujo del sistema

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

Existirá una pantalla inicial para entrar al sistema en donde se solicitará el nombre de usuario y su contraseña para entrar al sistema.

Una vez proporcionados y validados estos datos hay dos vertientes, si el usuario es administrador, entonces va a una pantalla principal del módulo de administración y si es un usuario normal, entonces va a una pantalla principal en donde existe un resumen de toda su información más reciente e importante.

Siguiendo por el módulo de administración, el administrador tendrá un menú especial en el que podrá acceder a los perfiles, catálogos y usuarios.

En perfiles habrá una pantalla donde se mostrarán los perfiles que actualmente se encuentran en el sistema y las funcionalidades a las que da acceso cada uno de ellos, y se tendrá la opción de agregar uno nuevo o modificar alguno de los existentes.

En catálogos existirán varias pantallas dependiendo el catálogo al que se quiera entrar en donde se verán los datos dados de alta en cada caso y para cada catálogo existirán la opciones de agregar, modificar y eliminar.

En usuarios se podrán dar de alta nuevos usuarios del sistema, al igual que en catálogos existirá una pantalla donde se podrán ver todos los usuarios registrados actualmente y se tendrá la opción de modificar los existentes o agregar uno nuevo, capturando sus datos generales y asignándole un nombre de usuario o *"login"* y una contraseña o *"password"*, así como un perfil para darle acceso a las funcionalidades.

Y finalmente también tendrá una opción de menú para terminar su sesión en el sistema, la cual también podrá ser accedida desde el menú del usuario normal.

Si el usuario que inicia sesión no es administrador, es decir, es un usuario normal, entonces entrará a la pantalla principal en donde tendrá un menú que le dará acceso a catálogos, reuniones, seguimiento, tareas, estadísticas y ayuda, así como también podrá terminar la sesión como en el módulo de administración.

En catálogos será similar a los catálogos del módulo de administrador pero la diferencia será los catálogos a los que se tendrá acceso en ambos casos.

En reuniones se podrán organizar las reuniones pero para tal efecto primero cada usuario deberá definir una estructura jerárquica de directorios o carpetas como cada uno desee, en este caso aparecerá una pantalla donde se podrá ver los directorios creados y su estructura y tendrá las opciones de agregar, modificar o eliminar directorios, cuidando de que no se pueden eliminar directorios que ya contengan otros directorios o reuniones y que no se puedan agregar al mismo nivel indistintamente directorios y regiones.

Una vez hecho esto, se seleccionará un directorio para comenzar a planear reuniones dentro de él, y con esto iniciar el ciclo de reuniones.

Al planear una reunión aparecerá una pantalla donde se capturarán los datos generales de la reunión, los puntos a tratar y los asistentes a dicha reunión. Al terminar esta captura la reunión tendrá el status de PLANEADA. En esta misma pantalla se tendrá la opción de pasar al status de CONVOCADA para que los convocados tengan conocimiento de la reunión.

Si se desea modificar los datos de una reunión, solamente se podrá hacer esto cuando la reunión tenga status de PLANEADA o CONVOCADA en otro caso no se podrá modificar. Igualmente si se desea eliminar una reunión.

Cuando una reunión esté en status CONVOCADA se activará la opción de Desarrollar para pasar al desarrollo de la reunión y cambiar su status a DESARROLLO.

En desarrollar existirá una pantalla donde se concentrará la información de la reunión en cuestión, es decir, tendrá en una parte la orden del día o los puntos que se definieron al planearla, los cuales pueden ser modificados en esta pantalla así como agregar otros o eliminar los innecesarios.

En otra sección de la pantalla se tendrán los acuerdos a los que se ha llegado en cada punto de la reunión, y en esta parte se tendrá la opción de agregar, modificar o eliminar acuerdos.

En otra sección se tendrán los responsables de cada acuerdo y su fecha compromiso y se tendrá la opción de calificarlos o cambiar su status desde aquí.

En la última sección se tendrán las actividades o avances de cada responsable del acuerdo y en esta parte el solicitante de los acuerdos se dará por enterado de dichos avances.

Para agregar acuerdos la logística será la siguiente: se capturará el texto del acuerdo y sus datos generales, al guardar esta información inmediatamente se debe seleccionar el o los responsables del acuerdo de una lista de usuarios o contactos del sistema y asignar una fecha compromiso para dichos responsables. Si además se desea adjuntar un archivo o documento al acuerdo también se tiene la opción de entrar a una pantalla donde se pueden ver los archivos actualmente ligados al acuerdo y se pueden agregar nuevos o eliminar existentes.

Al terminar esto, los acuerdos y los responsables aparecerán en las secciones correspondientes.

Para terminar la reunión existirá una opción de Terminar Reunión que hará que se cambie el status a TERMINADA, pero para esto primero se debe pasar lista de asistencia. Y se regresará a la pantalla de reuniones.

Finalmente en la pantalla de reuniones tendrá la opción de imprimir los reportes correspondientes a cada reunión, para esto se deberá seleccionar la reunión y el tipo de reporte que se desea, es decir, orden del día, minuta o reporte de avances.

En Seguimiento, se tendrá una pantalla con la información de todos los acuerdos o tareas que el usuario en sesión ha asignado a otras personas, para que pueda llevar el seguimiento de estas. En esta pantalla se contará con varios filtros para poder hacer búsquedas especializadas y se tendrá la opción de imprimir un reporte con la información que aparece en ese momento.

También en Seguimiento se podrá ver los avances o actividades que los responsables han reportado al solicitante y los documentos que dichas personas han adjuntado a sus correspondientes actividades y existirá también la opción de que el solicitante califique a cada responsable eligiendo la fecha en la que se resolvió la tarea.

También aquí existirá la opción de dar de alta tareas libres, las cuales serán igual que los acuerdos, con la diferencia de que estas no provienen de una reunión. Al igual que los acuerdos, al dar de alta una tarea se debe asignar a los responsables y se pueden adjuntar archivos a estas.

En Tareas se podrán ver todas las tareas o acuerdos en los que el usuario en sesión es responsable, es decir, todo lo que le han encargado cumplir. Esta pantalla será similar a la de Seguimiento con la diferencia de las opciones que se tendrán en esta pantalla, las cuales en este caso serán : Ver los detalles, Imprimir reporte, Delegar la tarea, Capturar Avances o actividades.

Para ver los detalles de la tarea, se debe seleccionar la tarea y se podrá acceder a una pantalla donde se verá la información general de la tarea, los responsables y los archivos adjuntos de la tarea. La opción de imprimir imprimirá el reporte de tareas con la información que se muestre en la pantalla en ese momento.

Para delegar la tarea, aparecerá una pantalla donde se podrá seleccionar el o las personas a las que el usuario en sesión podrá delegar la tarea, convirtiéndose en el solicitante para los delegados en cuestión.

En la captura de actividades, el usuario reportará al solicitante de la tarea, sus avances con respecto a dicha tarea y opcionalmente podrá adjuntar archivos a la actividad para que el responsable pueda tener acceso a ellos.

En Estadísticas el usuario podrá seleccionar un periodo de tiempo y las personas de las que desea ver la información y al hacer esto visualizará una estadística con los avances por persona con los filtros seleccionados y contará también con la opción de imprimir un reporte con esta información.

Por último, en Ayuda, el usuario tendrá acceso a una guía rápida o manual del sistema en línea para solucionar dudas o problemas que pueda tener.

5.- Estimación

Cuando se planifica un proyecto de software se tienen que obtener estimaciones del esfuerzo humano requerido, de la duración cronológica del proyecto y del coste.

Se requieren estimaciones cuantitativas, pero no hay disponible ninguna información sólida. Un análisis detallado de los requisitos del software proporcionaría la información necesaria para las estimaciones, pero el análisis suele durar semanas o meses. Las estimaciones son necesarias “ahora”.

El objetivo de la planificación del proyecto de software es el de suministrar una estructura que permita hacer estimaciones razonables de recursos, costes y agendas. Estas estimaciones se hacen sin un marco de tiempo limitado, al principio de un proyecto de software y deben de ser actualizadas regularmente a medida que progresa el proyecto.

La primera actividad, es determinar el ámbito del software. Se deben evaluar la función y el rendimiento asignados al software durante el análisis de requerimientos, con el fin de establecer un ámbito del proyecto que no sea ambiguo ni incomprensible. La especificación del ámbito del software debe estar delimitada, es decir, han de establecerse explícitamente los datos cuantitativos, restricciones y / o limitaciones, etc.

Para este sistema se tiene una estimación de tiempo como se muestra a continuación.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Requerimientos**	**30 días?**	**lun 10/01/05**	**vie 18/02/05**
2	Reuniones con el cliente	10 días?	lun 10/01/05	vie 21/01/05
3	Planificacion	4 días?	lun 24/01/05	jue 27/01/05
4	Definición del problema	8 días?	vie 28/01/05	mar 08/02/05
5	Especificación de requer	8 días?	mié 09/02/05	vie 18/02/05
6	**Analisis**	**40 días?**	**lun 21/02/05**	**vie 15/04/05**
7	Análisis de requerimient	5 días?	lun 21/02/05	vie 25/02/05
8	Análisis del sistema	10 días?	lun 28/02/05	vie 11/03/05
9	Casos de Uso	15 días?	lun 14/03/05	vie 01/04/05
10	Definición de flujo del sis	10 días?	lun 04/04/05	vie 15/04/05
11	**Diseño**	**45 días?**	**lun 18/04/05**	**vie 17/06/05**
12	Diseño de la BD	25 días?	lun 18/04/05	vie 20/05/05
13	Definición de objetos y m	10 días?	lun 23/05/05	vie 03/06/05
14	Elección de tecnologías	10 días?	lun 06/06/05	vie 17/06/05
15	**Desarrollo**	**70 días?**	**lun 20/06/05**	**vie 23/09/05**
16	Metodología de desarroll	5 días?	lun 20/06/05	vie 24/06/05
17	Arquitectura	5 días?	lun 27/06/05	vie 01/07/05
18	Proceso de datos	5 días?	lun 04/07/05	vie 08/07/05
19	Codificación	55 días?	lun 11/07/05	vie 23/09/05
20	**Pruebas**	**20 días?**	**lun 26/09/05**	**vie 21/10/05**
21	Instalación	5 días?	lun 26/09/05	vie 30/09/05
22	Pruebas	15 días?	lun 03/10/05	vie 21/10/05
23	**Liberacion**	**6 días?**	**lun 24/10/05**	**lun 31/10/05**
24	Entregables	3 días?	lun 24/10/05	mié 26/10/05
25	Capacitación	3 días?	jue 27/10/05	lun 31/10/05

Figura 15 Tabla de estimación de tiempo

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

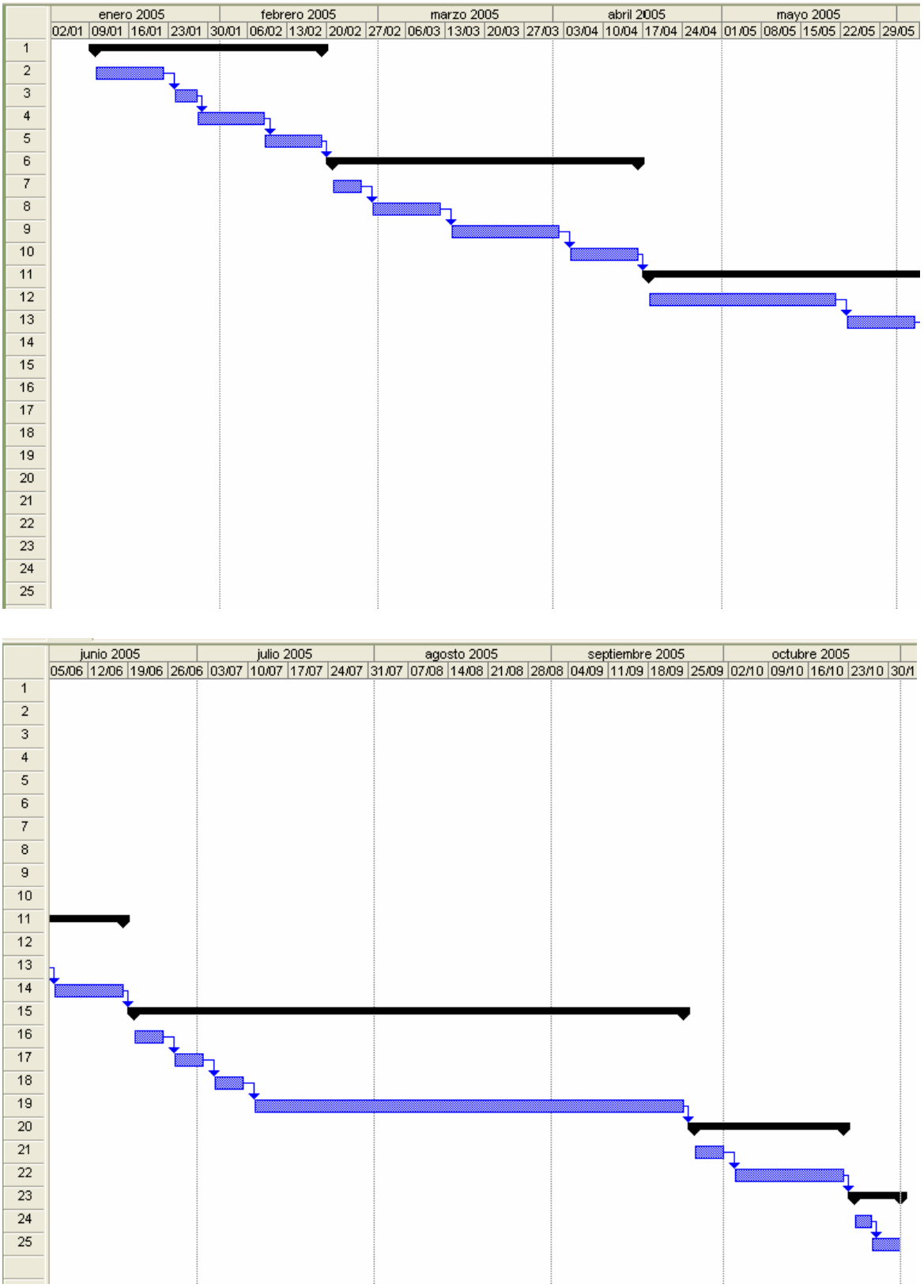


Figura 16 Diagrama de Gant de la estimación de tiempo

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

CAPITULO 4 DISEÑO

El diseño es el primer paso de la fase de desarrollo de cualquier producto o sistema de ingeniería. Puede definirse como: *"... el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física".*

El objetivo del diseñador es producir un modelo o representación de una entidad que se construirá más adelante. El proceso por el cual se desarrolla el modelo combina la intuición y los criterios en base a la experiencia de construir entidades similares, un conjunto de principios y / o heurísticas que guían la forma en la que se desarrolla el modelo, un conjunto de criterios que permiten discernir sobre la calidad y un proceso de iteración que conduce finalmente a una representación del diseño final.

1.-Diseño Orientado a Objetos

El diseño orientado a los objetos (DOO), al igual que otras metodologías de diseño orientadas a la información, crea una representación del campo del problema del mundo real y la hace corresponder con el ámbito de la solución que es el software. A diferencia de otros métodos, el DOO produce un diseño que interconecta objetos de datos y operaciones de procesamiento en una forma que modulariza la información y el procesamiento, en lugar de dejar aparte el procesamiento.

La naturaleza única del diseño orientado a los objetos queda reflejada en su capacidad de construir sobre tres pilares conceptuales importantes del diseño de software: abstracción, ocultamiento de información y modularidad. Todos los métodos de diseño intentan desarrollar software con esas tres características fundamentales, pero solo el DOO proporciona un mecanismo que permite al diseñador conseguir las tres sin complejidad ni necesidad de compromisos.

El análisis orientado a objetos, el diseño orientado a objetos y la programación orientada a objetos comprenden un conjunto de actividades de ingeniería de software para la construcción del sistema orientado a los objetos.

2.-Orientación a Objetos

Es un enfoque de desarrollo de software que organiza tanto el problema como su solución como una colección de objetos discretos; tanto la estructura de datos como el comportamiento están incluidos en la representación. Una representación orientada a objetos puede reconocerse por sus siete características:

- ✓ Identidad
- ✓ Abstracción
- ✓ Clasificación
- ✓ Encapsulamiento
- ✓ Herencia
- ✓ Polimorfismo
- ✓ Persistencia

Algunas representaciones sólo utilizan un subconjunto compuesto por parte de estas características, no obstante lo cual se les otorga la denominación de orientadas a objetos.

Identidad:

Se refiere al hecho de que los datos son organizados en entidades discretas, distinguibles, denominadas objetos. Un objeto tiene estados y comportamientos asociados. Cada objeto en un sistema orientado a objetos (OO) usualmente tiene un nombre, el cual distingue a un objeto de otros.

Abstracción:

La abstracción es esencial para construir cualquier sistema, sea OO o no. Las abstracciones en un sistema OO ayudan a representar los diferentes puntos de vista incorporados en el sistema. En conjunto, las abstracciones forman una jerarquía que muestra cómo las diferentes perspectivas de un sistema se relacionan unas con otras.

Clasificación:

En OO se utiliza la clasificación para agrupar objetos que tienen atributos y comportamientos en común. Un conjunto de estos objetos forma una clase. Se dice que un objeto es una instancia de una clase. Cada instancia tiene sus propios atributos pero comparte nombre y comportamiento con las otras instancias de la clase. Así una clase describe a 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 en particular de otro.

Encapsulamiento:

Una clase encapsula los atributos y comportamientos de un objeto, ocultando los detalles de implementación. Sin embargo, encapsulamiento no es lo mismo que ocultamiento de información, el encapsulamiento es una técnica para empaquetar la información de tal forma que se oculte lo que debe ser ocultado y se haga visible lo que esta pensado que sea visible.

Herencia:

Claramente algunos atributos son compartidos entre miembros de una clase dada, por lo que las clases pueden ser organizadas jerárquicamente de acuerdo con las semejanzas y diferencias entre ellas; esta jerarquía exhibe la estructura de herencia de clases de la OO.

Para construir la jerarquía se comienza definiendo ampliamente una clase para refinarla luego en subclases más especializadas. Una subclase puede heredar la estructura así como el comportamiento y atributos de su superclase. A veces se usa una clase abstracta para simplificar la jerarquía de manera que no se pueda definir ningún objeto de la clase abstracta si no es como una instancia de una subclase.

Polimorfismo:

Un comportamiento es una acción o transformación que un objeto realiza a la cual está sujeto. Un comportamiento de un objeto se activa por la recepción de un mensaje o por la entrada en un estado particular. A veces el mismo comportamiento se manifiesta de manera diferente en diferentes clases o subclases, propiedad a la que se denomina 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. Un lenguaje de programación OO se diseña para seleccionar automáticamente el método correcto para implementar una operación, los datos asociados con la operación como parámetros y el nombre de la clase de objeto. El polimorfismo permite agregar nuevas clases sin cambiar el código existente.

Persistencia:

Es 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 transforma cuando el objeto es transformado.

3.- Diseño de la Base de Datos

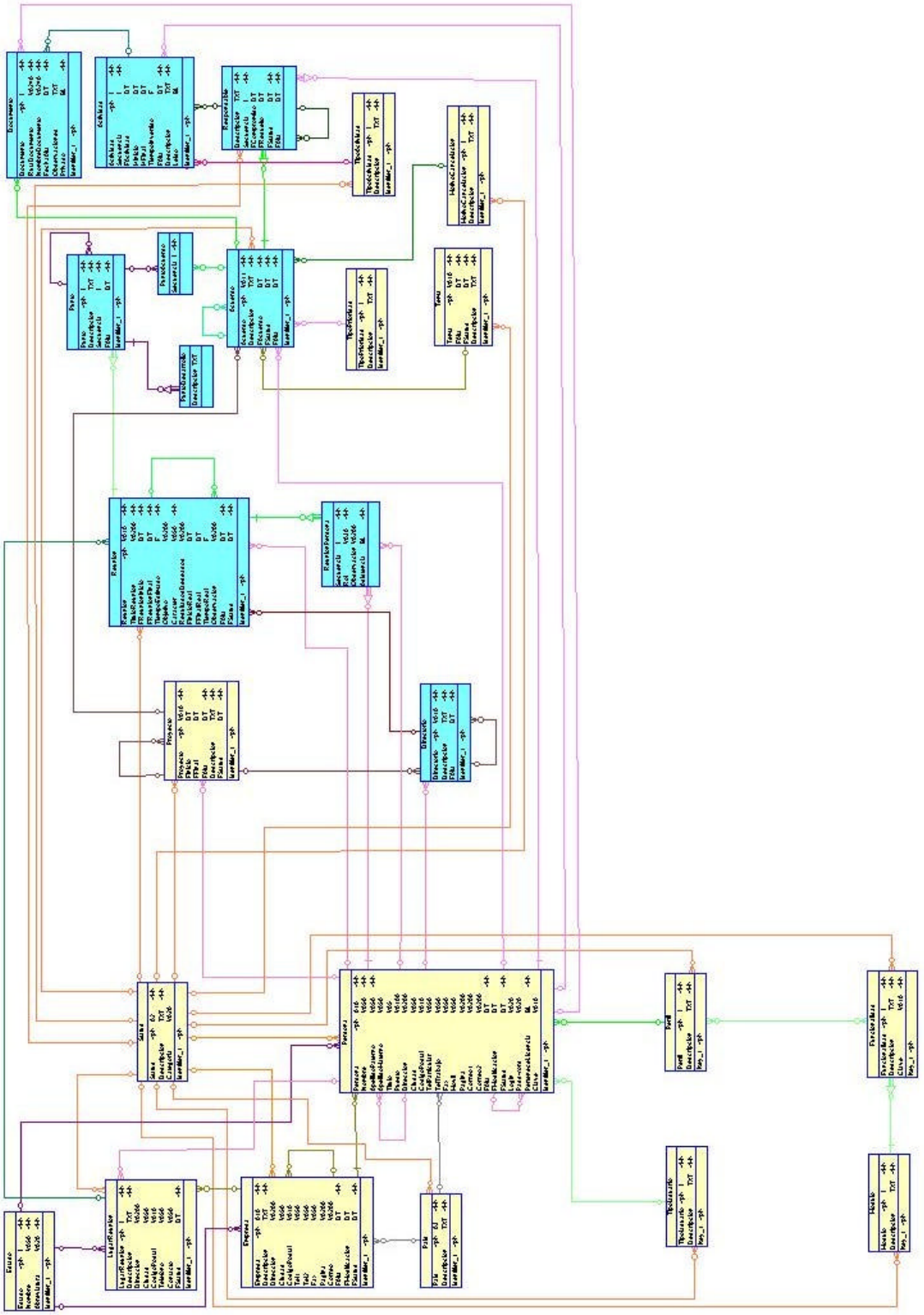
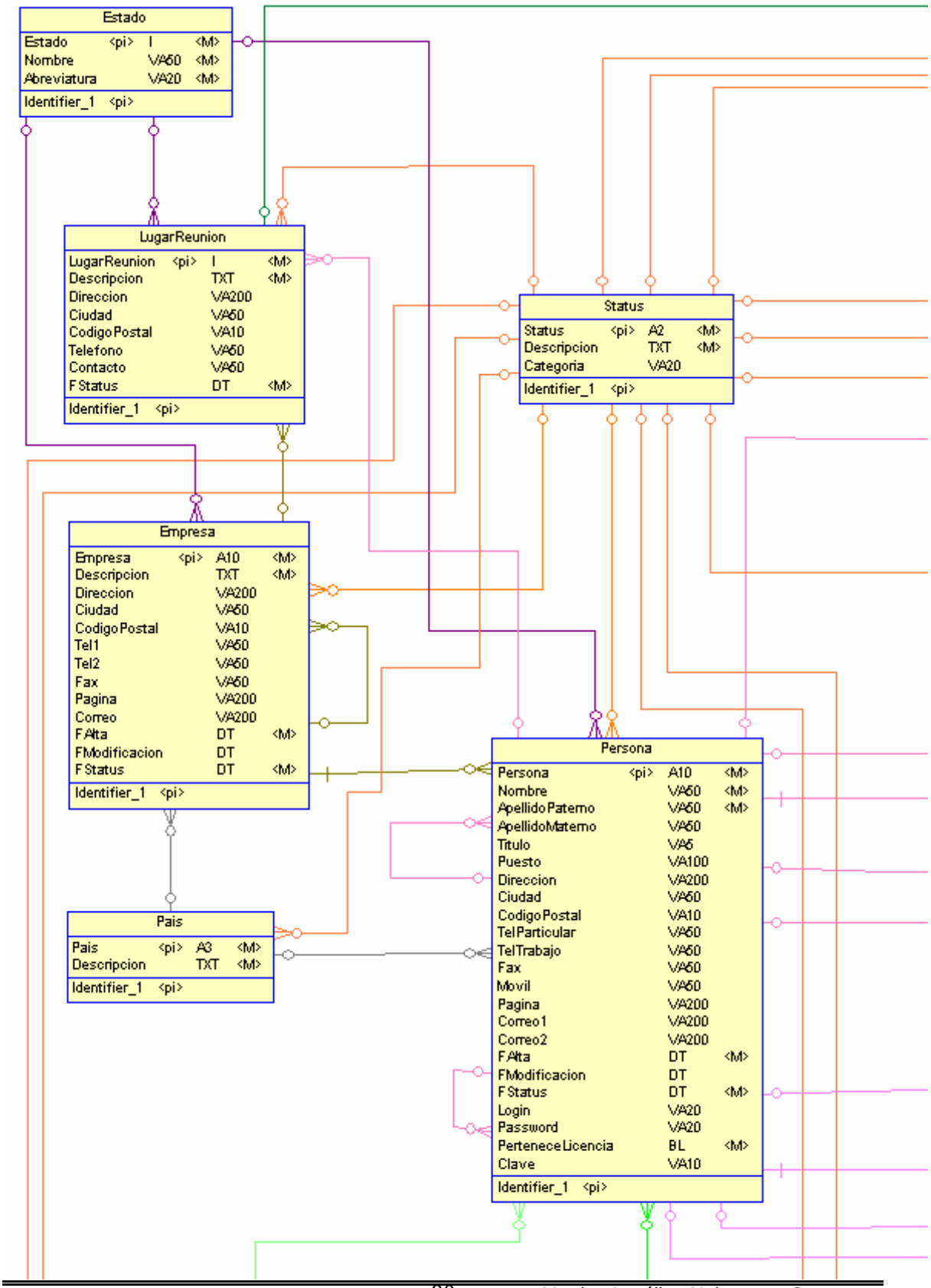
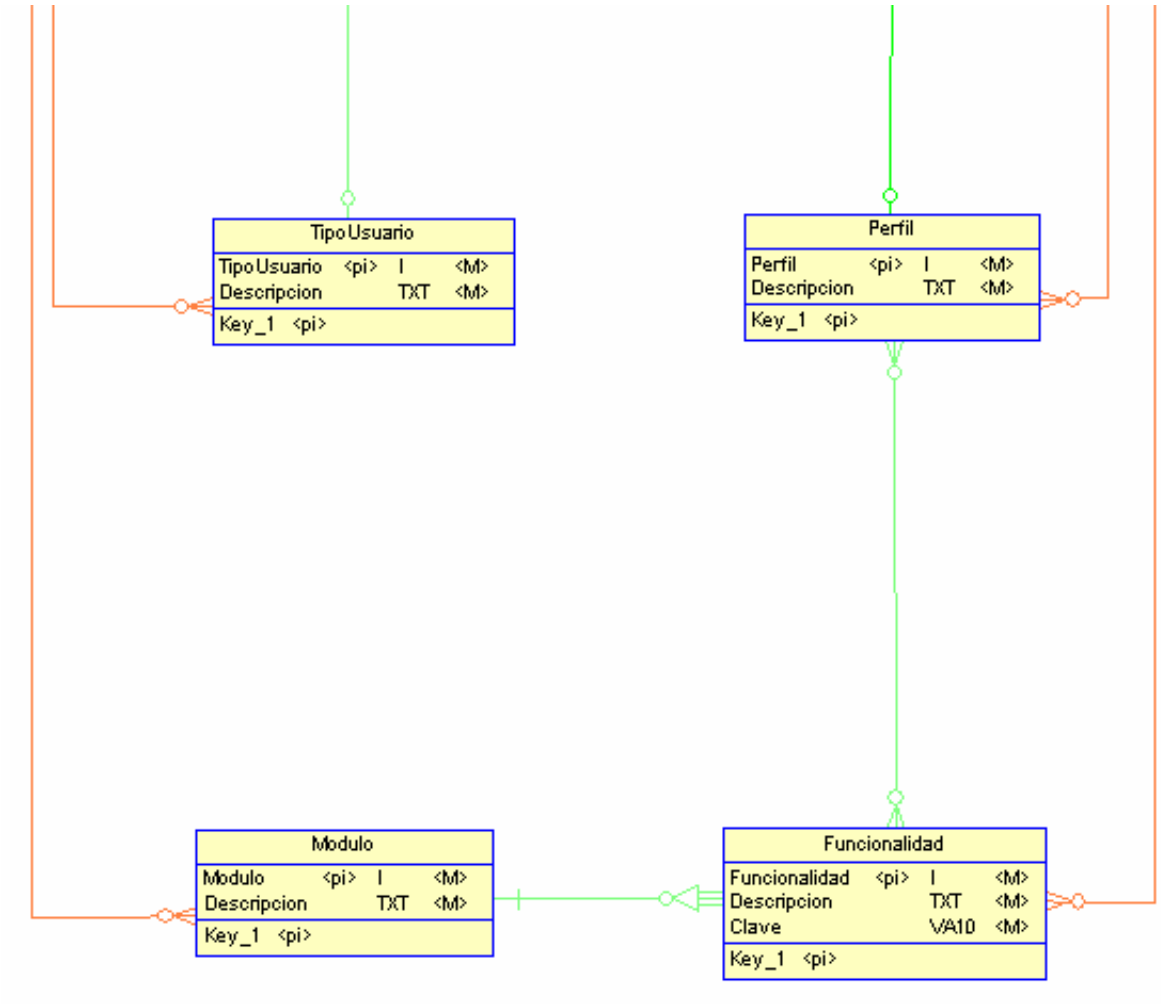
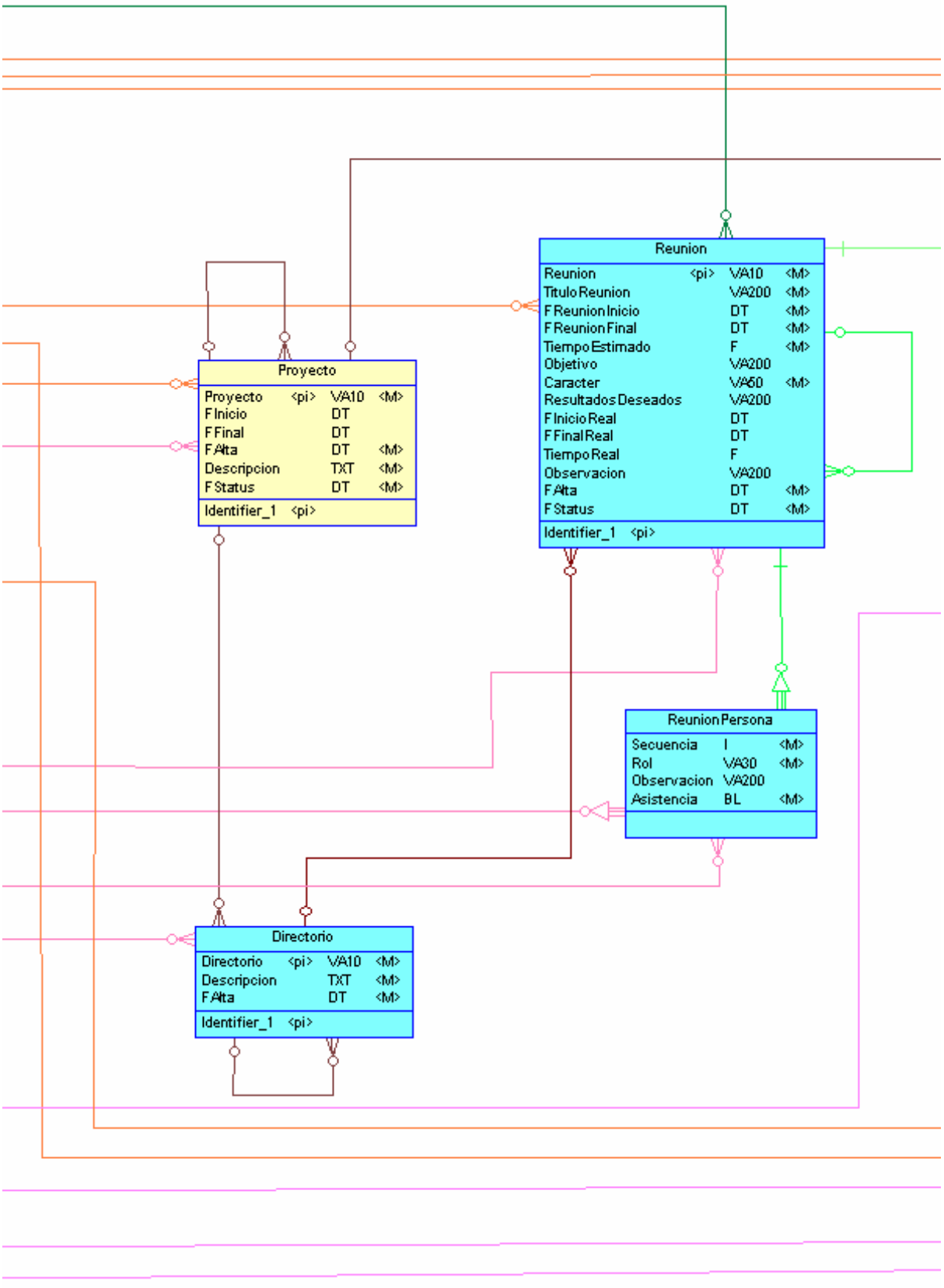


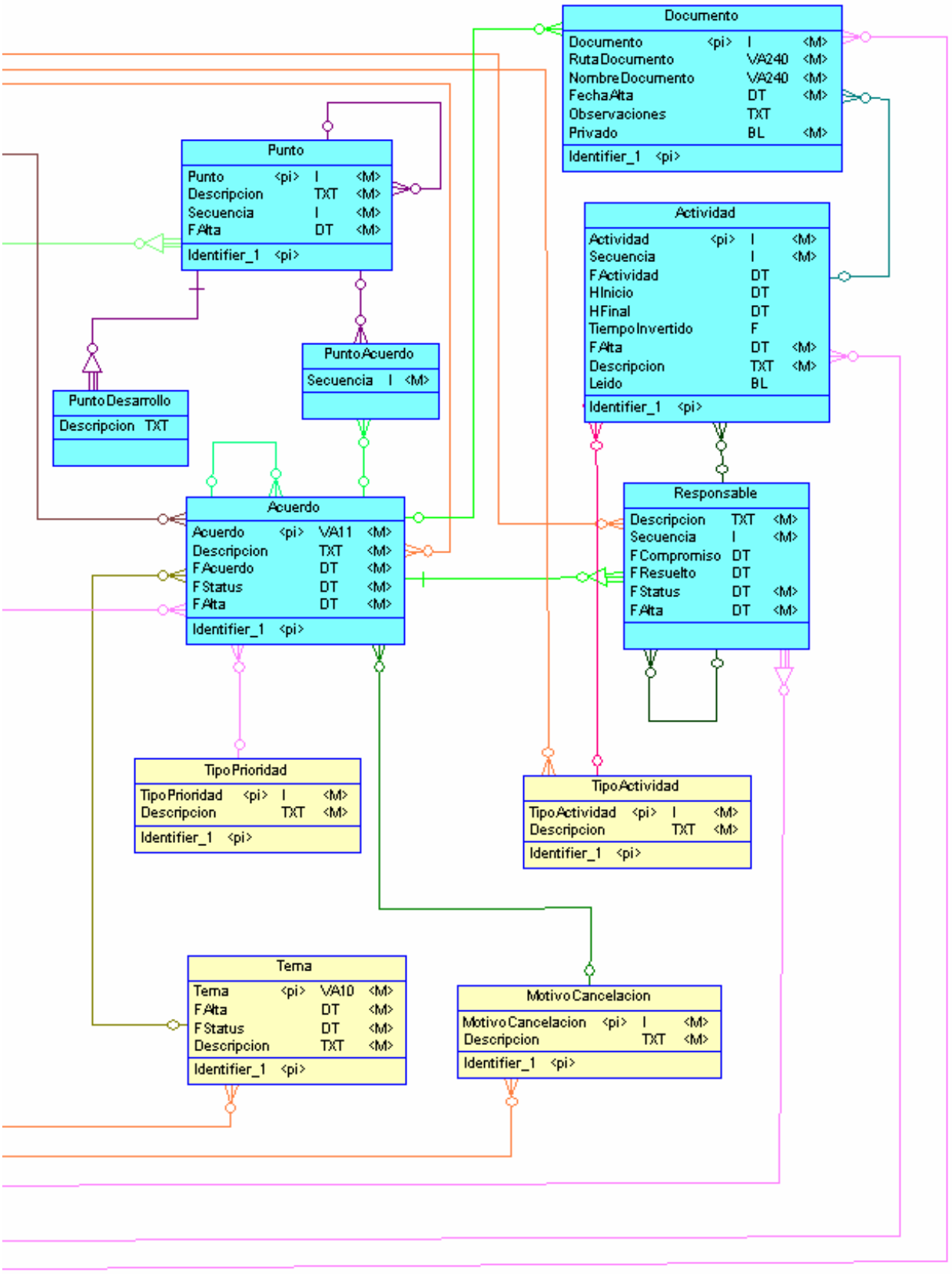
Figura 17 Diseño de la base de datos


Estado
Estado <pi> I <M>
Nombre VA50 <M>
Abreviatura VA20 <M>
Identifier_1 <pi>
LugarReunion
LugarReunion <pi> I <M>
Descripcion TXT <M>
Direccion VA200
Ciudad VA50
CodigoPostal VA10
Telefono VA50
Contacto VA50
FStatus DT <M>
Identifier_1 <pi>
Status
Status <pi> A2 <M>
Descripcion TXT <M>
Categoria VA20
Identifier_1 <pi>
Empresa
Empresa <pi> A10 <M>
Descripcion TXT <M>
Direccion VA200
Ciudad VA50
CodigoPostal VA10
Tel1 VA50
Tel2 VA50
Fax VA50
Pagina VA200
Correo VA200
FAlta DT <M>
FModificacion DT
FStatus DT <M>
Identifier_1 <pi>
Persona
Persona <pi> A10 <M>
Nombre VA50 <M>
ApellidoPaterno VA50 <M>
ApellidoMaterno VA50
Titulo VA5
Puesto VA100
Direccion VA200
Ciudad VA50
CodigoPostal VA10
TelParticular VA50
TelTrabajo VA50
Fax VA50
Movil VA50
Pagina VA200
Correo1 VA200
Correo2 VA200
FAlta DT <M>
FModificacion DT
FStatus DT <M>
Login VA20
Password VA20
PerteneceLicencia BL <M>
Clave VA10
Identifier_1 <pi>
Pais
Pais <pi> A3 <M>
Descripcion TXT <M>
Identifier_1 <pi>


TipoUsuario
TipoUsuario <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Perfil
Perfil <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Modulo
Modulo <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Funcionalidad
Funcionalidad <pi> I <M>
Descripcion TXT <M>
Clave VA10 <M>
Key_1 <pi>



Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.


Documento
Documento <pi> I <M>
RutaDocumento VA240 <M>
NombreDocumento VA240 <M>
FechaAlta DT <M>
Observaciones TXT
Privado BL <M>
Identifier_1 <pi>
Punto
Punto <pi> I <M>
Descripcion TXT <M>
Secuencia I <M>
FAlta DT <M>
Identifier_1 <pi>
Actividad
Actividad <pi> I <M>
Secuencia I <M>
FActividad DT
HInicio DT
HFinal DT
TiempoInvertido F
FAlta DT <M>
Descripcion TXT <M>
Leido BL
Identifier_1 <pi>
PuntoAcuerdo
Secuencia I <M>
PuntoDesarrollo
Descripcion TXT
Acuerdo
Acuerdo <pi> VA11 <M>
Descripcion TXT <M>
FAcuerdo DT <M>
FStatus DT <M>
FAlta DT <M>
Identifier_1 <pi>
Responsable
Descripcion TXT <M>
Secuencia I <M>
FCompromiso DT
FResuelto DT
FStatus DT <M>
FAlta DT <M>
TipoPrioridad
TipoPrioridad <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>
TipoActividad
TipoActividad <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>
Tema
Tema <pi> VA10 <M>
FAlta DT <M>
FStatus DT <M>
Descripcion TXT <M>
Identifier_1 <pi>
MotivoCancelacion
MotivoCancelacion <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>

El diseño de la base de datos completo (lógico y físico) se encuentra en la sección de anexos

Diccionario de datos

Tabla :	Estado		
Descripción :	Esta tabla es un catálogo de estados, en este caso de México		
Campo	Descripción	Tipo	Observaciones
Estado	identificador consecutivo	INT	PK
Nombre	Nombre del estado	VARCHAR(50)	NOT NULL
Abreviatura	Abreviatura del nombre del estado	VARCHAR(20)	NOT NULL

Tabla :	LugarReunion		
Descripción :	Esta tabla es un catálogo de lugares para realizar una reunión		
Campo	Descripción	Tipo	
LugarReunion	Identificador consecutivo	INT	PK
Status	Estatus del lugar para reunión, tiene relación con la tabla Status	CHAR(2)	FK
Empresa	Empresa a la que pertenece el lugar para reunión, tiene relación con la tabla Empresa	CHAR(10)	FK
PersonaAlta	Persona que da de alta el registro, tiene relación con la tabla Persona	CHAR(10)	FK
Estado	Estado en donde se encuentra el lugar para reunión, tiene relación con la tabla Estado	INT	FK
Descripcion	Descripción o nombre del lugar para reunión	TEXT	NOT NULL
Direccion	Dirección del lugar para reunión	VARCHAR(200)	NULL
Ciudad	Ciudad en donde se encuentra el lugar para reunión	VARCHAR(50)	NULL
CodigoPostal	Código postal del lugar para reunión	VARCHAR(10)	NULL
Telefono	Teléfono del lugar para reunión	VARCHAR(50)	NULL
Contacto	Datos de la persona contacto en el lugar para reunión	VARCHAR(50)	NULL
Fstatus	Ultima fecha en la que se modificó el estatus del lugar para reunión	DATE	NOT NULL

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

Tabla :	Empresa		
Descripción :	Esta tabla es un catálogo de empresas		
Campo	Descripción	Tipo	
Empresa	Clave de la empresa, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	CHAR(10)	PK
Status	Estatus de la Empresa, tiene relación con la tabla Status	CHAR(2)	FK
Estado	Estado en donde se encuentra la empresa, tiene relación con la tabla Estado	INT	FK
EmpresaPadre	Empresa de jerarquía mayor a la que pertenece la empresa, tiene relación con la tabla Empresa	CHAR(10)	FK
Pais	País en donde se encuentra la empresa, tiene relación con la tabla Pais	CHAR(3)	FK
Descripcion	Descripción o nombre de la empresa	TEXT	NOT NULL
Direccion	Dirección de la empresa	VARCHAR(200)	NULL
Ciudad	Ciudad en la que se encuentra la empresa	VARCHAR(50)	NULL
CodigoPostal	Código postal de la empresa	VARCHAR(10)	NULL
Tel1	Teléfono principal de la empresa	VARCHAR(50)	NULL
Tel2	Teléfono secundario de la empresa	VARCHAR(50)	NULL
Fax	Número de fax de la empresa	VARCHAR(50)	NULL
Pagina	Dirección de la página web de la empresa	VARCHAR(200)	NULL
Correo	Correo electrónico de la empresa	VARCHAR(200)	NULL
FAlta	Fecha en la que se da de alta la empresa	DATE	NOT NULL
FModificacion	Última fecha de modificación de los datos de la empresa	DATE	NULL
FStatus	Última fecha en la que se modificó el estatus de la empresa	DATE	NOT NULL

Tabla :	Pais		
Descripción :	Esta tabla es un catálogo de paises		
Campo	Descripción	Tipo	
Pais	Abreviatura internacional del país	CHAR(3)	PK
Status	Estatus del país	CHAR(2)	FK
Descripcion	Nombre completo del país	TEXT	NOT NULL

Tabla :	TipoUsuario		
Descripción :	Esta tabla es un catálogo de los tipos de usuario que pueden existir		
Campo	Descripción	Tipo	
TipoUsuario	Identificador consecutivo	INT	PK
Status	Estatus del tipo de usuario, tiene relación con la tabla Status	CHAR(2)	FK
Descripcion	Descripción del tipo de usuario	TEXT	NOT NULL

Tabla :	Modulo		
Descripción :	Esta tabla es un catálogo de los módulos en los que se divide el sistema		
Campo	Descripción	Tipo	
Modulo	Identificador consecutivo	INT	PK
Status	Estatus del módulo	CHAR(2)	FK
Descripcion	Descripción o nombre del módulo	TEXT	NOT NULL

Tabla :	Funcionalidad		
Descripción :	Esta tabla es un catálogo de las funcionalidades existentes en cada módulo del sistema		
Campo	Descripción	Tipo	
Modulo	Módulo al que pertenece la funcionalidad, tiene relación con la tabla Modulo	INT	PK, FK
Funcionalidad	Identificador consecutivo	INT	PK
Status	Estatus de la funcionalidad, tiene relación con la tabla Status	CHAR(2)	FK
Descripcion	Descripción o nombre de la funcionalidad	TEXT	NOT NULL
Clave	Clave de la funcionalidad, generada de la siguiente manera: 2 primeros caracteres para abreviatura del módulo, caracteres intermedios para la abreviatura de la funcionalidad y 2 últimos caracteres para la abreviatura de la acción que realiza (ej. CTEMCN CaTalogos, EMpresa, CoNsultar)	VARCHAR(10)	NOT NULL

Tabla :	Perfil		
Descripción :	Esta tabla es un catálogo de los perfiles existentes		
Campo	Descripción	Tipo	
Perfil	Identificador consecutivo	INT	PK
Status	Estatus del perfil	CHAR(2)	FK
Descripcion	Descripción o nombre del perfil	TEXT	NOT NULL

Tabla :	PerfilFuncionalidad		
Descripción :	Tabla para asignar funcionalidades a los perfiles		
Campo	Descripción	Tipo	
Perfil	Perfil al que se asigna la funcionalidad	INT	PK, FK
Modulo	Módulo al que pertenece la funcionalidad	INT	PK, FK
Funcionalidad	Funcionalidad que se asigna a un perfil	INT	PK, FK

Tabla :	Status		
Descripción :	Esta tabla es un catálogo de los estatus existentes		
Campo	Descripción	Tipo	
Status	Abreviatura del estatus	CHAR(2)	PK
Descripcion	Descripción o nombre del estatus	TEXT	NOT NULL
Categoria	Categoría para el criterio de aplicación del estatus, las categorías son: Catálogo, Acuerdo y Reunión	VARCHAR(20)	NULL

Tabla :	Proyecto		
Descripción :	Esta tabla es un catálogo de los proyectos existentes		
Campo	Descripción	Tipo	
Proyecto	Clave del proyecto, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	VARCHAR(10)	PK
PersonaAlta	Persona que da de alta el registro, tiene relación con la tabla Persona	CHAR(10)	FK
ProyectoPadre	Proyecto de jerarquía mayor del que se desprende el proyecto, tiene relación con la tabla Proyecto	VARCHAR(10)	FK
Status	Estatus del proyecto, tiene relación con la tabla Status	CHAR(2)	FK
FInicio	Fecha en la que inicia el proyecto	DATE	NULL
FFinal	Fecha en la que finaliza el proyecto	DATE	NULL
FAlta	Fecha en la que se da de alta el proyecto	DATE	NOT NULL
Descripcion	Descripción o nombre del proyecto	TEXT	NOT NULL
FStatus	Última fecha de modificación del estatus del proyecto	DATE	NOT NULL

Tabla :	Persona		
Descripción :	Esta tabla es un catálogo de personas existentes		
Campo	Descripción	Tipo	
Persona	Clave de la persona, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	CHAR(10)	PK
Empresa	Empresa a la que pertenece la persona, tiene relación con la tabla Empresa	CHAR(10)	FK
Pais	País de origen de la persona, tiene relación con la tabla Pais	CHAR(3)	FK
Perfil	Perfil asignado a la persona, tiene relación con la tabla Perfil	INT	FK
PersonaAlta	Persona que da de alta el registro, tiene relación con la tabla Persona	CHAR(10)	FK
Estado	Estado de origen de la persona, tiene relación con la tabla Estado	INT	FK
TipoUsuario	Tipo de usuario asignado a la persona, tiene relación con la tabla TipoUsuario	INT	FK
Status	Estatus de la persona, tiene relación con la tabla Status	CHAR(2)	FK
PersonaPadre	Persona de quien depende en jerarqía la persona , tiene relación con la tabla Persona	CHAR(10)	FK
Nombre	Nombre de la persona	VARCHAR(50)	NOT NULL
ApellidoPaterno	Apellido paterno de la persona	VARCHAR(50)	NOT NULL
ApellidoMaterno	Apellido materno de la persona	VARCHAR(50)	NULL
Titulo	Titulo o tratamiento de la persona (ej. Ing , Lic, Dr, etc.)	VARCHAR(5)	NULL
Puesto	Puesto que desempeña la persona	VARCHAR(100)	NULL
Direccion	Dirección de la persona	VARCHAR(200)	NULL
Ciudad	Ciudad de residencia de la persona	VARCHAR(50)	NULL
CodigoPostal	Código postal de la persona	VARCHAR(10)	NULL
TelParticular	Teléfono particular de la persona	VARCHAR(50)	NULL
TelTrabajo	Teléfono del trabajo de la persona	VARCHAR(50)	NULL
Fax	Fax de la persona	VARCHAR(50)	NULL
Movil	Teléfono móvil de la persona	VARCHAR(50)	NULL
Pagina	Pagina web de la persona	VARCHAR(200)	NULL
Correo1	Correo principal de la persona	VARCHAR(200)	NULL

Correo2	Correo secundario de la persona	VARCHAR(200)	NULL
FAlta	Fecha en que se da de alta la persona	DATE	NOT NULL
FModificacion	Ultima fecha de modificación de los datos de la persona	DATE	NULL
FStatus	Última fecha de modificación del estatus de la persona	DATE	NOT NULL
Login	Nombre de usuario o login para el sistema	VARCHAR(20)	NULL
Password	Password o contraseña para el sistema	VARCHAR(20)	NULL
Clave	Clave o número de empleado de la persona	VARCHAR(10)	NULL

Tabla :	Reunión		
Descripción :	Tabla en la que se registran las reuniones		
Campo	Descripción	Tipo	
Reunion	Clave dela reunión, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	VARCHAR(10)	PK
PersonaAlta	Persona que da de alta el registro y es por lo tanto la dueña de la reunión, tiene relación con la tabla Persona	CHAR(10)	FK
Status	Estatus de la reunión, tiene relación con la tabla Status	CHAR(2)	FK
LugarReunion	Lugar en el que se lleva a cabo la reunión, tiene relación con la tabla LugarReunion	INT	FK
ReunionPadre	Reunión de jerarquía superior de la cual se desprende la reunión, tiene relación con la tabla Reunion	VARCHAR(10)	FK
Directorio	Directorio en el que se encuentra la reunión, tiene realción con la tabla Directorio	VARCHAR(10)	FK
TituloReunion	Título o nombre de la reunión	VARCHAR(200)	NOT NULL
FReunionInicio	Fecha y hora en la que se planea iniciará la reunión	DATETIME / TIMESTAMP	NOT NULL NOT NULL
FReunionFinal	Fecha y hora en la que se planea finalizará la reunión	DATETIME / TIMESTAMP	NOT NULL
TiempoEstimado	Tiempo total estimado de duración de la reunión	FLOAT	NOT NULL

Objetivo	Objetivo de la reunión	VARCHAR(200)	NULL
Caracter	Carácter de la reunión, este puede ser: Ordinaria o Extraordinaria	VARCHAR(50)	NOT NULL
ResultadosDeseados	Resultados que se desean obtener al finalizar la reunión	VARCHAR(200)	NULL
FInicioReal	Fecha y hora real de inicio de la reunión	DATETIME / TIMESTAMP	NULL
FFinalReal	Fecha y hora real de termino de la reunión	DATETIME / TIMESTAMP	NULL
TiempoReal	Tiempo total real de duración de la reunión	FLOAT	NULL
Observacion	Observaciones generales de la reunión	VARCHAR(200)	NULL
FAlta	Fecha en la que se da de alta la reunión	DATE	NOT NULL
FStatus	Última fecha de modificación del estatus de la reunión	DATE	NOT NULL

Tabla :	Directorio		
Descripción :	Tabla en la que se registran los directorios de cada usuario		
Campo	Descripción	Tipo	
Directorio	Clave del directorio, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	VARCHAR(10)	PK
Proyecto	Proyecto al que con el que se relaciona el directorio, tiene relación con la tabla Proyecto	VARCHAR(10)	FK
DirectorioPadre	Directorio de mayor jerarquía dentro del que se encuentra el directorio	VARCHAR(10)	FK
PersonaAlta	Persona que da de alta el registro y es la dueña del directorio	CHAR(10)	FK
Descripcion	Descripción o nombre del directorio	TEXT	NOT NULL
FAlta	Fecha en la que se da de alta el directorio	DATE	NOT NULL

Tabla :	ReunionPersona		
Descripción :	Tabla en la que se registran los asistentes a la reunión		
Campo	Descripción	Tipo	
Reunion	Es la reunión a la que se le registra la asistencia	VARCHAR(10)	PK, FK

Persona	Persona que se invita o convoca a la reunión, tiene relación con la tabla Persona	CHAR(10)	PK, FK
PersonaSuplente	Persona que asiste como suplente de la persona convocada, tiene relación con la tabla Persona	CHAR(10)	FK
Secuencia	Secuencia u orden de los asistentes	INT	NOT NULL
Rol	Rol que desempeña la persona, puede ser: Convocante o Convocado	VARCHAR(30)	NOT NULL
Observacion	Observaciones de o para la persona	VARCHAR(200)	NULL
Asistencia	Registro de asistencia o falta a la reunión	BIT	NOT NULL

Tabla :	Punto		
Descripción :	Tabla en la que se registran los puntos de la agenda de la reunión		
Campo	Descripción	Tipo	
Reunion	Reunión a la que se le asigna el punto, tiene relación con la tabla Reunión	VARCHAR(10)	PK, FK
Punto	Número de punto consecutivo en la reunión	INT	PK
ReunionPadre	Punto de mayor jerarquía del que se desprende el punto, tiene relación con la tabla Punto	VARCHAR(10)	FK
PuntoPadre		INT	FK
Descripcion	Descripción breve o nombre del punto	VARCHAR(200)	NOT NULL
Secuencia	Secuencia u orden de los puntos	INT	NOT NULL
FAlta	Fecha en la que se da de alta el punto	DATE	NOT NULL

Tabla :	PuntoDesarrollo		
Descripción :	Tabla en la que se registra la narrativa del punto		
Campo	Descripción	Tipo	
Reunion	Punto al que se le registra la narrativa, tiene relación con la tabla Punto	VARCHAR(10)	PK, FK
Punto		INT	PK, FK
Descripcion	Narrativa o descripción detallada del punto	TEXT	NOT NULL

Tabla :	PuntoAcuerdo		
Descripción :	Tabla en la que se registra el punto al que pertenece un acuerdo		
Campo	Descripción	Tipo	
Reunion	Punto dentro del que esta un acuerdo, tiene relación con la tabla Punto	VARCHAR(10)	PK, FK
Punto		INT	PK, FK
Acuerdo	Acuerdo que se asigna al punto	VARCHAR(11)	PK, FK
Secuencia	Secuencia u orden de los acuerdos	INT	NOT NULL

Tabla :	Acuerdo		
Descripción :	Tabla en la que se registran los acuerdos		
Campo	Descripción	Tipo	
Acuerdo	Clave del acuerdo, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 6 últimos dígitos para un consecutivo	VARCHAR(11)	PK
Prioridad	Prioridad de cumplimiento del acuerdo, tiene relación con la tabla TipoPrioridad	INT	FK
Tema	Tema con el que esta relacionado el acuerdo, tiene relación con la tabla Tema	VARCHAR(10)	FK
Status	Estatus del acuerdo	CHAR(2)	FK
Proyecto	Proyecto al que esta relacionado el acuerdo, tiene relación con la tabla Proyecto	VARCHAR(10)	FK
MotivoCancelacion	Motivo por el cual se cancela el acuerdo, tiene relación con la tabla MotivoCancelacion	INT	FK
AcuerdoPadre	Acuerdo de jerarquía mayor del que se deriva el acuerdo, tiene relación con la tabla Acuerdo	VARCHAR(11)	FK
PersonaAlta	Persona que da de alta el registro y es la solicitante del acuerdo	CHAR(10)	FK
Descripcion	Descripción detallada del acuerdo	TEXT	NOT NULL
FAcuerdo	Fecha en la que se realiza el acuerdo	DAT	NOT NULL
FStatus	Última fecha de modificación del estatus	DATE	NOT NULL
FAlta	Fecha en la que se da de alta el acuerdo	DATE	NOT NULL

Tabla :	Tema		
Descripción :	Esta tabla es un catálogo de temas		
Campo	Descripción	Tipo	
Tema	Clave del tema, generada de la siguiente manera: 5 primeros dígitos para el número de licencia y 5 últimos dígitos para un consecutivo	VARCHAR(10)	PK
Status	Estatus del tema, tiene relación con la tabla Status	CHAR(20)	FK
FAlta	Fecha de alta del tema	DATE	NOT NULL
FStatus	Última fecha de modificación del estatus del tema	DATE	NOT NULL
Descripcion	Descripción o nombre del tema	VARCHAR(200)	NOT NULL

Tabla :	TipoPrioridad		
Descripción :	Esta tabla es un catálogo de los tipos de prioridades		
Campo	Descripción	Tipo	
TipoPrioridad	Identificador consecutivo	INT	PK
Descripcion	Descripción o nombre del tipo de prioridad	VARCHAR(50)	NOT NULL

Tabla :	MotivoCancelacion		
Descripción :	Esta tabla es un catálogo de los motivos de cancelación		
Campo	Descripción	Tipo	
MotivoCancelacion	Identificador consecutivo	INT	PK
Status	Estatus del motivo de cancelación	CHAR(2)	FK
Descripcion	Descripción o nombre del motivo de cancelación	VARCHAR(100)	NOT NULL

Tabla :	Responsable		
Descripción :	Tabla para registrar los responsables de cumplir con un acuerdo		
Campo	Descripción	Tipo	
Acuerdo	Acuerdo del cual es responsable la persona, tiene relación con la tabla Acuerdo	VARCHAR(11)	
Persona	Persona a la que se le solicita cumplir el acuerdo, tiene relación con la tabla Persona	CHAR(10)	
Status	Estatus del responsable, tiene relación con la tabla Status	CHAR(2)	

AcuerdoPadre	Responsable de mayor jerarquía que delega el acuerdo al responsable, tiene relación con la tabla Resoponsable	VARCHAR(11)	
PersonaPadre		CHAR(10)	
Descripcion	Descripción o instrucción específica para el responsable	TEXT	
Secuencia	Secuencia u orden de los responsables	INT	
FCompromiso	Fecha para la que el responsable tiene que cumplir con el acuerdo	DATE	
FResuelto	Fecha en la que el responsable resuelve el acuerdo	DATE	
FStatus	Última fecha de modificación del estatus del responsable	DATE	
FAlta	Fecha de alta del responsable	DATE	

Tabla :	Actividad		
Descripción :	Tabla donde se registra las actividades o avances de los responsables para cumplir con su acuerdo		
Campo	Descripción	Tipo	
Actividad	Identificador consecutivo	INT	PK
TipoActividad	Tipo de actividad para clasificación de la actividad, tiene relación con la tabla TipoActividad	INT	FK
Persona	Responsable que realiza la actividad, tiene relación con la tabla Responsable	CHAR(10)	FK
Acuerdo		VARCHAR(11)	FK
Secuencia	Secuencia u orden de las actividads	INT	NOT NULL
FActividad	Fecha en la que se realiza la actividad	DATE	NULL
HInicio	Hora en la que se inicia la actividad	TIME	NULL
HFinal	Hora en la que se termina la actividad	TIME	NULL
TiempoInvertido	Tiempo que se utiliza para la actividad	FLOAT	NULL
FAlta	Fecha de alta de la actividad	DATE	NOT NULL
Descripcion	Descripción o detalle de la actividad	TEXT	NOT NULL
Leido	Criterio para indicar si ya se ha enterado el solicitante de la actividad realizada por el responsable	BIT	NOT NULL

Tabla :	TipoActividad		
Descripción :	Esta tabla es un catálogo de los tipos de actividades		
Campo	Descripción	Tipo	
TipoActividad	Identificador consecutivo	INT	PK
Status	Estatus del tipo de actividad, tiene relación con la tabla Status	CHAR(2)	FK
Descripcion	Descripción o nombre del tipo de actividad	VARCHAR(50)	NOT NULL

Tabla :	Documento		
Descripción :	Tabla para registrar los documentos o archivos adjuntos a un acuerdo o actividad		
Campo	Descripción	Tipo	
Documento	Identificador consecutivo	INT	PK
Actividad	Actividad a la que se adjunta el documento, tiene relación con la tabla Actividad	INT	FK, NULL
Acuerdo	Acuerdo al que se adjunta el documento, tiene relación con la tabla Acuerdo	VARCHAR(11)	FK, NULL
RutaDocumento	Ruta en donde se guarda el documento	VARCHAR(240)	NOT NULL
NombreDocumento	Nombre con el que se visualiza el documento	VARCHAR(240)	NOT NULL
FechaAlta	Fecha de alta del documento	DATE	NOT NULL
Observaciones	Observaciones al documento	TEXT	NULL
Privado	Criterio para acceso al documento	BIT	NOT NULL
Persona	Persona que adjunta el documento o dueña del archivo, tiene relación con la tabla Persona	CHAR(10)	FK

4.-Elección de tecnologías

Cuando se considera como un paso del proceso de la ingeniería de software, la codificación es una consecuencia natural del diseño. Sin embargo, las características del lenguaje de programación y el estilo de programación pueden afectar profundamente a la calidad y al mantenimiento del software.

Un planteamiento de ingeniería de software sobre las características de los lenguajes de programación se centra en las necesidades que puede tener un proyecto específico de desarrollo de software. Aunque se podrían derivar esotéricos requisitos para el código fuente, podemos establecer un conjunto general de características de ingeniería :

- ✓ Facilidad de traducción del diseño al código
- ✓ Eficiencia del compilador
- ✓ Portabilidad del código fuente
- ✓ Disponibilidad de herramientas de desarrollo
- ✓ Facilidad de mantenimiento

El *grado de facilidad* de la traducción del diseño al código proporciona una indicación de cómo se aproxima un lenguaje de programación a la representación del diseño.

La *portabilidad del código fuente* es una característica de los lenguajes de programación que se puede interpretar de tres formas:

1. El código fuente puede ser transportado de un procesador a otro y de un compilador a otro sin ninguna o muy pocas modificaciones.
2. El código fuente permanece inalterado cuando cambia su entorno de funcionamiento, por ejemplo cuando se actualiza el sistema operativo.
3. El código fuente puede ser integrado en diferentes paquetes de software sin que prácticamente se requieran modificaciones debidas a las características propias del lenguaje de programación.

La *disponibilidad de herramientas de desarrollo* puede acortar el tiempo requerido para la generación del código fuente y puede mejorar la calidad del código.

La *facilidad de mantenimiento del código fuente* es críticamente importante para cualquier esfuerzo no trivial de desarrollo de software. El mantenimiento no se puede llevar a cabo hasta que no se entiende el software.

Anteriores elementos de configuración de software proporcionan un fundamento para la facilidad de comprensión, ya que el código fuente final debe ser leído y modificado de acuerdo con los cambios en el diseño. Además las propias características de documentación de un lenguaje tienen una fuerte influencia sobre el mantenimiento.

5.-Elección del lenguaje de programación

La elección del lenguaje de programación es un paso importante ya que determinará las posibilidades y el desempeño del sistema. Por ser este un sistema WEB existen actualmente dos tecnologías importantes con las que se puede desarrollar, Active Server Pages (ASP) de Microsoft y Java Server Pages (JSP) de Sun Microsystems. Dado que ambas tecnologías tienen una gran demanda en el mercado, se tiene que hacer una comparación entre ellas para tomar la decisión más acertada.

Comparación entre Java Server Pages (JSP) y Active Server Pages (ASP)

A primera vista, las tecnologías JSP y ASP tienen muchas similitudes, ambas están diseñadas para crear páginas interactivas como parte de una aplicación basada en WEB, pero también existen muchas diferencias.

La tecnología JSP está diseñada para ser independiente tanto de la plataforma como del servidor en contraste con la tecnología ASP que es una tecnología de Microsoft que se basa principalmente en las tecnologías Microsoft.

La tecnología JSP se basa en la filosofía de *"Write Once, Run Anywhere"* de la arquitectura Java, por lo que la tecnología JSP puede correr en cualquier servidor WEB y es apoyado por una gran variedad de herramientas de distintos proveedores.

La tecnología ASP está básicamente restringida a las plataformas basadas en Microsoft Windows, la tecnología ASP no funciona fácilmente en un gran número de servidores WEB porque los objetos ActiveX son específicos de la plataforma.

Aunque la tecnología ASP esta disponible en otras plataformas a través de productos *"third-party porting",* para acceder a componentes e interactuar con otros servicios, los objetos ActiveX deben estar presentes en la plataforma seleccionada, si no, se requiere de una plataforma puente que los soporte.

La tecnología JSP utiliza el lenguaje Java, mientras que ASP usa VBScript o Jscript. El lenguaje Java es maduro, poderoso y un lenguaje escalable de programación que proporciona muchos beneficios sobre los lenguajes de script basados en BASIC. Los lenguajes de script son buenos para las aplicaciones pequeñas, pero no se escalan bien al manejar aplicaciones más grandes y mas complejas. Debido a que el lenguaje Java es orientado a objetos, es mas fácil construir y mantener aplicaciones más grandes y modulares con el.

El énfasis de la tecnología JSP en los componentes sobre los scripts hace más fácil modificar contenido sin afectar la lógica o modificar la lógica sin cambiar el contenido. La arquitectura Enterprise Java Beans encapsula la lógica del negocio, como el acceso a las bases de datos, seguridad e integridad de transacciones, y las separa de la aplicación misma.

Dado que la tecnología JSP es una arquitectura *"cross-platform"* abierta, los servidores WEB, las plataformas y otros componentes pueden ser fácilmente actualizados o cambiados sin afectar las aplicaciones basadas en JSP. Esto hace a JSP mas recomendable para aplicaciones en mundo real, donde el cambio constante y el crecimiento es la norma. Las paginas JSP Tienen acceso a todos los servicios del J2EE Standard, como son:

- ✓ Java Naming and Directory Interface API
- ✓ JDBC API
- ✓ JavaMail
- ✓ Java Message Service (JMS)

En resumen:

	Tecnología ASP	Tecnología JSP
Servidor WEB	IIS de Microsoft o Personal WEB Server	Cualquier servidor WEB incluyendo Apache, Netscape, IIS, etc.
Plataformas	Windows de Microsoft	Las plataformas más populares incluyendo Solaris, Windows, Mac OS, Linux y otras plataformas UNIX
Reutilizable, Componentes Cross-Platform	NO	Si
Seguridad contra System Crashes	NO	SI
Protección contra desbordamiento de memoria	NO	SI
Lenguaje de script	VBScript, JScript	Java
Etiquetas personalizables	NO	SI
Compatible con Legacy Databases Habilidad de integrarse con Data Sources	Si (COM) Trabaja con cualquier base de datos ODBC	SI (Usando el API JDBC) Trabaja con cualquier base de datos ODBC o JDBC
Componentes	Componentes COM	JavaBeans, Enterprise JavaBeans o etiquetas extensibles JSP
Soporte extensivo de herramientas	SI	SI

Figura 18 Tabla de comparación JSP vs ASP

Dada la comparación anterior y con base en las necesidades específicas del sistema se decidió utilizar la tecnología de Sun Microsystems JAVA SERVER PAGES (JSP).

6.-Elección de Manejador de Bases de datos

Al igual que el lenguaje de programación, la selección del manejador de bases de datos es muy importante ya que en la base de datos estará toda la información del sistema.

Existen diferentes manejadores de Bases de datos e idealmente el sistema debe ser capaz de adaptarse a cualquiera de estos, pero se debe elegir uno para el desarrollo del sistema y posteriormente se harán pruebas con otros.

SQL Server y PostgreSQL

Después de hacer un análisis entre los manejadores más populares, se encontró que SQL Server es utilizado por varias grandes empresas las cuales son clientes potenciales del sistema además de que por sus características cubre las necesidades del sistema y tiene amplio soporte, por lo que se decidió que el sistema se desarrollara inicialmente con este manejador.

SQL Server es un gestor de base de datos relacionales que utiliza la arquitectura cliente / servidor. Es un entorno de trabajo sobre servidores Windows, siendo sus principales armas la rapidez, la integración con el sistema operativo y una gran facilidad de administración del sistema.

Una de las principales cualidades de SQL Server es aprovechar al máximo las posibilidades del sistema operativo, característica que hace que los rendimientos que obtiene SQL Server en Windows sean muy superiores a sus competidores.

Sin embargo, SQLServer requiere la compra de una licencia, lo cual para los clientes potenciales que no la tienen implicaría un gasto extra, por este motivo se buscó un manejador libre y se decidió utilizar Postgre SQL para estos casos, para probar que el sistema se adaptará sin problemas a otro manejador y para los casos en los que el sistema operativo no sea Windows.

CAPITULO 5 DESARROLLO

Todos los pasos de la ingeniería de software que se han realizado, van dirigidos hacia un objetivo final: traducir las representaciones de software a una forma que pueda ser *"comprendida"* por la computadora. Se ha llegado al paso de la codificación, es decir, el proceso que transforma el diseño en un lenguaje de programación.

1.-Lenguaje de desarrollo

El lenguaje JAVA

Es un lenguaje de programación de alto nivel, estandarizado, que permite escribir programas para la plataforma JAVA.

La plataforma Java.

Es un conjunto de programas, corriendo sobre una plataforma de hardware, que proveen un ambiente en donde un programa se ejecutará (*programas corriendo programas*).

Se compone de:

- La Maquina Virtual de Java (*Java Virtual Machine,JVM*)
  - Intérprete
  - Conjunto de API's (*Aplication Programable Interface*)

- Plataforma de desarrollo
  - Compilador
  - Depurador
  - Generador de documentación
  - Compresor de archivos

Esquema de la plataforma Java

Se programa en lenguaje de alto nivel (Java), al compilar se genera un *"código objeto"* llamado *bytecode* que es tomado por el intérprete y ejecutado en la maquina anfitrión. El *bytecode* es un formato estándar, puede ser ejecutado en cualquier plataforma que implemente la*JVM.*

Todas las implementaciones de la *JVM* contienen el *Java Runtime Enviroment (JRE)* que incluye el intérprete y una *API* base, si el programa ocupa funcionalidades extras (proporcionadas por otras clases) a esta última es necesario llevarlas también.

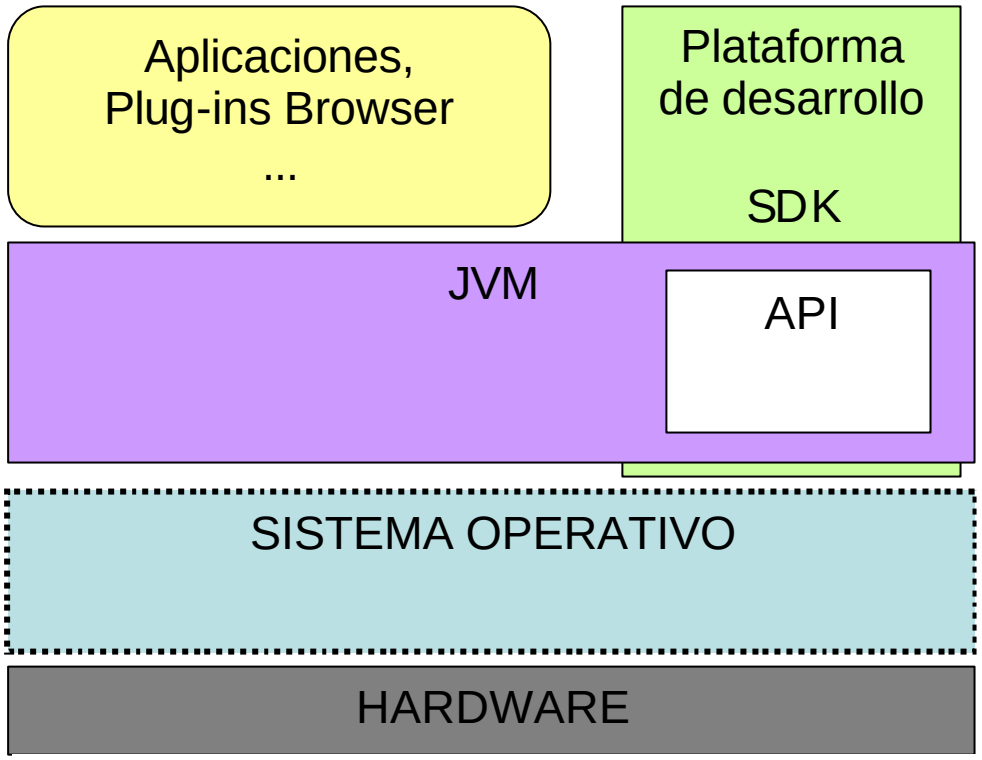


Figura 19 Plataforma JAVA

Orientado a Objetos, Portátil y Arquitectura Neutral, Seguro.

- Enfoque en los datos y la manipulación de los mismos.
- A excepción de ocho tipos primitivos, todo en Java es un objeto.
- El *bytecode* es un formato estándar, puede ser ejecutado en cualquier plataforma que implemente la *JVM.*
- Los *bytecodes* pueden ser ejecutados en distintas plataformas, no necesariamente nuestros programas.
- La *JVM* realiza distintas verificaciones durante las distintas etapas de una aplicación para hacerlo mas seguro.
  - No se permite acceso directo a memoria.
  - Verificación de limites de arreglos, cadenas y estructuras dinámicas.
- Asignación de memoria, destrucción de objetos y recolección de basura automática.
  - En Java no se manejan apuntadores de manera explicita.

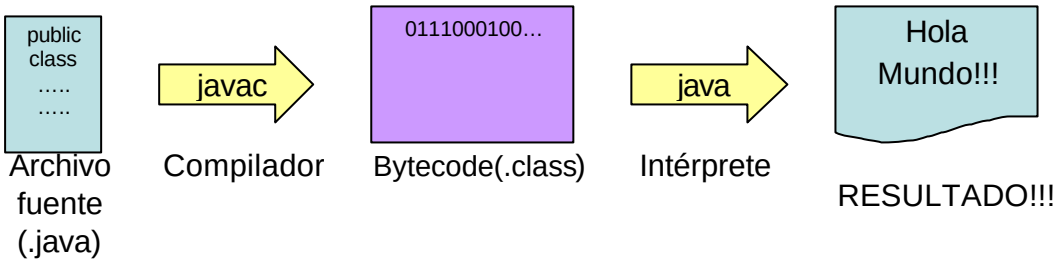


Figura 20 Flujo de una aplicación en la plataforma Java

- ✓ ***Tipos de datos***

Java no es un lenguaje orientado a objetos puro, ya que cuenta con ocho tipos de datos primitivos:

Enteros:
- **byte** **8 bits**
- **short** **16 bits**
- **int** **32 bits**
- **long** **64 bits**

Punto flotante:
- **float** **32 bits**
- **double** **64 bits**

Además:
- **char** **16 bits**
- **boolean** **verdadero/falso**

Los tipos primitivos y sus dimensiones están definidos en la especificación de la *JVM*

- ✓ ***Arreglos***

Son estructuras de almacenamiento estáticas, no pueden crecer en tiempo de ejecución, pero si pueden ser definidas e inicializadas en cualquier momento. Se pueden crear arreglos de cualquier tipo de datos, ya sea primitivo o definido por el usuario.

- ✓ ***Sentencias y control de flujo***

La sintaxis de las sentencias de control de flujo son como sigue:

if(<exprLógica>***)***(<*sentencia*>|**{**<*bloque*>**}**)[**else**(<*sentencia*>|**{**<*bloque*>**}**)]

switch(<exprEntera>**){**(**case(**<litEntera>**):**<*bloque*>[**break;**])*(**default:** <*bloque*>[**break;**])**}**

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

while(<exprLógica>**)** *(<sentencia>*|**{**<*bloque*>**})**

do *(<sentencia>*|**{**<*bloque*>**})* **while(**<exprLógica>**);**

for(<*sentencia*>*(,<sentencia>)**;* <exprLógica>*; <sentencia>(,<sentencia>)**)* *(<sentencia>*|**{**<*bloque*>**})**

- ✓ ***Objetos***

Objeto de software: es un conjunto de implementaciones en software que modelan un objeto del mundo real.

Los objetos se caracterizan por su estado, dado por un conjunto de atributos (variables), y comportamiento, representado por métodos (antes funciones). Muchos objetos comparten atributos, que pueden tener distintos valores, y acciones que se llevan a cabo de la misma manera. Podemos agruparlos y hacer una descripción general de los objetos del tipo que nos indique cuales son los atributos y acciones que tienen.

Creamos entonces una clase, la cual es una plantilla, que nos dice que es lo que define a los objetos que se crean a partir de ella. Una vez definida la clase podemos crear objetos a partir de esta, a este proceso se le conoce como crear instancias de una clase. Cada uno de estos objetos asigna valores a sus atributos y es totalmente independiente de los demás. Un objeto o una clase no hace nada por si solo, el secreto es hacer que objetos de distintas clases cooperen para realizar una acción.

- ✓ ***Variables***

Existen dos tipos de variables o miembros de una clase:

- Miembros estáticos: también llamados miembros de clase ,son los primeros que se colocan en memoria, se cargan una sola vez, todos los objetos comparten esa localidad de memoria.
- Miembros de instancia: existe cada objeto tiene su propia dirección de memoria para la variable.

- ✓ ***Herencia***

Muchas clases (no objetos) pueden compartir atributos y métodos, esto puede ser un indicador de relación de herencia. Podemos sacar los atributos que comparten y asignarlos a una clase mas general, de la que hereden. La herencia nos permite reutilizar software, ya que podemos escribir un método que es común a dos clases en la clase padre y así no lo escribimos dos veces.

- ✓ ***Polimorfismo***

Con la herencia aparece la propiedad de polimorfismo, la que nos dice que toda objeto de una clase hija puede ser considerado como un objeto de su clase padre.

La referencia nos restringe el acceso a los atributos, el tipo verdadero del objeto nos proporciona los métodos. Solo se puede hacer cast a una referencia, si el objeto al que apunta la referencia es del mismo tipo del destino.

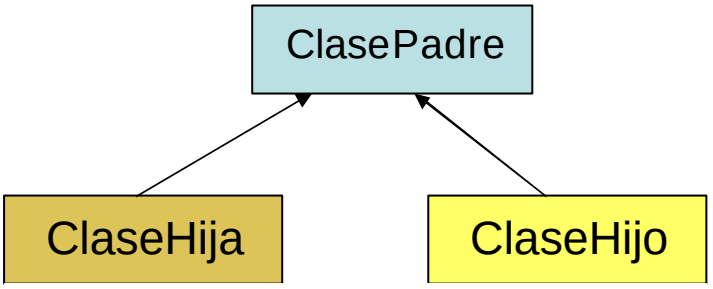


Figura 21 Polimorfismo

- ✓ ***Interfaces***

En Java “no existe” el concepto de herencia múltiple. Para simular herencia de este tipo se utilizan interfaces. Una interfaz es un conjunto de constantes y declaraciones de métodos, es un elemento de diseño que obliga a las clases a presentar un determinado comportamiento. Las interfaces nos aseguran el comportamiento de una clase.

Los objetos instancia de clases que implementan interfaces, también pueden ser considerados como objetos instancia de la interfaz. Las clases no pueden crecer ni ser mejoradas, ya que afectarían a las clases que las implementan

- ✓ ***Paquetes***

Para una mejor organización de las clases se utilizan paquetes. Un paquete es un conjunto de clases e interfaces que tienen una relación lógica o de diseño. Por convención los nombres de paquetes utilizan el dominio de la empresa de manera inversa:

org.apache.jakarta.*
com.sun.*
com.ibm.*

- ✓ ***Control de acceso a miembros***

Muchas veces, para su implementación, las clases necesitan de atributos y métodos con funcionalidad interna que nunca serán llamados por otras clases.

El control de acceso a miembros nos permite aplicar el principio de mínimo privilegio en el diseño de nuestras clases.

Especificador	clase	subclase	paquete	todos
private	X			
protected	X	X*	X	
package	X		X	
public	X	X	X	X

Figura 22 Control de acceso a miembros

- ✓ ***Excepciones y manejo de errores***

Java tiene un estricto control y manejo de las excepciones. Las excepciones representan casos *"no esperados"* en el flujo normal de nuestro programa.

Existen dos clases de errores:

- Error: son errores graves difíciles de manejar y de recuperarse.
- Excepciones: pueden ser manejadas de manera mas fácil, normalmente se puede recuperar el flujo

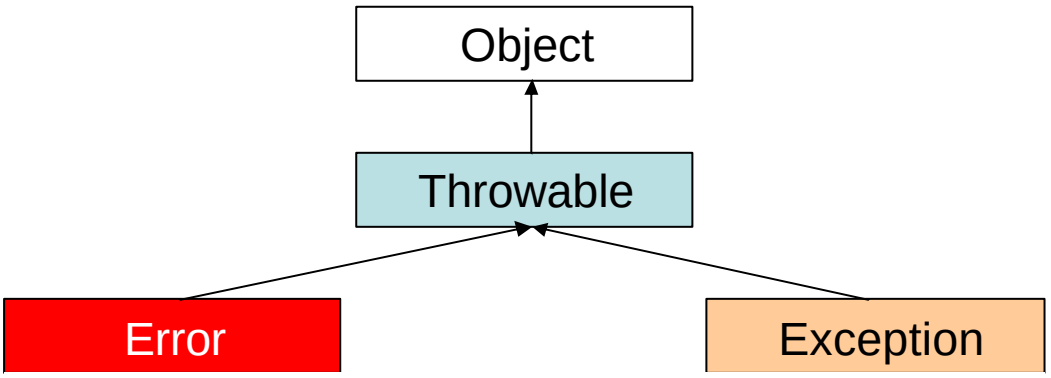


Figura 23 Excepciones

- Excepciones reportadas: son excepciones que sabemos que pueden ocurrir.
- Excepciones en tiempo de ejecución: son excepciones no reportadas debidas a errores propios de un ambiente de ejecución.

✓ ***Manejo de Excepciones***

- Reportar excepciones: cuando no queremos manejar la excepción en ese momento y dejarlo a otro nivel.

  public void metodo() ***throws Exception***

- Manejo de excepciones: bloques try,catch,finally

try
{bloquePeligroso}

catch(tipoExcepcion nombre){bloqueAEjecutar}
catch(otroTipoExcepcion nombre){bloqueAEjecutar}

finally{bloqueFinal}

✓ ***Documentación***

JavaDoc es una herramienta del *JDK* que nos ayuda a generar la documentación en un formato estándar con salida *HTML*.

Para poder generar la documentación debemos incluir en nuestro código comentarios de documentación. Son bloques de comentario que comienzan con /** y terminan con */. Estos bloques se colocan justo antes de la declaración de lo que se quiera documentar.

Etiquetas:

- @see nombreClase: agrega el campo *See Also* que contiene *hiperligas* a las clases referenciadas.
- @see nombreClase#metodo: agrega una *hiperliga* al método de otra clase.
- @version descripción : agrega el campo *Version.*
- @author autor: crea el campo *Author.*
- @param parámetro descripción: agrega la descripción de un parámetro a la lista.
- @return descripción: agrega una descripción para los datos regresados por un método.
- @exception nombreClase descripción: agrega una descripción a la lista de excepciones lanzadas por el método.
- @deprecated explicación: marca como obsoleto al método.

- @since versión: Agrega el campo *Since* para indicar cual es el *API* mínimo que se debe usar para esta clase.

✓ ***Archivos transportables***

Es conveniente que una vez creadas nuestras clases las empaquetemos para poder transportarlas fácilmente a los distintos ambientes de producción.

Los archivos .jar son archivos compresos (.zip) que contienen los recursos necesarios para nuestra aplicación (por ejemplo clases, archivos de configuración).

✓ ***Archivos de propiedades.***

La clase *java.util.Properties* nos da la facilidad de crear archivos de propiedades, muy usados para crear archivos de configuración.

Los archivos de propiedades son de la forma:

```
propiedad=valor
#comentario
!comentario
propiedadVariasLineas=palabra1\
                      palabra2
```

Los archivos de propiedades se cargan y manipulan en memoria, podemos agregar y eliminar propiedades, si embargo para que se vean reflejados los cambios debemos guardarlos.

✓ ***JavaBeans***

Son piezas de software que engloban una funcionalidad especifica, ya sea reglas de negocio, elementos gráficos, etc.

- *Beans* gráficos: son componentes que nos ayudan a construir *GUI*.

- *Enterprise Java Beans*: es un componente que encapsula lógica de negocio de la aplicación.

    - *Session Beans*: realizan una tarea especifica con el usuario, pueden o no guardar un estado que represente el comportamiento del cliente en la aplicación.
    - *Entity Bean*: Representa un objeto persistente de nuestra aplicación.
    - *Message Driven Bean*: *JMS AP*

- ✓ ***JDBC***

Es el *API* que nos provee las clases e interfases necesarias para tener interacción con bases de datos.

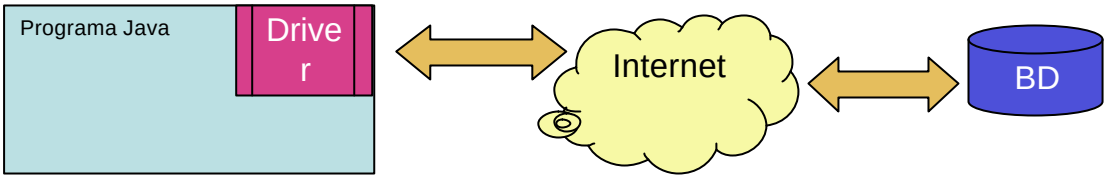


Figura 24 JDBC

Los programas Java con conexión a BD utilizan un *driver* para conectarse. Un *driver* es un conjunto de clases que implementan la especificación *JDBC* para una BD especifica.

Para interactuar con la BD es necesario:

1. Conseguir el *driver* para el DBMS requerido. El distribuidor del DBMS debe proveer el *driver* y la documentación para usarlo.
2. Registrar el *driver* en la JVM.
   - Cargándolo desde el código
   - Usando la opción jdbc.drivers del intérprete
2. Crear una conexión a la BD.
   - Parámetros en el código
   - Archivo de propiedades
3. Ejecutar sentencias.
   - Sentencias.
   - Sentencias precompiladas
4. Manejar resultados.
   - Iterar sobre el ResultSet
5. Cerrar todos los flujos

Existen cuatro tipos de *Drivers :*

- Puente *JDBC-ODBC*: convierte las llamadas *JDBC* en llamadas *ODBC* y las pasa a la maquina *ODBC*.

- *API* nativo - *Driver* parte en Java: Este tipo de *drivers* se comunican directamente con el *DBMS*, requiere de programas extras a las clases de Java (librerías, dll, etc.) Convierte las llamadas de *JDBC* en comandos específicos de la BD.

- Protocolo de red - *Driver* completo en Java: Utiliza protocolos de red para comunicar los programas con el BD, un programa de enlace en el servidor convierte el protocolo de red al protocolo especifico de la BD.

- Protocolo nativo – *Driver* completo en java: convierte las llamadas *JDBC* en el protocolo nativo del *DBMS*.

Contenido de un Driver

Implementación de:

- java.sql.Driver que es la clase encargada de crear conexiones y convertir las llamadas *JDBC* a llamadas nativas del *DBMS*.

- java.sql.Connection la clase que maneja la conexión a la BD.

- java.sql.Statement representa las sentencias que se ejecutaran en la BD.

Según el tipo del *driver* varia la instalación del mismo, pero en nuestro código el formato no cambia.

La puerta de entrada al *driver* es la clase que implementa java.sql.Driver, no necesitamos preocuparnos por el resto de las clases.

- ✓ ***Conversión de tipos Java-SQL***

BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

- ✓ ***Stored Procedures***

El JDBC API nos provee de una forma estándar para invocar procesos almacenados en nuestra BD. Sin embargo no todos los *DBMS* soportan *stored procedures*.

java.sql.CallableStatement es la interface que nos permite invocar procesos, los pasos para manejar *stored procedures*:

- Crear un *CallableStatement* con una sentencia parametrizada.
  conexion.prepareCall("{call sp_proceso ?}");
- Establecer los argumentos de entrada
  cs.setXXX(indice,valor);
- Registrar los argumentos de SALIDA
  cs.registerOutParameter(indice,tipo);
- Establecer y registrar los parámetros ENTRADA / SALIDA.
- Ejecutar la llamada según el tipo de argumentos del *SP*.
  - execute()
  - executeQuery()
  - executeUpdate()

- ✓ ***Servlets***

Los *Servlets* fueron la respuesta de la tecnología Java a la necesidad de contenido dinámico generado en el servidor. Son clases Java que son invocadas por el motor de contenido dinámico y responden a este mismo generando contenido HTML.

- Servidor Web
  - Apache
  - IIS
- Motor de contenido dinámico (Contenedor de *Servlets*)
  - Apache Tomcat (Servidor *Web* + contenedor *Servlets*)
- Java Virtual Machine.

El *Servlet API* es un conjunto de clases e interfases que nos ayudan a crear servlets

Todo *servlet* hereda de *javax.servlet.http.HttpServlet* y debe manejar los distintos tipos de peticiones.

- POST ←
- GET ←
- DELETE
- PUT

- OPTIONS
- TRACE

El *servlet* se comunica con el exterior a través de objetos *request* y *response*.

- *javax.servlet.http.HttpServletRequest* este tipo de objetos pasan parámetros al *servlet*.
- *javax.servlet.http.HttpServletResponse* representa el flujo de salida del *servlet*.

Los *servlets* son invocados y controlados por el contenedor, por lo que los flujos de entrada y salida estándar no son utilizados (están reservados para el contenedor).

System.in y System.out son sustituidos por *request* y *response* respectivamente

- ✓ ***Sesiones***

Muchas aplicaciones requieren que se mantenga información acerca del cliente, como selecciones, preferencias, datos etc. El protocolo HTTP no permite mantener estados, solo se basa en peticiones y respuestas.

Dos formas de guardar el estado de un cliente:

- *Cookies*: del lado del cliente
- *Session*: en el servidor.

El *Servlet API* proporciona una serie de clases que nos permite guardar y manejar el estado en una sesión. Una sesión es representada por objetos de la clase *javax.servlet.http.HttpSession* La sesión es proporcionada por el objeto *request* a través del método getSession().

- ✓ ***JSP***

Usamos *servlets* para recibir y procesar datos, sin embargo para presentar resultados en pantalla es un proceso muy complicado. Ante esta necesidad surgieron los *Java Server Pages* que nos permiten crear la salida HTML e incrustar código Java. Un JSP es un archivo muy parecido a HTML con la extensión .jsp, en el cual podemos incluir bloques de código java.

Para insertar bloques en lenguaje java tenemos las etiquetas:

- Directivas: no generan salida, simplemente ayudan al servidor a configurar el resultado del *jsp*. Se encierran entre las etiquetas <%@ y %>
- Declaraciones: el cuerpo del *jsp* que creamos con el editor se engloba en un solo método, para declarar otros métodos o bien algunas variables de instancia usamos las etiquetas <%! y %>

- Expresiones: generan una cadena en el cuerpo de la pagina de salida, sustituyen a la expresión *out.println(),* lo que se escriba en las etiquetas debe devolver una cadena, la expresión NO lleva punto y coma al final: <%= expresion %>

- *Scriptlets*: El código que se escriba entre sus etiquetas será pasado al servicio principal del *JSP* que atiende todas las peticiones <% y %>

Objetos implícitos.

- *request :* representa el objeto *HttpServletRequest* que contiene la petición que llego al *JSP*.
- *response* : representa el objeto *HttpServletResponse* que comunica el *JSP* con su flujo de salida.
- *application* : representa el *ServletContext* de este *JSP*, es un reflejo de la aplicación *web* en nuestro servidor.
- *out* : el *PrintWriter* de salida del *JSP*
- session : un objeto *HttpSession* valido para este *JSP*

✓ ***Archivos externos***

Los JSP nos permiten incluir de manera estática y dinámica archivos de texto externos, de esta manera podemos tener en distintos archivos funcionalidades independiente y ponerlas todas juntas en un jsp:

- Directiva *include:* incluye el archivo en el momento de la transformación de JSP-Servlet.
  <%@ include file="rutaRelativa" %>
- Acción *jsp:include* incluye el archivo en el momento de la ejecución del Servlet.
  <jsp:include page="rutaRelativa"/>

De ambas formas se pueden incluir html estático o bien jsp, en este ultimo caso primero es incluido el código jsp y después interpretado.

✓ ***Uso de Beans en los JSP***

Los JSP nos proporcionan un conjunto de etiquetas predefinidas que nos ayudan a manejar los datos que llegan a través de una petición

<jsp:useBean id="usuario" class="paquete.Clase"> : crea una instancia de la clase y la pone disponible para el resto del codigo.

<jsp:setProperty name="usuario" property="propiedad" value="valor"/> :
establece un valor a una propiedad, convirtiendo "valor" al tipo de dato que reciba getPropiedad()

2.-Estándares de codificación

Es importante conocer y aplicar convenciones al generar código porque:

- El 80% de la vida de un programa es mantenimiento.
- Difícilmente es el autor original del código quien hace este mantenimiento.
- Las convenciones mejoran la lectura ayudando a comprender mas fácilmente el código.

La especificación del lenguaje Java propone estándares para la codificación de software, tales como:

- Todos los archivos Java contienen una sola clase, o interfase, publica. Si se tienen clases privadas asociadas se deben poner en el mismo archivo, la clase publica es la primera que aparece.

- Un archivo Java tiene la siguiente estructura.
  - ✓ Comentarios de inicio como copyright, versión, fecha, etc.
  - ✓ Sentencia *package*.
  - ✓ Sentencias *import*. Si no se necesitan todas las clases de un paquete se debe evitar el uso de *, listando una a una las clases importadas.
  - ✓ Declaración de la clase.
- La declaración de la clase lleva la siguiente estructura.
  - ✓ Comentario de documentación.
  - ✓ Sentencia *class* o *interface*.
  - ✓ Comentario de implementación (opcional).
  - ✓ Variables de clase (*static*) ordenadas por su modificador de acceso: *public, protected, package y private.*
  - ✓ Variables de instancia ordenadas de la misma manera que las de clase.
  - ✓ Constructores ordenados ascendentemente por el numero de argumentos.
  - ✓ Métodos ordenados y agrupados por funcionalidad.
- Las líneas no deben ser de mas de 80 caracteres.
- Se hace una declaración por línea. Nunca se declaran dos tipos de datos en la misma línea.
- Hacer todas las declaraciones al inicio de un bloque.

Convenciones de nomenclatura.

- *Paquetes :* inician con un nombre de dominio de primer nivel (mx, org, com, edu) y es todo con minúsculas.
- *Clases e interfaces :* son sustantivos y comienzan con mayúscula, se concatenan las palabras de su nombre con la letra inicial mayúscula.
- *Métodos :* son verbos comienzan en minúscula y se concatenan las palabras con la primera letra mayúscula.
- *Variables :* deben ser descriptivas y respetan la misma convención que los métodos, los nombre de una sola letra se usan solo para variables de desecho, como contadores en los ciclos.
- *Constantes (final):* todas sus letras son mayúsculas y las palabras se separan por un guión bajo '_'.

Aplicaciones Web

Se tiene una jerarquía de carpetas y archivos con una organización estándar que los servidores web esperan al levantar una aplicación (especificación Servlet 2.2):

- **Raíz**
  - ✓ *.html,*.jsp,*.gif…
  - ✓ *jsp*
  - ✓ *js*
  - ✓ *images*
  - ✓ *css*
  - ✓ *flash*
  - ✓ **WEB-INF**
    - ✓ **web.xml**
    - ✓ **classes**
    - ✓ **lib**

3.-Metodología de desarrollo

Se creará un API que nos ayude a trabajar con objetos de la BD, que realice de manera sencilla las tareas mas comunes:

- Inserción
- Búsqueda
- Modificaciones
- Eliminar

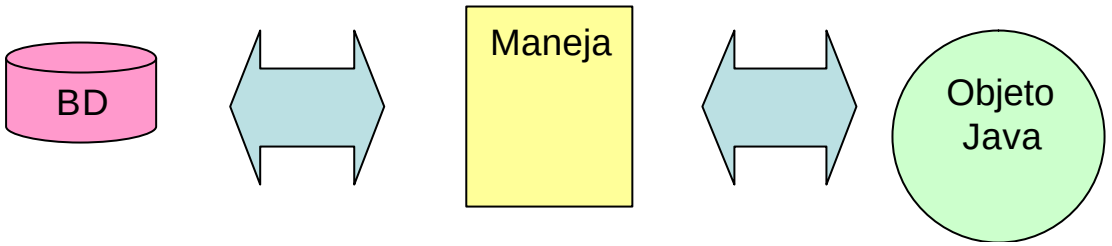


Figura 25 Arquitectura

Arquitectura

Para cumplir con esta metodología, para cada tabla de la base se creará una clase con la misma estructura que ésta, haciendo una correspondencia entre tipos de datos de sql con java.

Por otro lado se crean una serie de clases que permitan pasar los datos entre las tablas y los objetos. A estas clases se les llamará *API de persistencia*.

API de Persistencia

Este API se encargará de la relación entre la Base de Datos y la aplicación. La idea básica de la estructura es tener clases espejo de las tablas de la BD, esto es, clases java que contengan atributos que puedan contener los campos de una tabla de la BD.

Las clases espejo tienen la estructura de un Bean, lo que permite manejarlos como tal , por ejemplo para las directivas de la especificación JSP

<jsp:useBean>, <jsp:setProperty>.

Esta estructura facilita el atrapar los datos por medio de las directivas jsp anteriormente mencionadas, sin embargo para introducirlos a la BD de datos queda el proceso de extraer los datos de la clase e insertarlos. Para esto se utilizan las clases *ManejaXXX.*

Las clases *ManejaXXX* sirven para manipular los datos e interactuar con la BD, idealmente cada clase espejo debe tener su clase ManejaXXX, así la clase Reunion tiene su respectiva clase ManejaReunion.

Todas las clases ManejaXXX heredan de la clase abstracta *ManejaBean* la cual se mencionará más adelante y sobrescriben sus métodos.

El proceso de interacción entre las clases espejo y la BD de datos requiere que se proporcionen los siguientes datos en la clases ManejaXXX.

- ✓ Nombre de la tabla a la que corresponde esta clase espejo.
- ✓ Tabla de correspondencia entre los nombres de los campos de la BD y el nombre de los atributos de la clase espejo

Además se deben sobrescribir los métodos de la clase ManejaBean por métodos mas específicos de la clase ManejaXXX que le corresponde. Esta sobre escritura consiste en mandar llamar a los métodos de ManejaBean con parámetros específicos de la clase ManejaXXX.

El modo de uso de este esquema de programación se puede resumir en tres pasos:

- ✓ Crear la clase espejo de la tabla de la BD.
- ✓ Crear la clase ManejaXXX de la clase espejo.
- ✓ Comenzar a utilizar ambas clases para realizar las acciones correspondientes.

Las clases base que permiten aplicar este esquema de programación son:

ManejaBean.java
PuenteBean_BD.java
ConexionBD.java

El propósito de estas clases es que el usuario pueda interactuar de manera transparente con la BD, aplicando las operaciones básicas de insertar, eliminar y modificar registros.

ConexionDB.java

Esta clase contiene métodos convenientes para realizar operaciones sobre una BD. Se encarga de obtener conexiones, realizar operaciones y liberar las conexiones.

Para utilizar esta clase se crea un nuevo objeto ConexionDB cuando se necesite realizar una operación, se realizan las operaciones con la base de datos y luego se cierra la conexión.

PuenteBean_BD.java

Como su nombre lo indica es el puente que une una clase espejo con la BD, sus métodos permiten llenar un objeto del tipo de una clase espejo con los resultados de una búsqueda (en esta implementación se toman datos de un objeto java.sql.ResultSet).

También permite formar las cadenas de búsqueda y actualización en lenguaje SQL, lo que ahorra la tarea de formarlas de forma manual.

ManejaBean.java

Se trata de una clase que se considera abstracta con el fin de que en el momento de la implementación del esquema todas las clases ManejaXXX hereden de ella e invoquen sus métodos con parámetros más específicos.

Las tres clases mencionadas anteriormente fueron diseñadas para ser lo mas generales posibles.

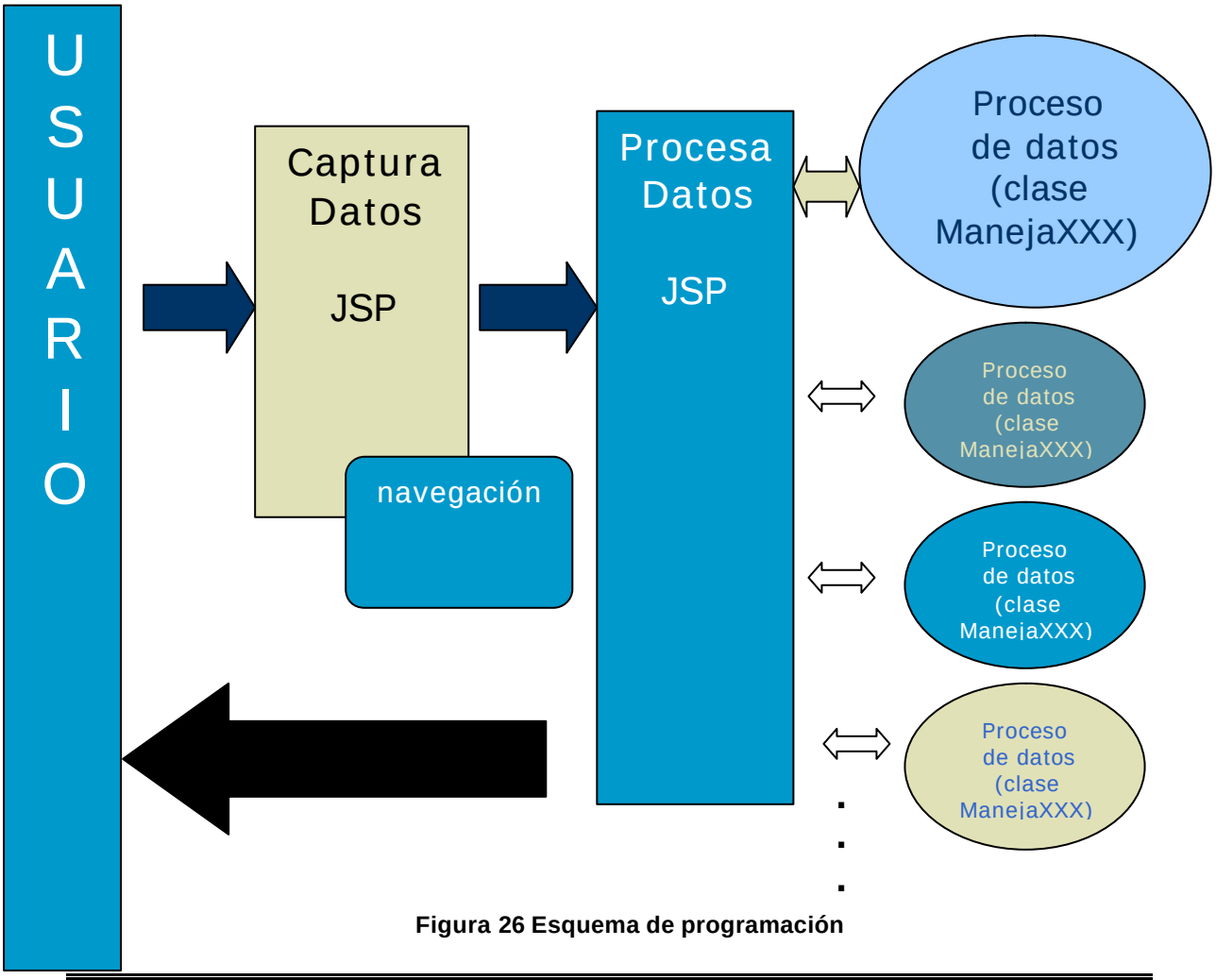


Figura 26 Esquema de programación

Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

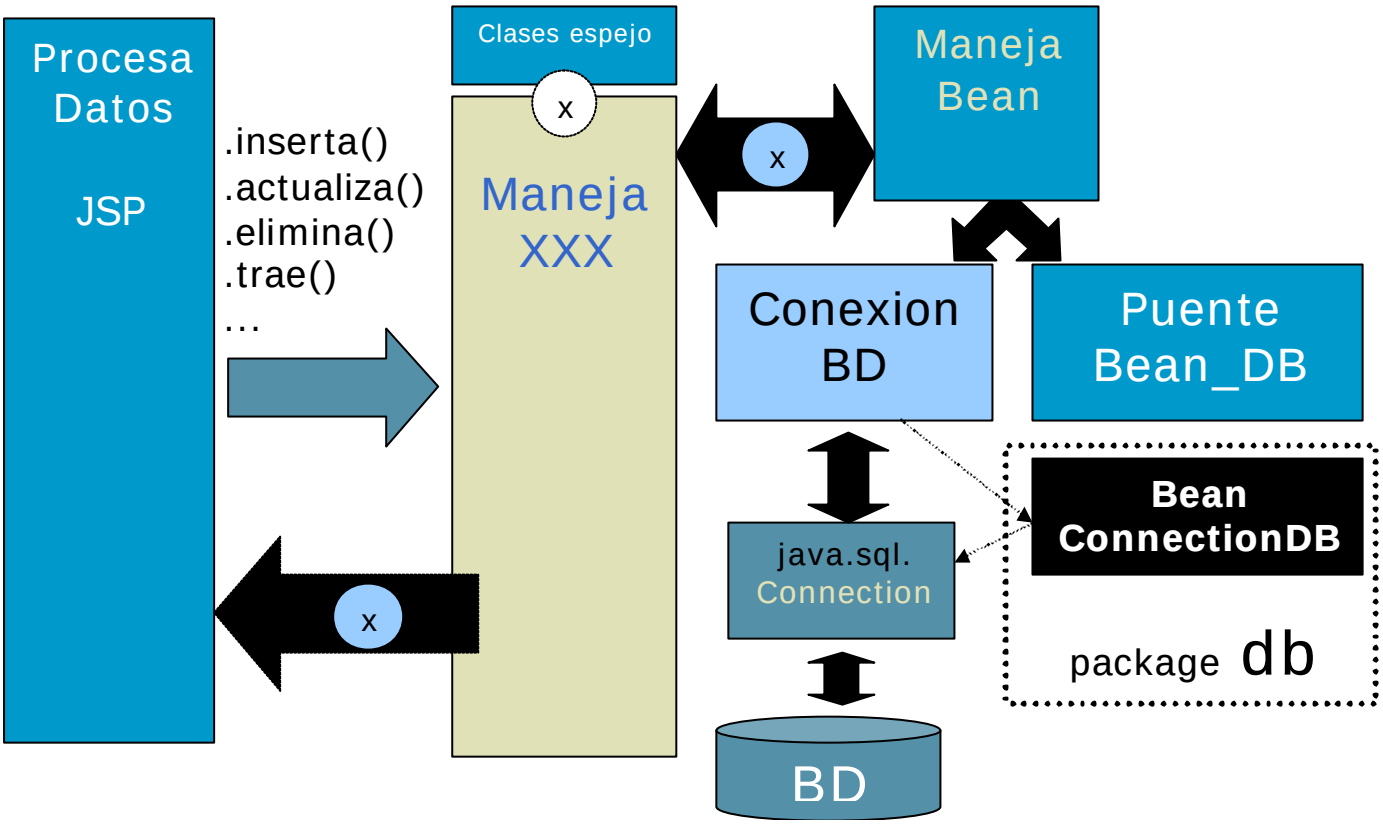


Figura 27 Procesamiento de datos

4.-Codificación

El codificación se facilita al tener la arquitectura definida como en este caso, por lo que la programación de todo el sistema se hará de una manera similar, para ejemplificar esto, a continuación se muestran fracciones de código para el ejemplo de un catálogo y sus funcionalidades correspondientes.

Tomaremos como ejemplo el catálogo de países. Los archivos involucrados son:

Pais.java (bean)
ManejaPais.java (maneja)
CatalogoPais.jsp (catalogo para vista)
CapturaPais.jsp (formulario de captura)
GuardaPais.jsp (guardar los cambios)

Pais.java

```
package beans;
public class Pais
{
   private java.lang.String pais;
   private java.lang.String descripcion;
   private java.lang.String status;

   public java.lang.String getPais()
   {
      return pais;
   }

   public void setPais(java.lang.String newPais)
   {
      pais = newPais;
   }

   public java.lang.String getDescripcion()
   {
      return descripcion;
   }

   public void setDescripcion(java.lang.String newDescripcion)
   {
      descripcion = newDescripcion;
   }
   public java.lang.String getStatus()
   {
      return status;
   }

   public void setStatus(java.lang.String newStatus)
   {
      status = newStatus;
   }
}
```

ManejaPais.java

```
package maneja;
import BD.ConexionBD;
import BD.ParametrosConexion;
import java.util.Hashtable;
import persistencia.utils.BD.ManejaBean;
import beans.Pais;
public class ManejaPais extends ManejaBean
{
    private static String tabla="Pais";
    private static Hashtable tablaMapeo=new Hashtable();
    static{
        tablaMapeo.put("pais","pais");
        tablaMapeo.put("descripcion","descripcion");
        tablaMapeo.put("status","status");
    }

    public static Pais trae(ParametrosConexion param, String condiciones)
    throws Exception
    {
        Pais pais= new Pais();
        return  (Pais)trae(pais,tablaMapeo,tabla,condiciones,ConexionBD.
        getConnection(param));
    }

    public static int inserta(ParametrosConexion param, Pais pais)throws
    Exception
    {
        Hashtable tmp= (Hashtable)tablaMapeo.clone();
        return inserta(pais,tabla, tmp, ConexionBD.getConnection(param));
    }

    public static Pais[] busca(ParametrosConexion param, String
    condiciones) throws Exception
    {
        return (Pais[]) busca( Pais.class , tablaMapeo, tabla,
        condiciones,ConexionBD.getConnection(param));
    }

    public static int actualiza(ParametrosConexion param, Pais pais,
    String condiciones)throws Exception
    {
        Hashtable tmp= (Hashtable)tablaMapeo.clone();
        tmp.remove("pais");
        return actualiza(pais, tmp, tabla, condiciones,ConexionBD.
        GetConnection(param));
    }

    public static int elimina(ParametrosConexion param, String
    condiciones) throws Exception
    {
        return elimina( tabla, condiciones,ConexionBD.getConnection
        (param));
    }
}
```

CatalogoPais.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java"
import="beans.Persona,beans.Pais, maneja.ManejaPais, beans.Status,
maneja.ManejaStatus"
errorPage="error.jsp"%>
<%String clave="CTPSCN";
Persona persona= (Persona) session.getAttribute("atributosPersona");
String pais = (String) session.getAttribute("pais");%>
<html>
<head>
<link href="../css/Acuerdate.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../js/Checar.js"></script>
<title>CatalogoPais</title>
<script language="javascript" type="text/javascript">
function modificar(pais)
{
      document.location.href="./CapturaPais.jsp?pais="+pais
}
function eliminar()
{
var cadena="";
var pa=arregloRegistros.length;
      for(i=this.arregloRegistros.length-1;i>=0;i--)
            cadena+="\n× "+arregloRegistros[i];

      if (confirm("¿Desea eliminar los siguientes registros " + cadena +
      " ?"))
      {
            document.location.href="./GuardaPais.jsp?elimina=Si&pa="+pa+"
            &aR="+arregloRegistros
      }
}
function msg(elemento)
{
      alert("No   se   pudo   eliminar   "+elemento+"   ya   que   existen
      dependencias");
}
</script>
<body onload="
<%if(pais!=null)
{%>
      msg('<%=pais%>')<%session.removeAttribute("pais");
}%>">
<%if (persona==null)
{
      out.println("No se ha logeado al sistema");
}
else
{%>
      <%@ include file="Claves.jsp" %>
      <%if (temp)
      {
            Pais[] catalogo=ManejaPais.busca(param," ORDER BY
            Descripcion");
```

```
            Status status;%>
.
.
.
<table width="100%">
  <tr class="encabezado">
      <td width="46%">Pa&iacute;s</td>
      <td width="27%" >Abreviatura</td>
      <td width="24%"><div align="center">Status</div></td>
      <td width="3%"> </td>
  </tr>
  <% for (int i=0; i< catalogo.length ;i++)
{%>
  <tr class="normal" style='cursor:pointer'
  onMouseOver=this.className='seleccion2'
  onMouseOut=this.className='normal'
  onDblClick=modificar("<%= catalogo[i].getPais().trim() %>")>
      <td><%= catalogo[i].getDescripcion()%></td>
      <td><%= catalogo[i].getPais()%></td>
      <td><div align="center">
        <% status=ManejaStatus.trae(param," WHERE Status='"+
        catalogo[i].getStatus()+"'");%>
            <%= status.getDescripcion() %></div></td>
      <td><div align="center">
        <input  id="<%=catalogo[i].getDescripcion().trim()%>" name=
        "checkbox" type="checkbox"  value="<%=catalogo[i].getPais()%>">
      </div></td>
  </tr>
  <%}%>
</table>
.
.
.
<%}
      else
      {%>
            <span class="Estilo15">
            No tiene acceso a esta funcionalidad</span></p>
      <%}
}%>
</body>
</html>
```

CapturaPais.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java"
import="beans.Persona,   beans.Pais,   maneja.ManejaPais,   beans.Status,
maneja.ManejaStatus"
%>
.
.
.
      Pais modificar=ManejaPais.trae(param, " WHERE Pais='"+pais+"'");
      Status[] statuss=ManejaStatus.busca(param," WHERE
      Categoria='Catalogo' ORDER BY Descripcion");
            if (pais==null)
            {%>
                  Nuevo Pa&Iacute;s
            <%}
            else
            {%>
                  Modificar el Pa&Iacute;s
                  <%= modificar.getDescripcion() %>
            <%}%>
.
.
.
<form action="GuardaPais.jsp" method="post" name="datos" id="datos">
.
.
.
<strong>Abreviatura</strong>
<%if (pais == null)
{%>
      <input name="pais" type="text" id="pais"   size="10" maxlength="3">
<%}
else
{%>
      <input type="hidden" name="pais" value="<%=modificar.getPais()%>" >
      <%=modificar.getPais()%>
<%}%>
Nombre <input name="descripcion" type="text" id="descripcion"
<% if (pais!= null)
{%>
      value="<%= modificar.getDescripcion() %>"
<%}%>
size="60" maxlength="50">
.
.
.
<input   name="Submit"   type="submit"   class="Estilo2"   value="Guardar"
onClick="validacampos();return false;">
.
.
.
```

GuardaPais.jsp

```
<%@page import="maneja.ManejaPais, beans.Persona, beans.Pais"
errorPage="error.jsp"%>
<jsp:useBean id="pais" class="beans.Pais">
<jsp:setProperty name="pais" property="*"/>
</jsp:useBean>
.
.
.
String pa=request.getParameter("pa");
String pagina="CatalogoPais.jsp";
String elimina=request.getParameter("elimina");
.
.
.
<%if (pa!=null)
{
      if (elimina!=null)
      {
            String aR=request.getParameter("aR");
            .
            .
            .
            try
            {
                  ManejaPais.elimina(param," WHERE Descripcion='"
                  +aR.trim()+"'");
            }
            catch(Exception e)
            {
                  session.setAttribute("pais", aR.trim());}
            }
            .
            .
            .
      }
      else
      {
            ManejaPais.actualiza(param,pais," WHERE Pais='"
            +pais.getPais()+"'");
      }
}
else
{
      ManejaPais.inserta(param,pais);
}
response.sendRedirect(pagina);
.
.
.
```

CAPITULO 6 LIBERACIÓN

1.-Requisitos de instalación

Para el correcto funcionamiento del sistema, se debe de contar con los siguientes requisitos técnicos en las computadoras donde se instalará y en las que se utilizará el sistema.

Requerimientos de software para el servidor contenedor del sistema:

- Sistema Operativo : Windows 2000, Windows XP SP2, Windows 2000 Server SP4, Windows 2003, Linux, Solaris
- Apache Tomcat v 4.1.31 (Instalación por nuestra cuenta)
- j2sdk v 1.4.1_01 de Sun Microsystems (Instalación por nuestra cuenta)

Requerimientos de software para el servidor contenedor de base de datos:

- Microsoft SQLServer 2000 Español SP3
- PostgreSQL 8.0

Requerimientos de hardware para el servidor contenedor del sistema:

- Tamaño en Disco Disponible: aprox. 75 MB

De los cuales:

Apache Tomcat	50 MB
J2sdk	10MB
Aplicación WEB	10MB
Base de Datos	5MB (Inicial)

- Memoria RAM 512 MB o mayor (recomendable)

Requerimientos para equipos clientes:

- Navegador Microsoft Internet Explorer v.6 o superior, Firefox.
- Opciones de bloqueo de menús emergentes desactivada.
- Permitir ejecutar lenguaje Javascript.
- Acrobat Reader 6.0 o superior
- Macromedia Flash Player v 7 o superior

2.-Instalación

Para instalar el sistema correctamente se deben llevar a cabo los pasos que se describen a continuación:

1. Ejecutar j2sdk_1_4_1_01-windows-i586.exe.

2. Crear: variable de Sistema: CLASSPATH=. (punto)
   Para hacer esto: Seleccionar Propiedades con el botón derecho sobre Mi PC, entrar a opciones avanzadas y crear nueva variable de sistema.

3. Ejecutar: jakarta-tomat-4.1.31.exe
   Al ejecutarlo, aparecerá una ventana reconociendo lo que se instaló en el punto anterior, se debe seleccionar aceptar.
   Instalar Apache Tomcat con usuario: admin., password: admin (opcional).
   Y como NTService. (para hacer esto: seleccionar la casilla que dice NTService; esto quiere decir que cuando se levante el sistema operativo también se levantará el servicio de tomcat y ya no se tendrá que hacer manualmente)

4. Crear: Variable de Sistema: JAVA_HOME=C:\j2sdk1.4.1_01 (o la ruta donde se instaló el punto 1)

5. Copiar los archivos:
   persistencia.jar,
   itextl.3.1.jar
   jtds-0.9.jar
   commons-fileupload-1.0
   en:
   C:\Archivos de programa\Apache Group\Tomcat 4.1\common\lib

6. Detener el Servicio de tomcat.
   Para hacer esto: Entrar a Panel de Control → Herramientas Administrativas →Servicios y ahí buscar en la lista el Servicio de Apache Tomcat 4.1, seleccionarlo y darle DETENER.

7. Copiar Carpeta del sistema en:
   C:\Archivos de programa\Apache Group\Tomcat 4.1\webapps

8. Iniciar el Servicio de tomcat.
   Para hacer esto: Entrar a Panel de Control → Herramientas Administrativas →Servicios y ahí buscar en la lista el Servicio de Apache Tomcat 4.1, seleccionarlo y darle INICIAR.

9. Correr el script: BaseDatos.sql.

Para hacer esto en SQLServer: entrar al Analizador de Consultas de SQLServer, y conectado a la base de datos “master”, abrir el archivo: BaseDatos.sql, y seleccionar F5 o en la barra de menú seleccionar Ejecutar Consulta.

10. En Caso de Tener instalado el ServicePack 2 y para que funcione correctamente el SQLServer:
    1. Respaldar las BDs en el Administrador Corporativo.
    2. Instalar el archivo:
       Si su SQLServer es en Español: **esn_eql2ksp3.exe**
       Si su SQLServer es en Ingles: **sl2ksp3.exe**

    Para hacer esto: desactivar los antispywhere. Ejecutar el archivo (español o inglés), aparecerá un mensaje de “entrega satisfactoria”, entrar a la carpeta que tiene el mismo nombre del archivo que se ejecuto en el paso anterior, buscar el **SETUP.bat** y ejecutarlo.

    Seguir la instalación y al finalizar reiniciar el equipo

11. Probar:
    http://localhost:8080/Tesis

Donde *localhost* es el nombre del servidor donde copio el Sistema y *Tesis* es el nombre de la carpeta que contiene el sistema.

3.-Configuración

Para configurar el sistema se debe de entrar a:

C:\Archivos de programa\Apache Group\Tomcat 4.1\webapps\ Tesis\ WEB-INF\classes\resources

Y en el archivo ConexionBD.cnf se configura el manejador de base de datos que se utilizará, el driver para hacer la conexión, el servidor (en este caso *"localhost"*) y la base de datos a la que se conectará (en este caso *"Tesis").*

Para SQLServer:

driver=net.sourceforge.jtds.jdbc.Driver
url=jdbc:jtds:sqlserver://localhost:1433/Tesis

Para PostgreSQL:

driver=org.postgresql.Driver
url=jdbc:postgresql://localhost:5432/Tesis

Para configurar el número de licencia en el mismo directorio se edita el archivo: lcn.prp, con la siguiente línea

lcn=00001

Donde 00001 será el número de la primer licencia, el 00002 para la segunda, etc.

Para configurar el correo electrónico en el mismo directorio se entra al archivo Mail.cnf y aquí se edita el servidor de smtp, el correo desde cual se envía y el dominio.

Por ejemplo:
host=smtp.gestiondereuniones.com
from=sistema@gestiondereuniones.com
from2=@gestiondereuniones.com

4.-Control de versiones

Es importante contar con un control de versiones ya que esto nos ayudará a saber el avance del desarrollo y mantenimiento del sistema.

Fases del ciclo de vida de una versión

Cada transición de la liberación de un producto pasa a través de un ciclo de vida. El ciclo de vida consiste en cada una de las siguientes fases.

- Beta.
- First Customer Shipment (FCS).
- General Availability (GA).
- End of Life (EOL).
- Obsolete.

Durante cada fase la disponibilidad comercial y el nivel de soporte para una versión varía.

Beta

Una versión beta del producto esta disponible a los clientes con fines no comerciales, con la finalidad de mostrar nuevas características y funcionalidades a los mismos. Una versión beta no esta orientada para uso en producción.

First Customer Shipment (FCS)

Una versión del producto esta disponible comercialmente como *first customer shipment (FCS).* Esta versión esta disponible para nuevos clientes y clientes existentes que requieren las características y funcionalidades de la versión. En este caso no se anima a los clientes existentes a actualizar a una nueva versión hasta que esta llega a la etapa de GA. Una versión FCS es soportada totalmente en un ambiente de producción pero puede estar disponible solamente en plataformas seleccionadas.

General Availability (GA)

La versión del producto estará en la fase de *general availability (GA)* para indicar que éste ha pasado rigurosas pruebas e instalaciones exitosas. Una vez hecha a la versión GA comenzará su fase EOL y se animará a los clientes a actualizar su versión. Una versión GA es totalmente soportada en ambientes de producción y deberá estar disponible en todas las plataformas.

End of Life (EOL)

Una versión del producto se encontrará en la etapa de *end of life (EOL)* para indicar que ha comenzado su transición para llegar a ser una versión obsoleta. Los

clientes con contratos de mantenimiento serán motivados para actualizar a una versión durante esta fase de EOL hasta que la versión sea obsoleta. Una versión en la fase de EOL es aún comercialmente disponible y soportada para un ambiente de producción.

Version Obsolescence

Una versión del producto se hace obsoleta para indicar que esta ya no esta disponible comercialmente. Se requerirá a los clientes inicialmente actualizar a una nueva versión si requieren soporte bajo un contrato de mantenimiento.

Especificaciones del formato de numeración para versiones.

Cada versión del producto deberá contener un identificador asociado. Las reglas generales para establecer dicho indicador son las siguientes:

- Se debe utilizar siempre un separador para las diferentes partes del identificador de versión. Por ejemplo un identificador de versión 1.3.1beta no es aceptable, lo es de la forma 1.3.1-beta.
- El formato deberá ser numérico y en letras minúsculas con un "." dentro de los campos y un "-" entre campos.
- Un separador "." indica una versión superior.
- Un separador "_" indica una actualización de versión.
- Un separador "-" es utilizado para indicar una versión que no sea GA ni FSC.
- Una identificador de versión GA o FCS no deberá contener "-".
- No se deberán utilizar los símbolos "*" ni "+".
- Todas las versiones están ordenadas basándose en el estándar de la notación punto. Por ejemplo 1.3.0 < 1.3.0_01 < 1.3.1 < 1.3.1_01.

Tipo de versión	Descripción	Formato de numeración de versión	Ejemplo
Característica	Contiene una nueva funcionalidad.	n.n.0<-*identificador*> • El último digito siempre es cero. • El –*identificador* siempre es requerido para cualquier versión que no sea FCS o GA. • Una versión FCS o GA nunca tienen un –*identificador.*	Acuérdate versión “1.3.0” Acuérdate Standar Edition 1.3.0-b24
Mantenimiento	Ingeniería enfocada a la corrección de bugs.	n.n.n <-*identificador*> • El último digito nunca es cero. • El –identificador siempre es requerido para cualquier versión que no sea FCS o GA. • Una versión FCS o GA nunca tienen un –identificador.	Acuérdate versión “1.3.1-beta” Acuérdate versión “1.3.1-beta-b09” Acuérdate versión “1.3.1_05-ea-b01”
Actualización	Enfocada a la corrección de bugs encontrados por los clientes.	n.n.n_nn<-*identificador*> • Los primero tres dígitos especifican la característica o versión que se esta actualizando. • Los dos dígitos que siguen al guión bajo indican el número de actualización. • El –*identificador* es requerido para cualquier versión que no sea FCS o GA. • Una versión FCS o GA nunca tienen un –*identificador.*	Acuérdate versión “1.3.1_05-ea” Acuérdate versión “1.3.1_05” Acuérdate versión “1.3.0_03_ea”

Figura 28 Numeración de versiones

CONCLUSIONES

Actualmente gran parte de las organizaciones necesitan herramientas que les asistan en las tareas más frecuentes y que les optimicen tiempo, dinero y recursos humanos. Este caso en específico se enfoca en la gestión de reuniones, ya que independientemente del giro de la organización, se llevan a cabo reuniones entre su personal para llegar a acuerdos y tomar decisiones.

Es importante que se tenga una metodología definida antes de tratar de usar una herramienta de cualquier tipo, por lo que primeramente definió una metodología de gestión de reuniones para poder posteriormente desarrollar el sistema que sirve de herramienta para seguir dicha metodología.

El siguiente paso fue identificar las necesidades de los usuarios finales, para esto se realizaron reuniones con los usuarios para detectarlas y al participar en estas reuniones se pudieron ver casos prácticos en los que el sistema será utilizado.

Una vez teniendo en mente todas estas necesidades se procedió a hacer el análisis de estos requerimientos para poder hacer una propuesta de cómo se desarrollaría el sistema y como se cubrirían dichas necesidades con las funcionalidades propuestas.

En estas etapas es importante que los usuarios finales hagan todas sus observaciones y expresen todas sus dudas al igual que los diseñadores ya que del buen entendimiento entre estos dependerá que el sistema cumpla con los objetivos deseados.

Cuando se terminó el análisis a detalle del sistema, se hizo el diseño del mismo así como de su base de datos y se determinó que tecnologías se utilizarían para el desarrollo y los manejadores de base de datos para implementar el sistema.

Para el desarrollo del sistema se diseñó una arquitectura y se definió un esquema de programación para optimizar esta tarea. Pero como era de esperarse, difícilmente el ciclo de desarrollo de software se sigue al pie de la letra, por eso en muchos casos al igual que este aunque se está en la etapa de desarrollo se tiene que volver a hacer análisis y rediseñar algunas cosas que no se habían contemplado inicialmente.

Una vez terminado el sistema, se hicieron pruebas y se corrigieron los detalles de programación pertinentes para poder hacer su liberación y posteriormente se capacitó a los usuarios para que utilizaran correctamente el sistema.

Como en todos los sistemas de software no fue fácil hacer que los usuarios se acostumbraran al uso del sistema pero una vez logrado esto se obtuvieron beneficios importantes ya que con el sistema como herramienta en la gestión de reuniones:

Todas las personas involucradas en una reunión tienen conocimiento de los temas que se tratarán en dicha reunión con lo que se evita que las personas que asistan retrasen el desarrollo de la reunión por desconocimiento de estos.

Se optimiza el tiempo que dura una reunión, el registro de acuerdos y actividades y se facilita la obtención de documentos importantes como la minuta.

Se facilita el seguimiento a proyectos y reuniones relacionados con estos ya que se tiene un orden para no perder tiempo en búsquedas innecesarias.

Los usuarios pueden organizar sus actividades diarias considerando las tareas que tienen pendientes y con esto se evitan retrasos en el cumplimiento de dichas actividades.

Se obtiene movilidad e independencia ya que al ser un sistema WEB hace que los usuarios puedan usar el sistema desde cualquier lugar del mundo solo con una conexión a Internet.

El seguimiento de las actividades que el usuario encarga a otros es mucho más fácil ya que mediante el sistema se comunican los avances de cada uno así como también se comparten archivos y documentos.

Además de todos estos beneficios el sistema se adapta fácilmente a las necesidades de diferentes organizaciones y en pocas palabras se puede decir que este sistema mejora notablemente el desempeño de todos sus usuarios.

Con todo esto se puede concluir que el sistema desarrollado cumple con todos los objetivos iniciales e incluso los mejora.

ANEXOS

1.-Glosario

ACTIVE X	Conjunto de tecnologías desarrolladas por Microsoft que proporciona objetos basados en el Modelo de Objetos Componente y que busca establecer un estándar especialmente para integrarse en Internet
ACTOR	Entidad externa a un sistema de software que interactúa con él para obtener un resultado importante. Un actor puede ser una persona, otro sistema, una parte del hardware o el paso del tiempo.
ALGORITMO	Procedimiento matemático o lógico para resolver un problema.
API	Abreviatura de Application Progra Interface. Conjunto de herramientas, rutinas y protocolos utilizados para la construcción de aplicaciones. Una API facilita el desarrollo de aplicaciones ya que provee todos los bloques de construcción
BASE DE DATOS	Serie de datos afines acerca de un tema organizado de una forma práctica tal, que proporciona una base o fundamento para procedimientos, como la recuperación de información, la elaboración de conclusiones y la toma de desiciones.
BROWSER	Explorador o navegador. Programa que permite al usuario navegar por la WWW. Un navegador sirve como cliente de servidores Web.
CASO DE USO	Es el conjunto de acciones que un actor realiza en el sistema para obtener un resultado importante.
CLIENTE	Cualquier elemento de un sistema de información que requiere un servicio mediante el envío de solicitudes al servidor. Cuando dos programas se comunican por una red, el cliente es el que inicia la comunicación, mientras que el programa que espera ser contactado es el servidor. Cualquier programa puede actuar como servidor para un servicio y como cliente para otro.
COMPILADOR	Programa que lee instrucciones escritas en un lenguaje de programación legible para los humanos y que traduce las instrucciones a un programa ejecutable legible para la máquina.
COOKIE	Mensaje que un servidor de Web envía a un navegador. El navegador almacena el mensaje en un archivo de texto. Este mensaje es enviado de regreso al servidor cada vez que el navegador solicita una página al servidor Web.
CROSS-PLATAFORM	Multiplataforma, plataforma cruzada. Capaz de operar

	en una red en la cual las estaciones de trabajo son de diferentes tipos.
DRIVER	Controlador. Archivo que contiene la información que necesita un programa .
ESTACIÓN DE TRABAJO	En una red de área local, computadora de escritorio que ejecuta programas de aplicación y sirve como punto de acceso a la red.
GUI	Del inglés Graphical User Interface. Interfaz gráfica de usuario, son los componentes del sistema con los que interactúa el usuario final.
HARDWARE	Componentes electrónicos, tarjetas, periféricos y equipo que conforman un sistema de computación. El hardware se distingue del software (programas), que es el que les indica a los componentes mencionados lo que deben hacer.
HIPERLIGA	También llamada hipervínculo. En un sistema de hipertexto, palabra o frase subrayada o enfatizada de alguna otra forma que, cuando se hace clic sobre ella, se despliega otro documento.
HIPERTEXTO	Método de preparar y publicar texto ideal para la computadora, en el cual los lectores pueden elegir su propia ruta a través del material.
HTML	Es el lenguaje utilizado para la creación de páginas Web. Sus siglas vienen del inglés HyperText Markup Language.
HTTP	Protocolo de comunicación utilizado para la transferencia de hipertexto. Sus siglas vienen del inglés HyperText Transfer Protocol.
INTERFAZ	La porción de un programa que interactúa con el usuario.
INTERNET	Red mundial de computadoras que transporta datos y hace posible el intercambio de información. Internet es en último término un conjunto de servidores que proporcionan servicios de transferencia de archivos, correo electrónico o páginas Web, entre otros.
JAVA BEAN	Clase Java que tiene las características: Clase pública, constructor sin argumentos, propiedades y contiene métodos “get” y “set”. Por cada propiedad deben existir dos métodos, uno “set” para dar valores a la propiedad y uno “get” para obtenerlos.
JDBC	Del inglés Java Database Connectivity. API de Java que brinda a los programa la posibilidad de interactuar con una base de datos y ejecutar instrucciones sql.
JDK	Siglas de Java Development Kit. Conjunto de herramientas de desarrollo que provee la tecnología Java.
JVM	Máquina virtual de Java. Intérprete y ambiente de

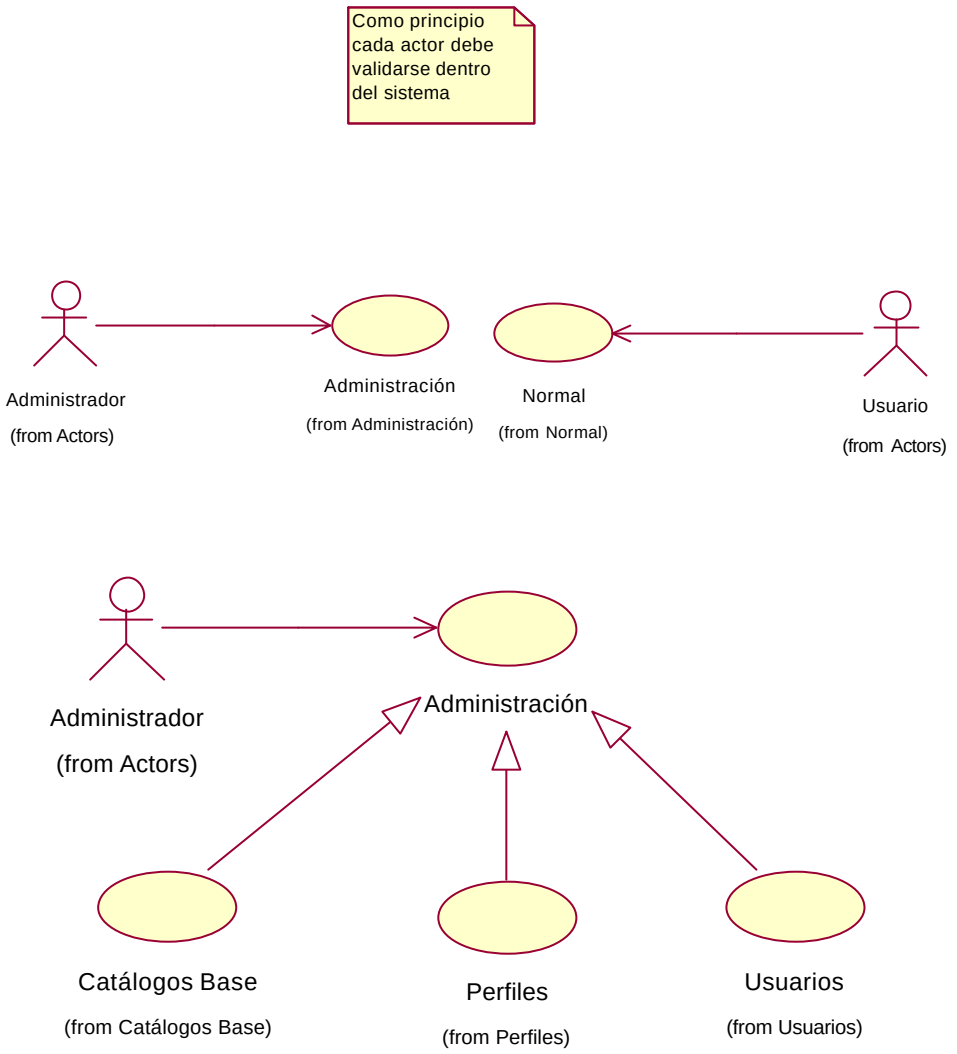
	tiempo de ejecución de Java para applets y aplicaciones de Java.
ODBC	Abreviatura del inglés Open DataBase Connectivity. Método estándar para el acceso a bases de datos desarrollado por Microsoft.
PLATAFORMA	Estándar de hardware, como el de las computadoras compatibles con la PC o el de Macintosh. Los dispositivos o programas creados para una plataforma no corren en otras
PROGRAMA	Lista de instrucciones, escritas en un lenguaje de programación que ejecuta una computadora para que la máquina actúe de una forma determinada. El término es sinónimo de software.
PROTOCOLO	Estándar que especifica el formato de los datos, además de las reglas que habrán de seguirse.
SCRIPT	Secuencia de comandos, serie de instrucciones parecidas a una macro y escritas en texto simple, que le indican a un programa cómo ejecutar un procedimiento específico.
SERVIDOR	Genéricamente, dispositivo de un sistema que resuelve las peticiones de otros elementos del sistema, denominados clientes. Suele utilizarse para mantener datos centralizados o para gestionar recursos compartidos.
SOFTWARE	Conjunto de instrucciones de computadora que al ser ejecutadas proporcionan el resultado y el comportamiento deseado.
STORED PROCEDURE	Programa conformado de una secuencia de sentencias SQL que se encuentra almacenado en la base de datos y es usado para desempeñar una tarea particular.
WWW	Sistema mundial de hipertexto que utiliza Internet como mecanismo de transporte.

2.-Documentos anexos

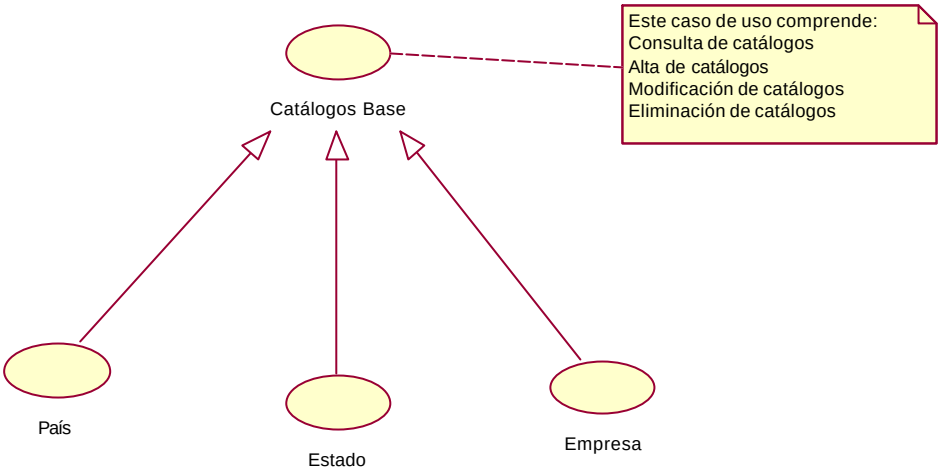
Los documentos involucrados en el desarrollo del sistema que se muestran a continuación son:

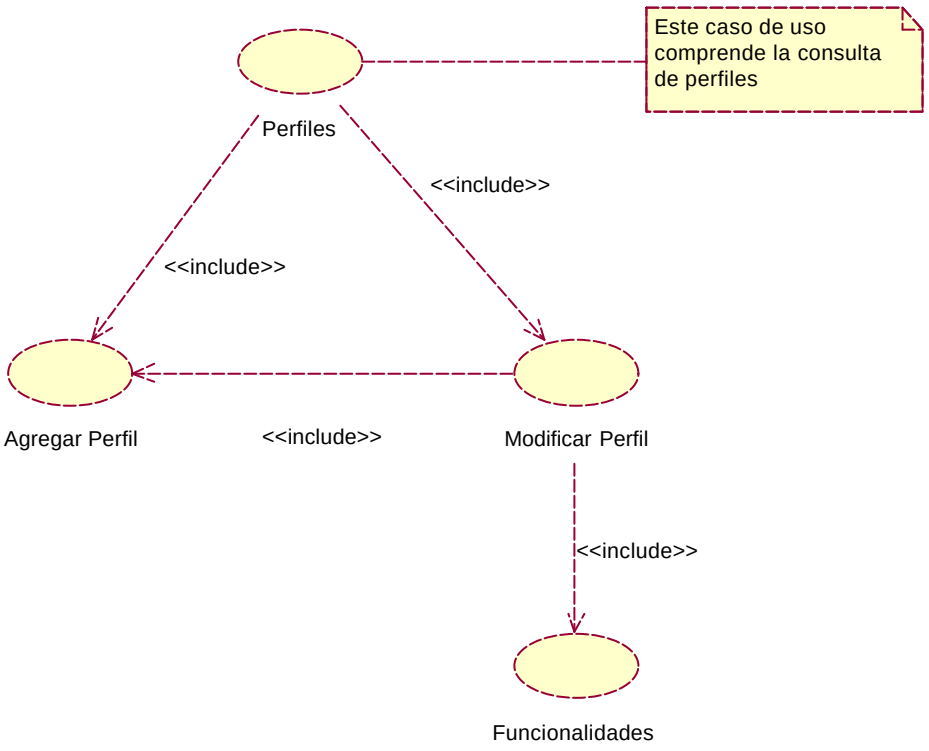
Casos de uso
Diseño de la base de datos (lógico y físico)
Instrucciones para probar el sistema

Casos de uso

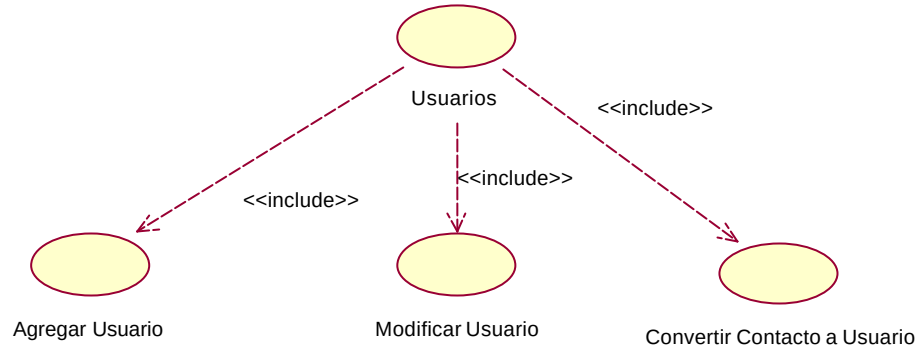


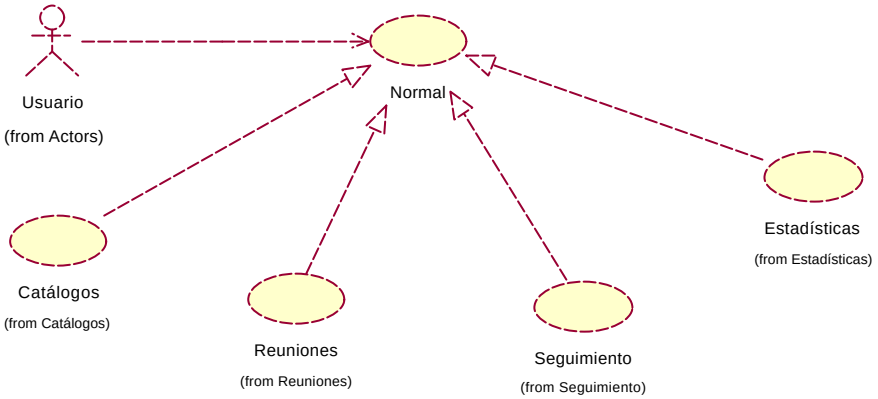
Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.



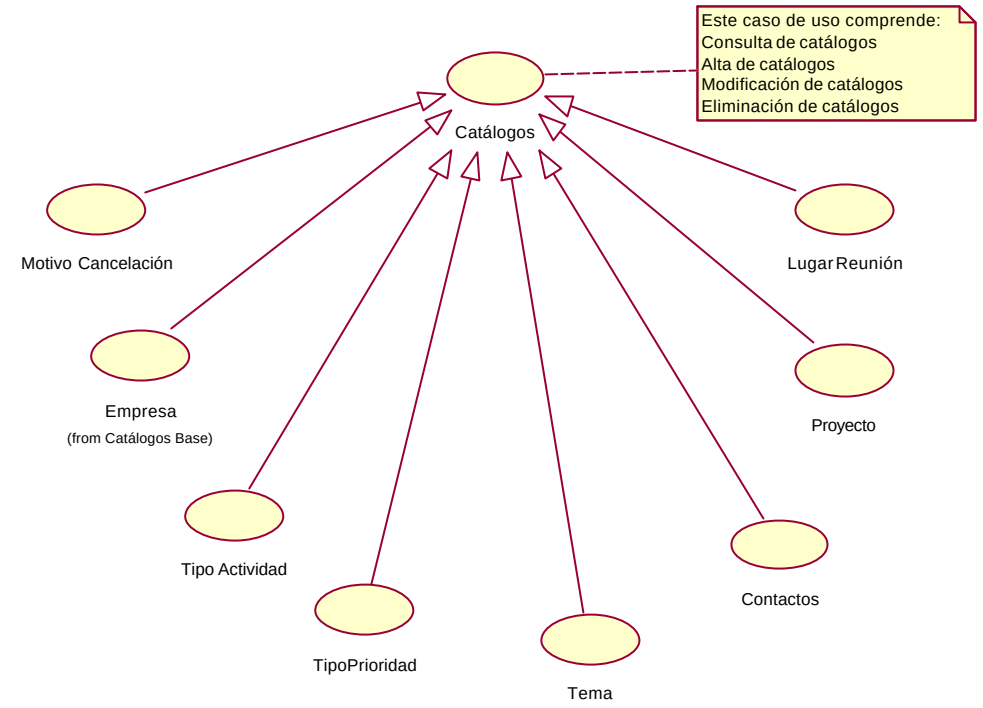


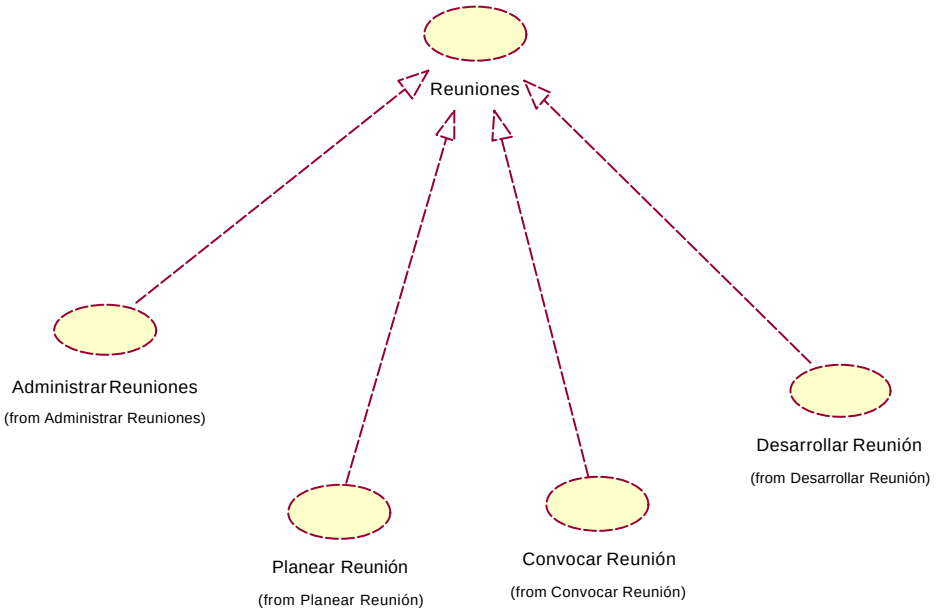
Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.



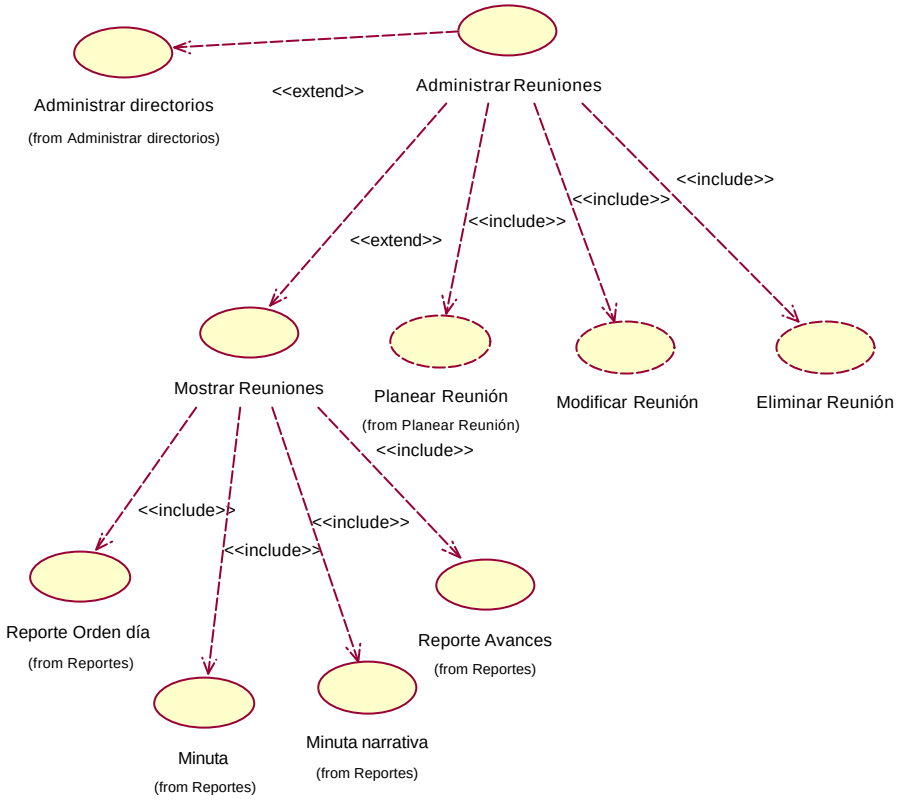


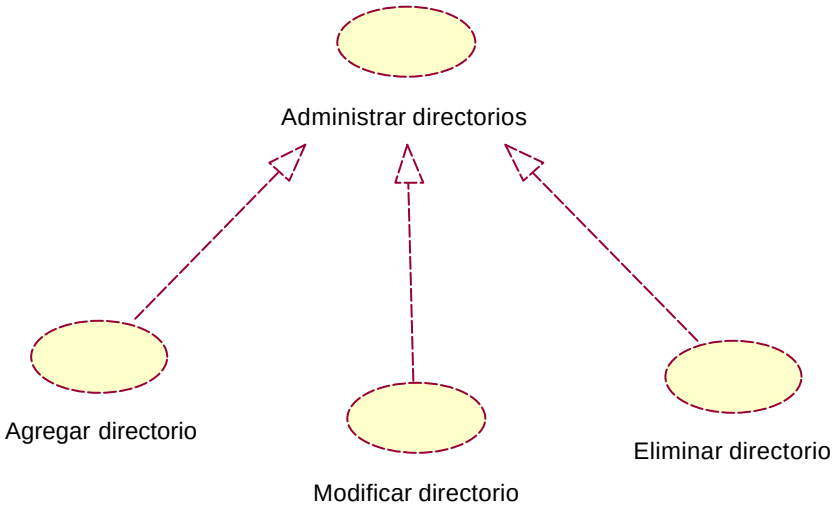
Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.



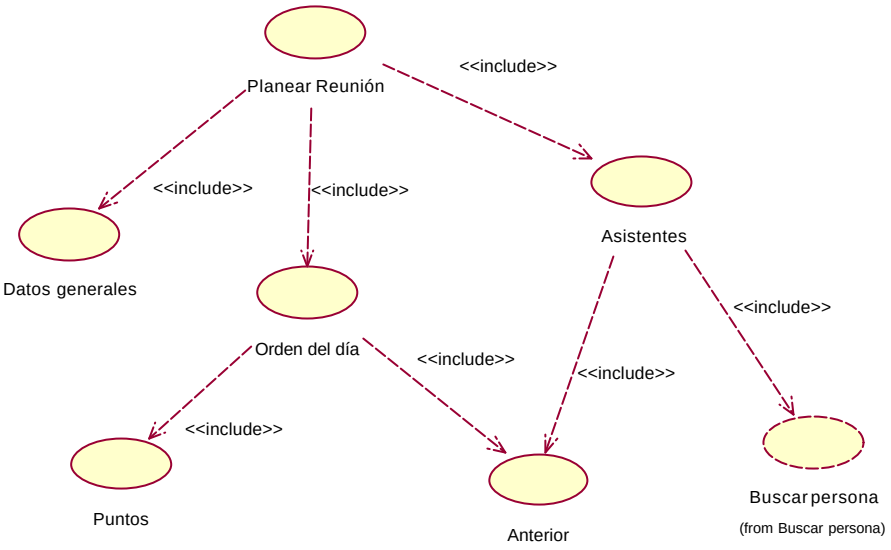


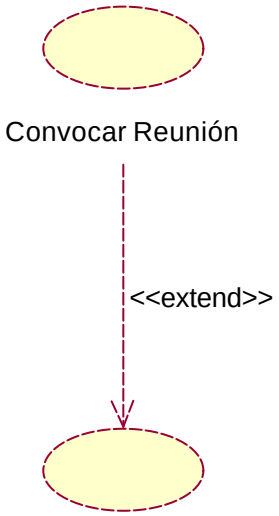
Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.



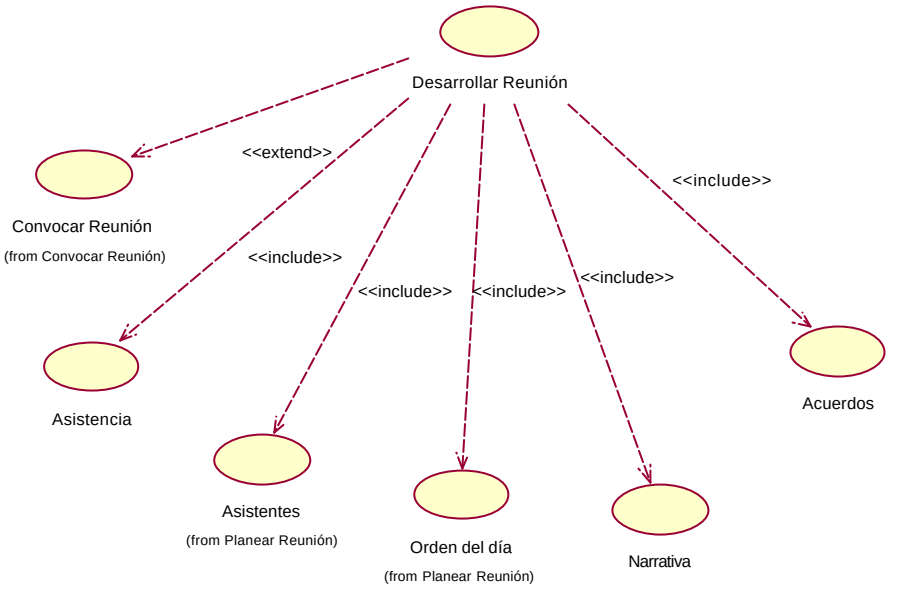


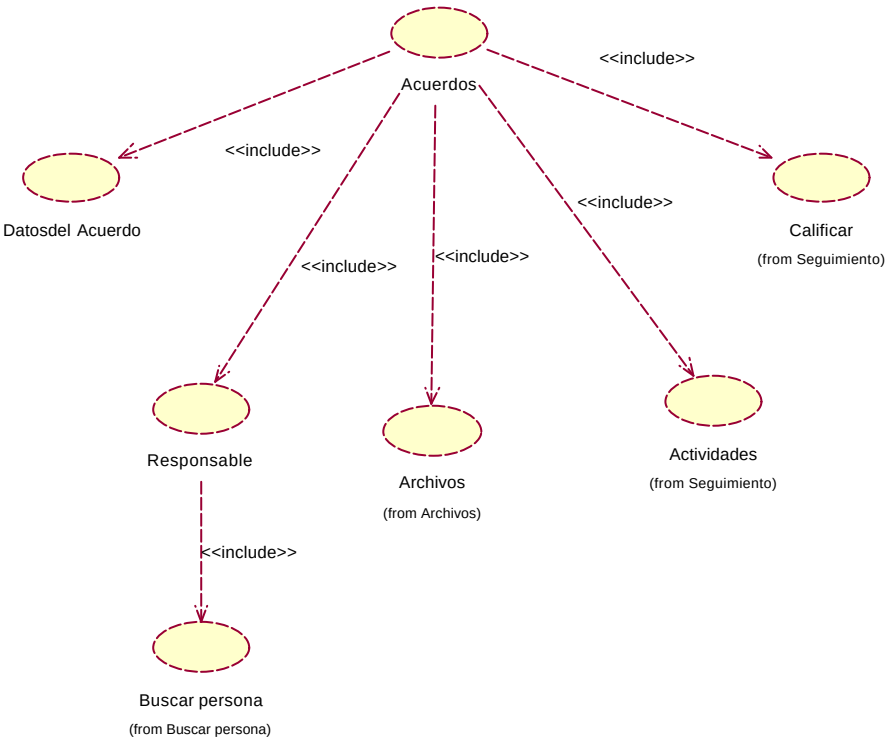
Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

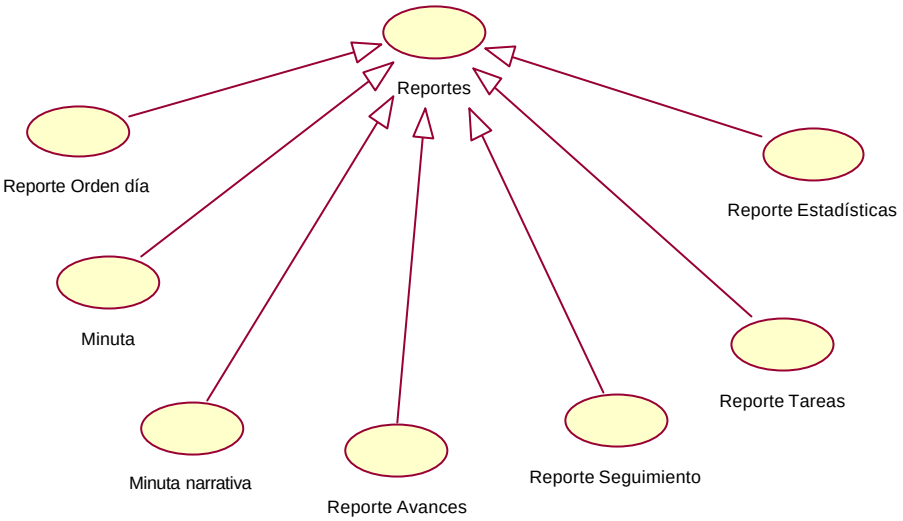


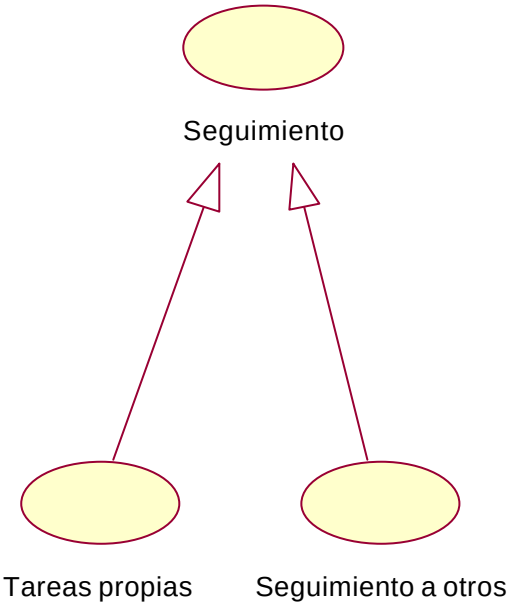


Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

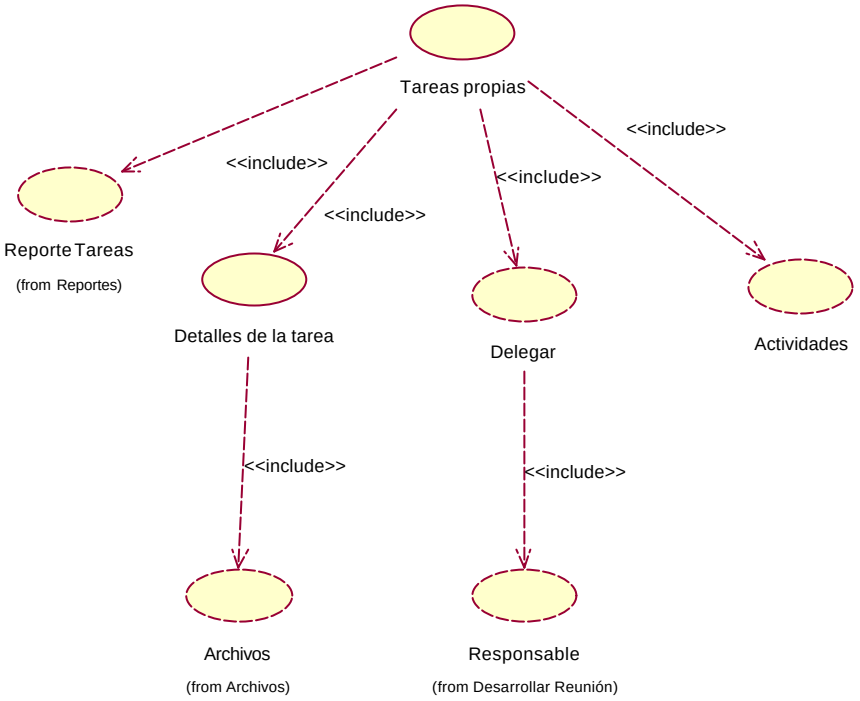

Desarrollar Reunión
<<extend>>
Convocar Reunión
(from Convocar Reunión)
<<include>>
Asistencia
<<include>>
Asistentes
(from Planear Reunión)
<<include>>
Orden del día
(from Planear Reunión)
<<include>>
Narrativa
<<include>>
Acuerdos

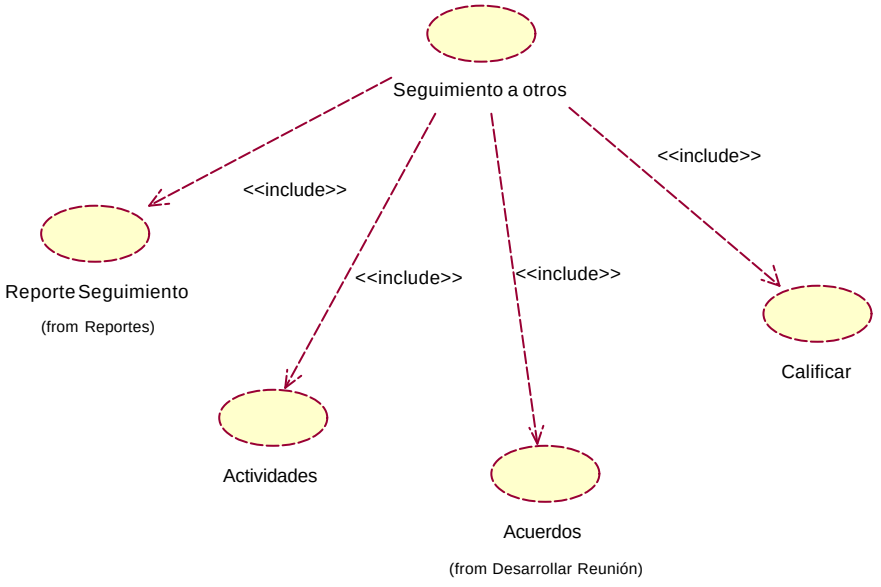

Acuerdos
<<include>>
Datosdel Acuerdo
<<include>>
Responsable
<<include>>
Archivos
(from Archivos)
<<include>>
Actividades
(from Seguimiento)
<<include>>
Calificar
(from Seguimiento)
<<include>>
Buscar persona
(from Buscar persona)



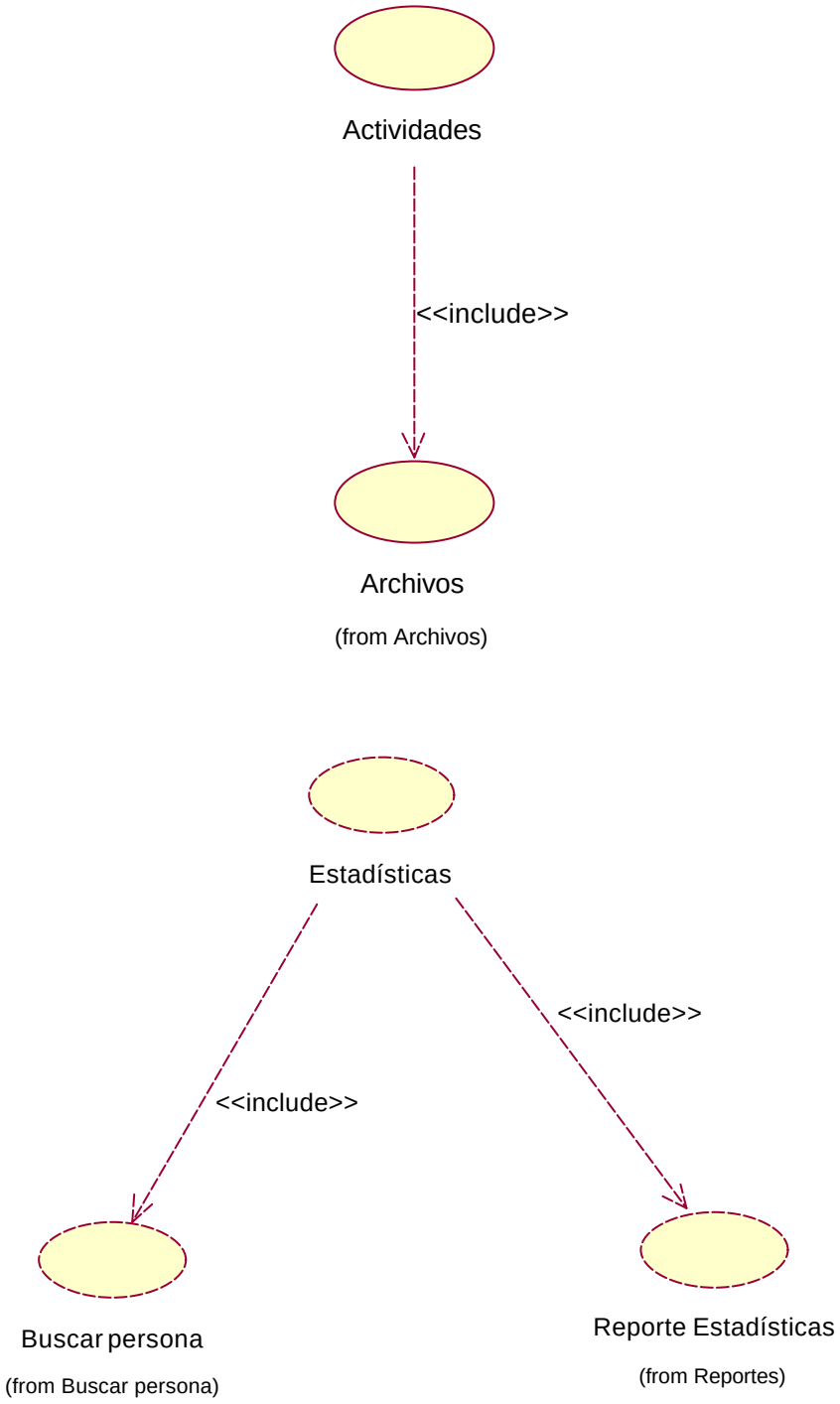


Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.



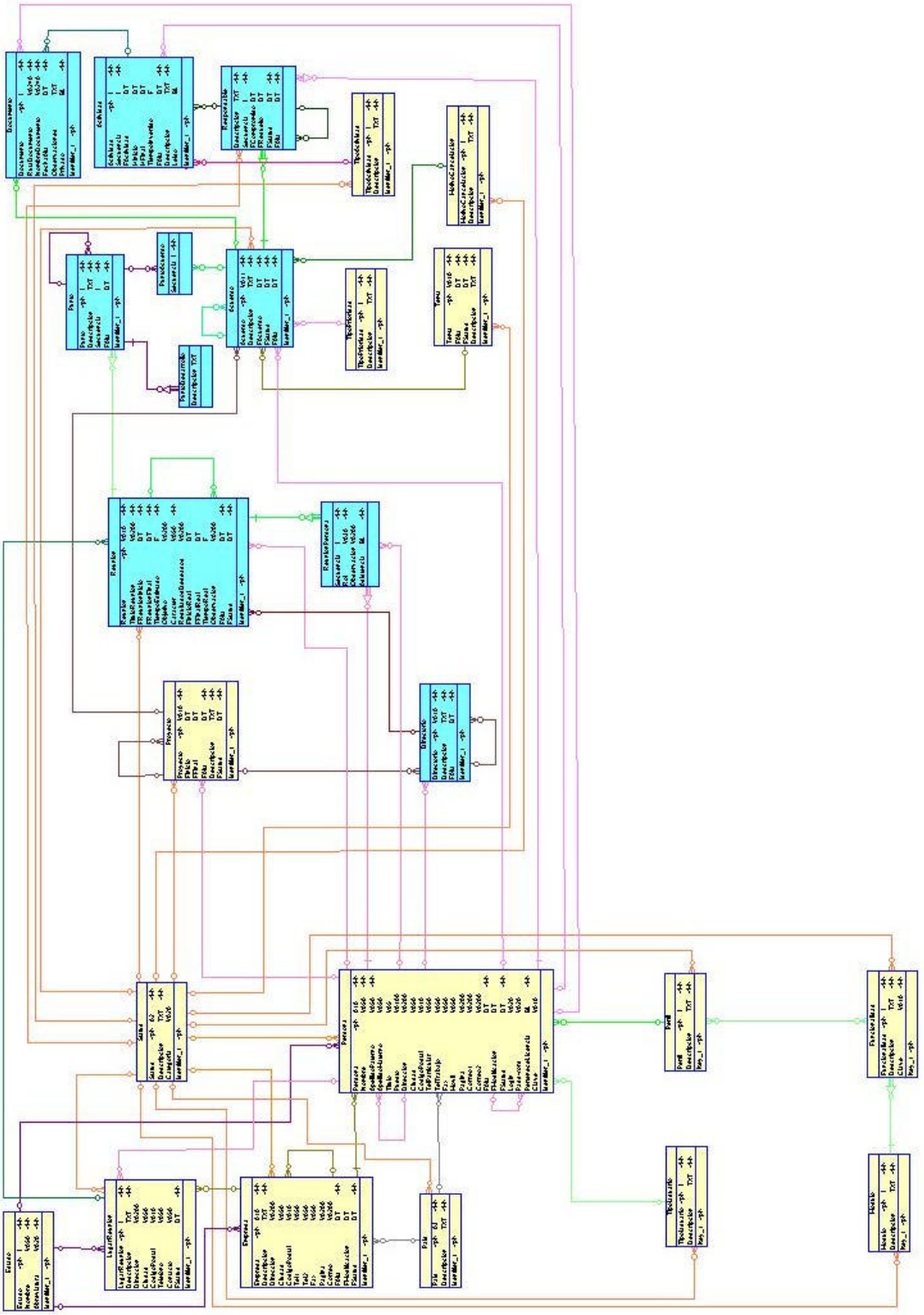


Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

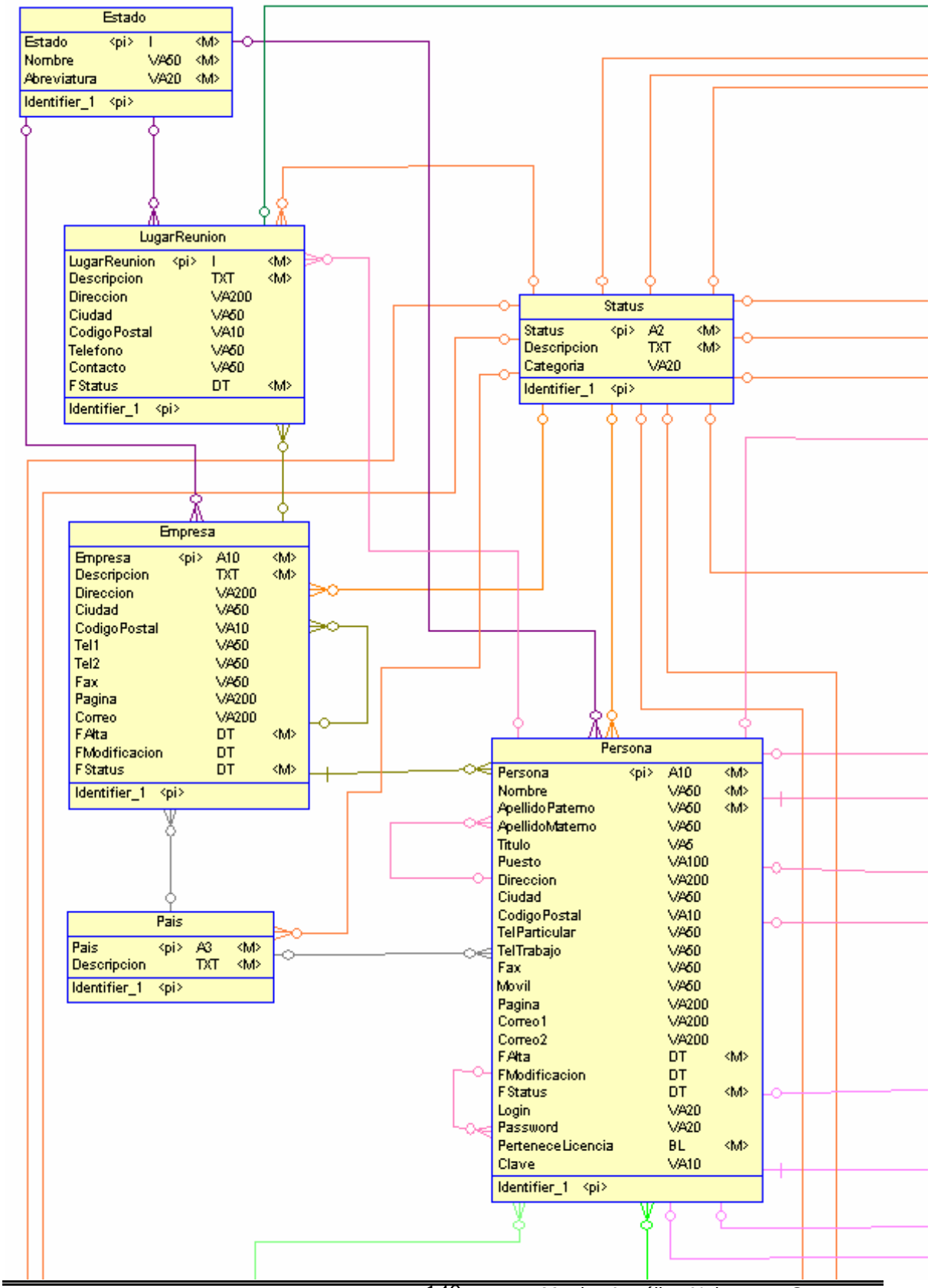

Actividades
<<include>>
Archivos
(from Archivos)
Estadísticas
<<include>>
<<include>>
Buscar persona
(from Buscar persona)
Reporte Estadísticas
(from Reportes)

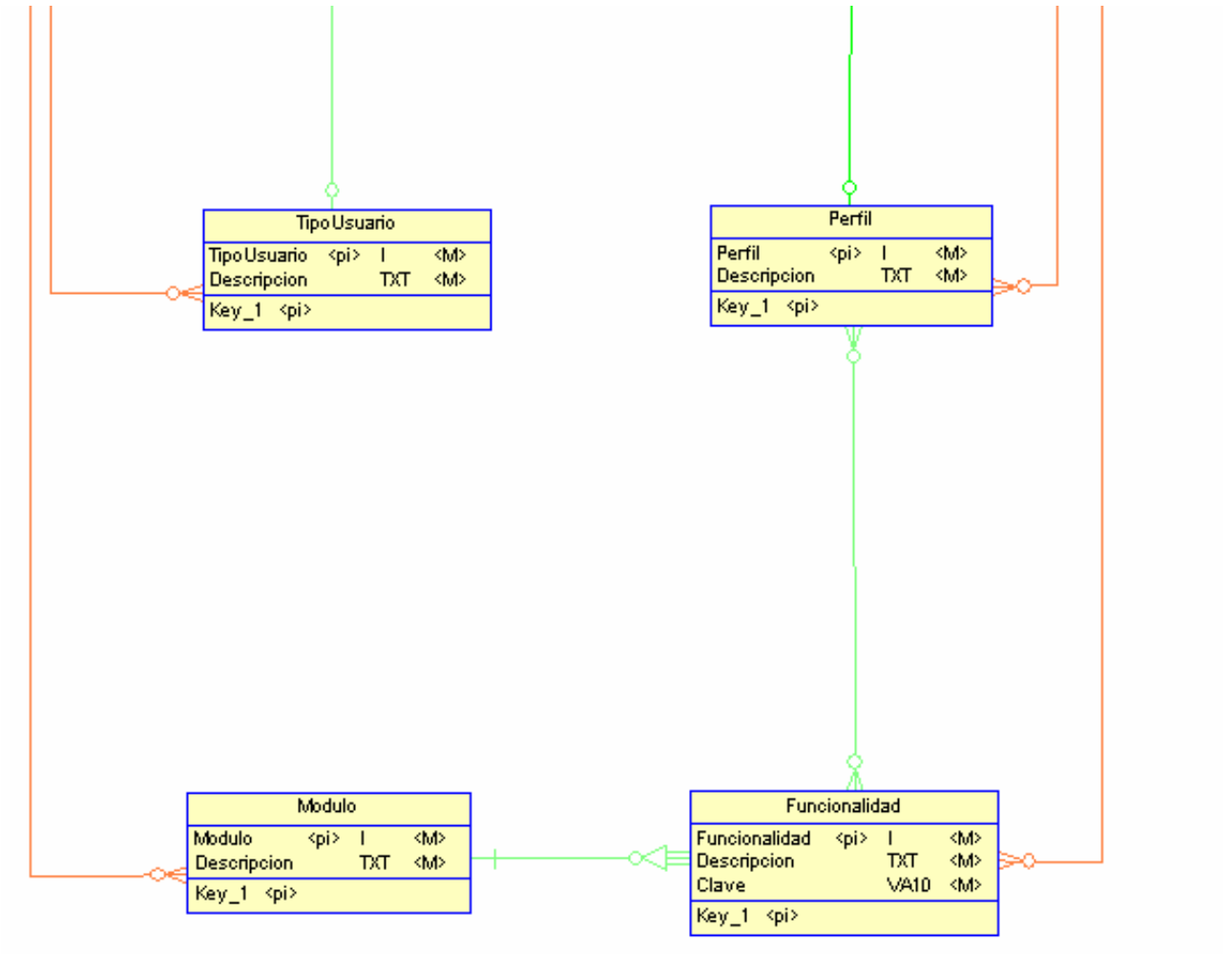
Base de datos

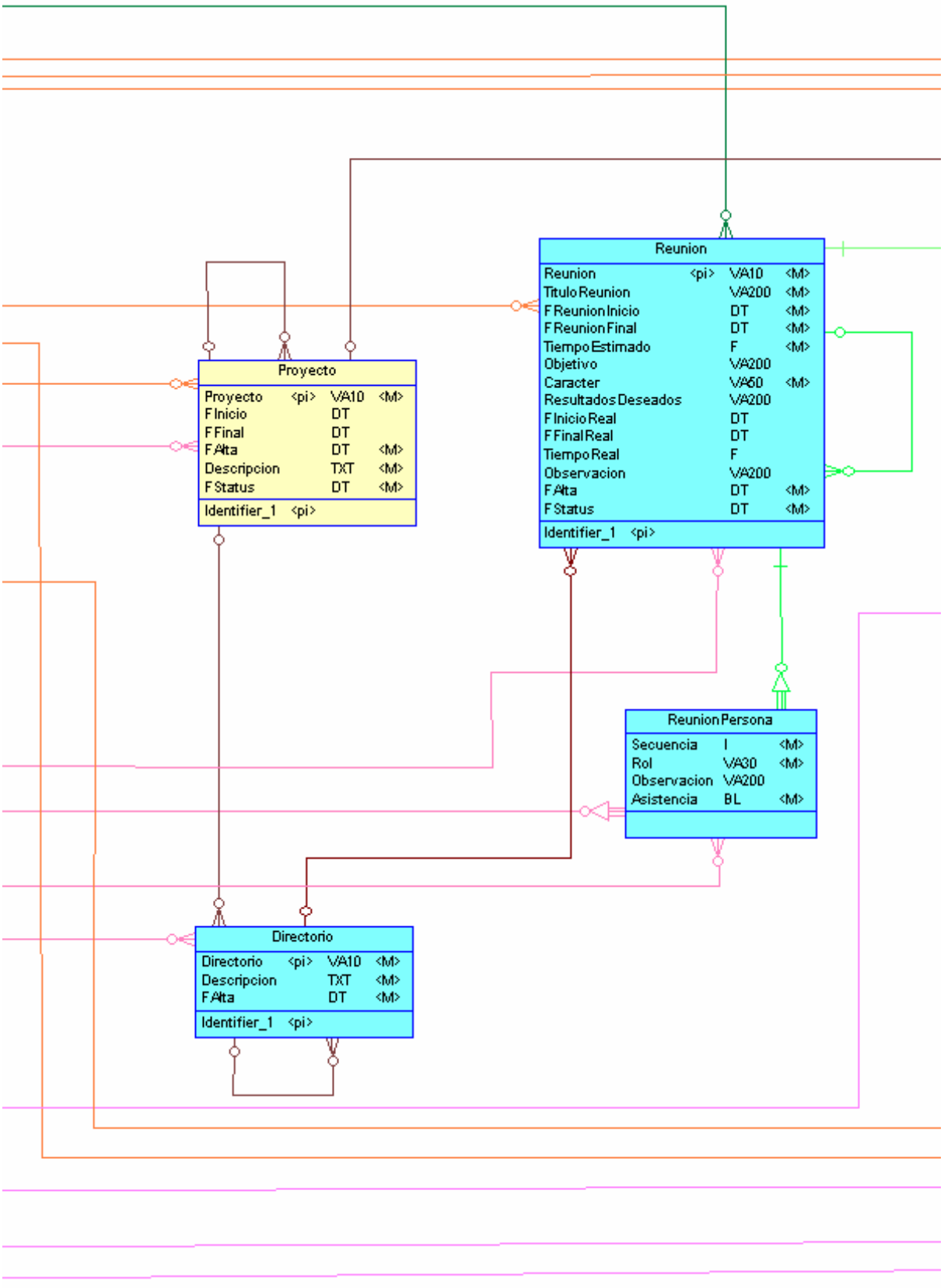
Modelo lógico

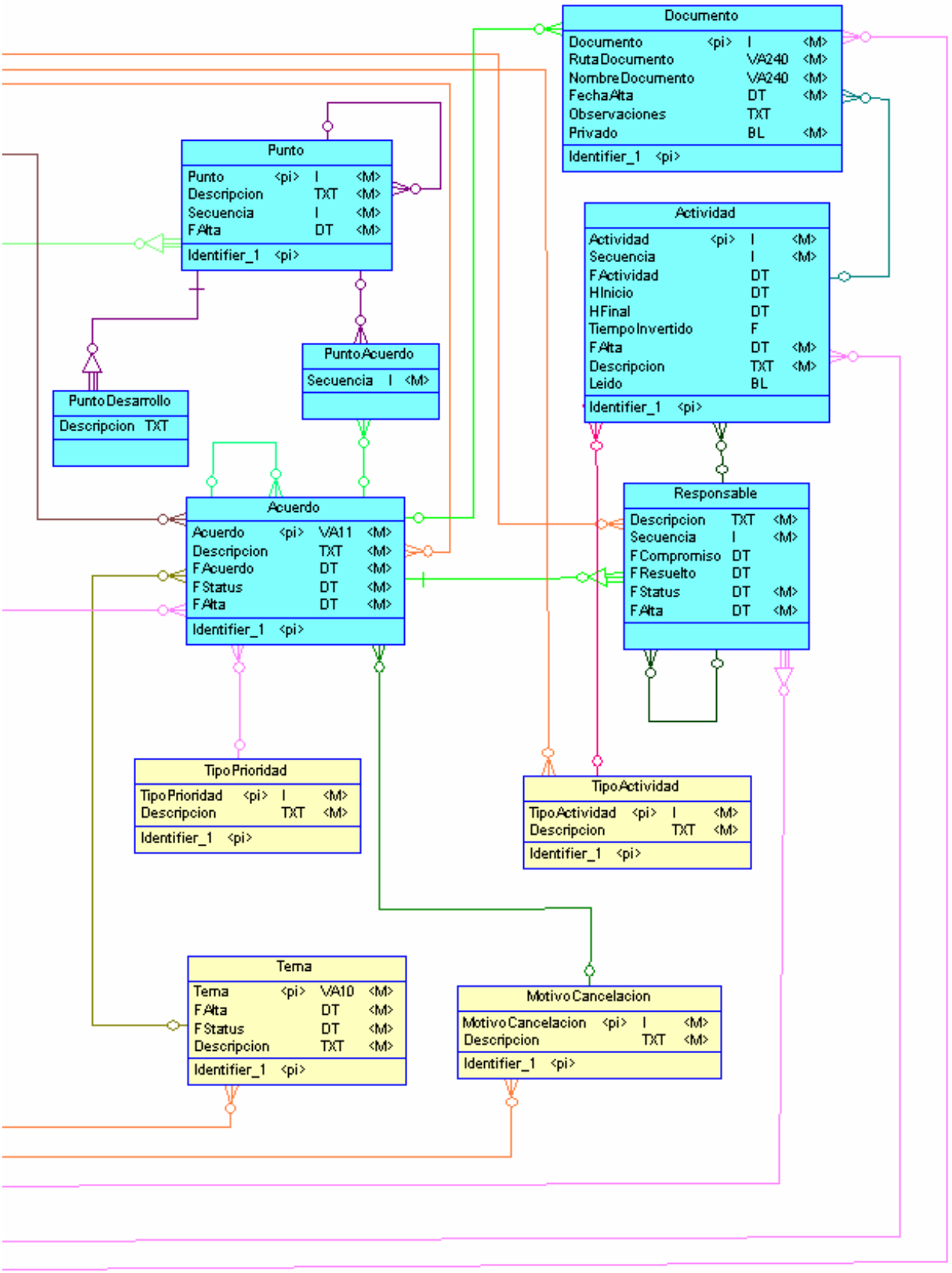


Martha Angélica Nakayama Cervantes
Facultad de Ingeniería
U.N.A.M.

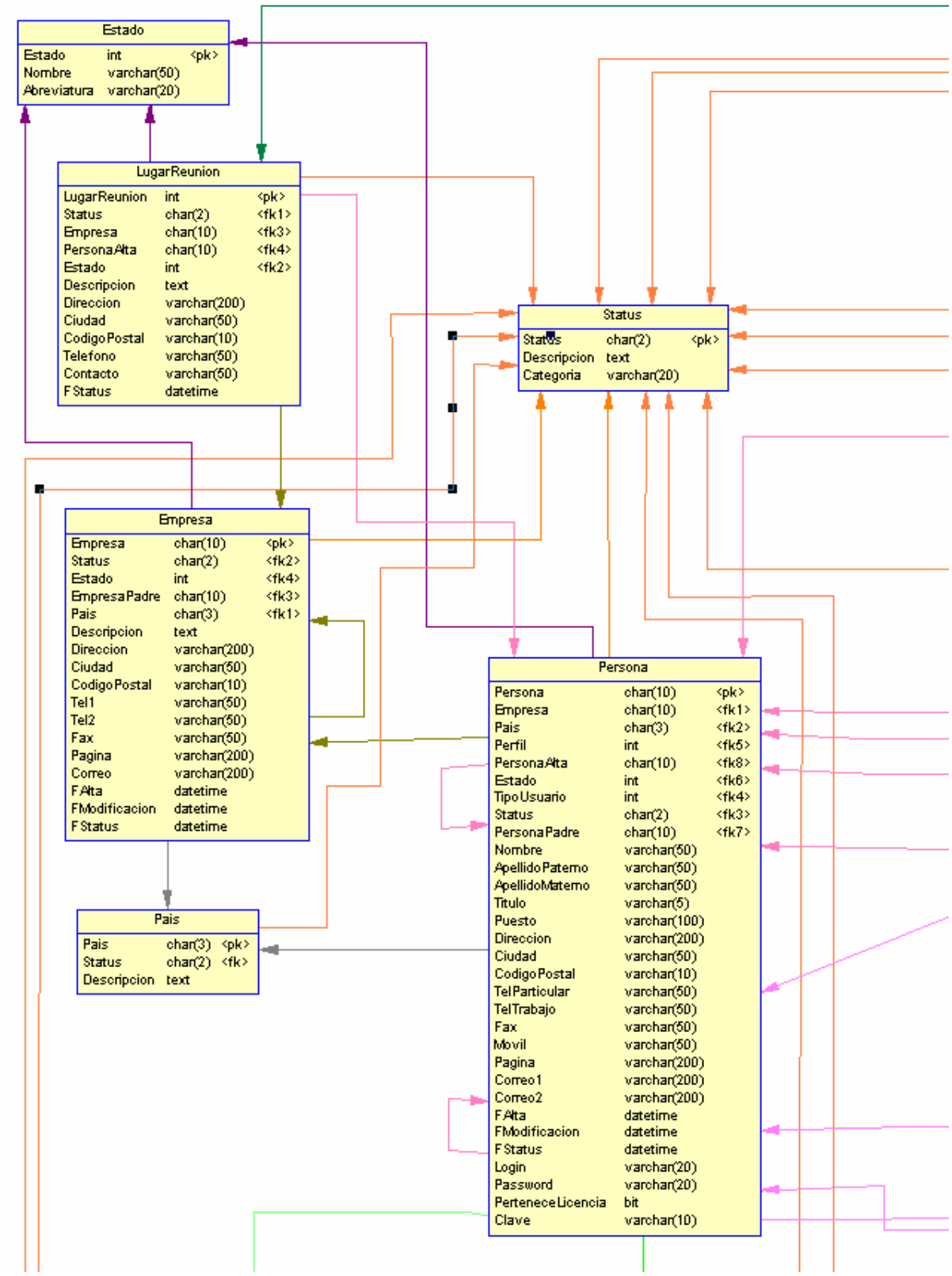

Estado
Estado <pi> I <M>
Nombre VA50 <M>
Abreviatura VA20 <M>
Identifier_1 <pi>
LugarReunion
LugarReunion <pi> I <M>
Descripcion TXT <M>
Direccion VA200
Ciudad VA50
CodigoPostal VA10
Telefono VA50
Contacto VA50
FStatus DT <M>
Identifier_1 <pi>
Status
Status <pi> A2 <M>
Descripcion TXT <M>
Categoria VA20
Identifier_1 <pi>
Empresa
Empresa <pi> A10 <M>
Descripcion TXT <M>
Direccion VA200
Ciudad VA50
CodigoPostal VA10
Tel1 VA50
Tel2 VA50
Fax VA50
Pagina VA200
Correo VA200
FAlta DT <M>
FModificacion DT
FStatus DT <M>
Identifier_1 <pi>
Persona
Persona <pi> A10 <M>
Nombre VA50 <M>
ApellidoPaterno VA50 <M>
ApellidoMaterno VA50
Titulo VA5
Puesto VA100
Direccion VA200
Ciudad VA50
CodigoPostal VA10
TelParticular VA50
TelTrabajo VA50
Fax VA50
Movil VA50
Pagina VA200
Correo1 VA200
Correo2 VA200
FAlta DT <M>
FModificacion DT
FStatus DT <M>
Login VA20
Password VA20
PerteneceLicencia BL <M>
Clave VA10
Identifier_1 <pi>
Pais
Pais <pi> A3 <M>
Descripcion TXT <M>
Identifier_1 <pi>

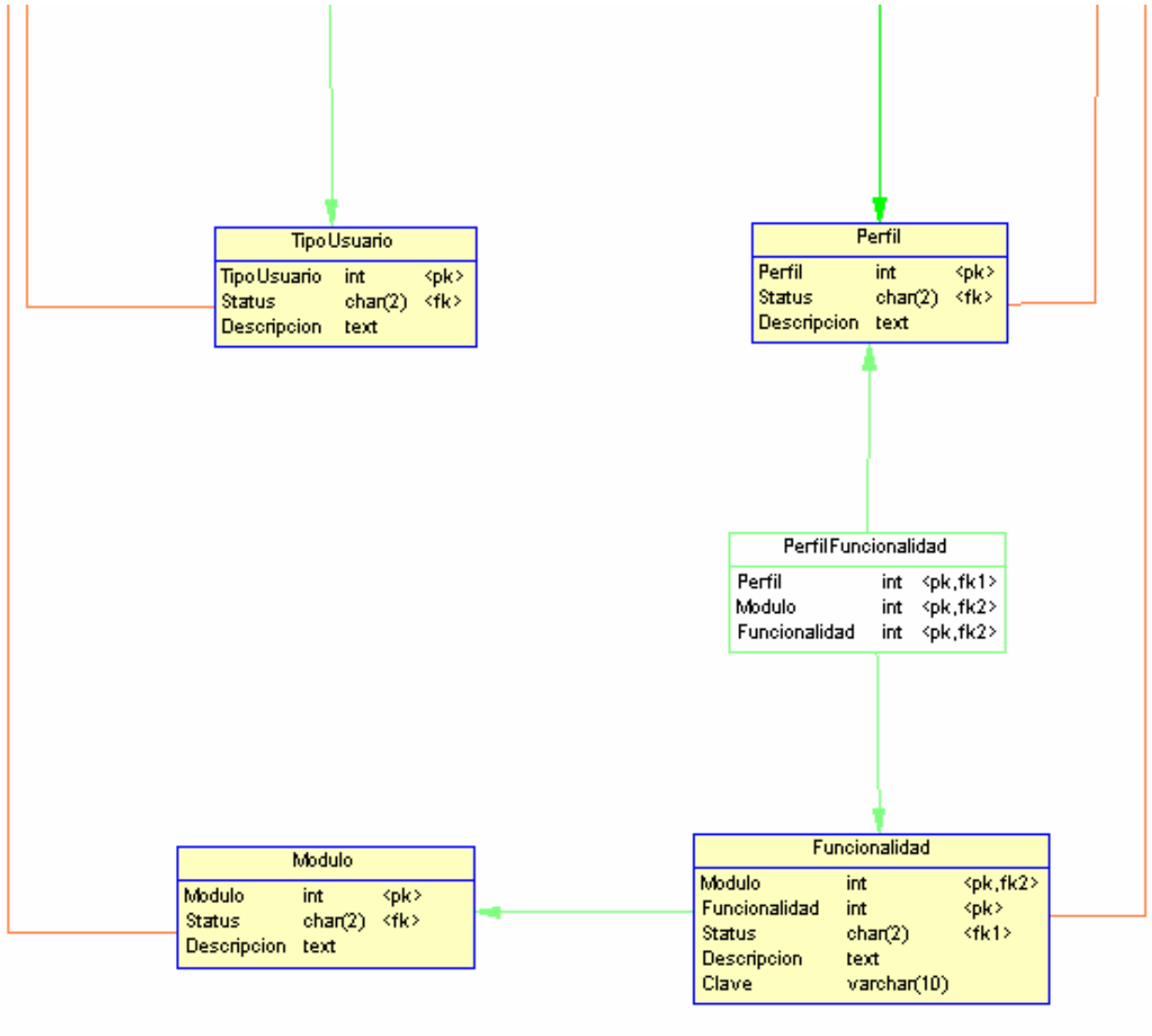

TipoUsuario
TipoUsuario <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Perfil
Perfil <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Modulo
Modulo <pi> I <M>
Descripcion TXT <M>
Key_1 <pi>
Funcionalidad
Funcionalidad <pi> I <M>
Descripcion TXT <M>
Clave VA10 <M>
Key_1 <pi>

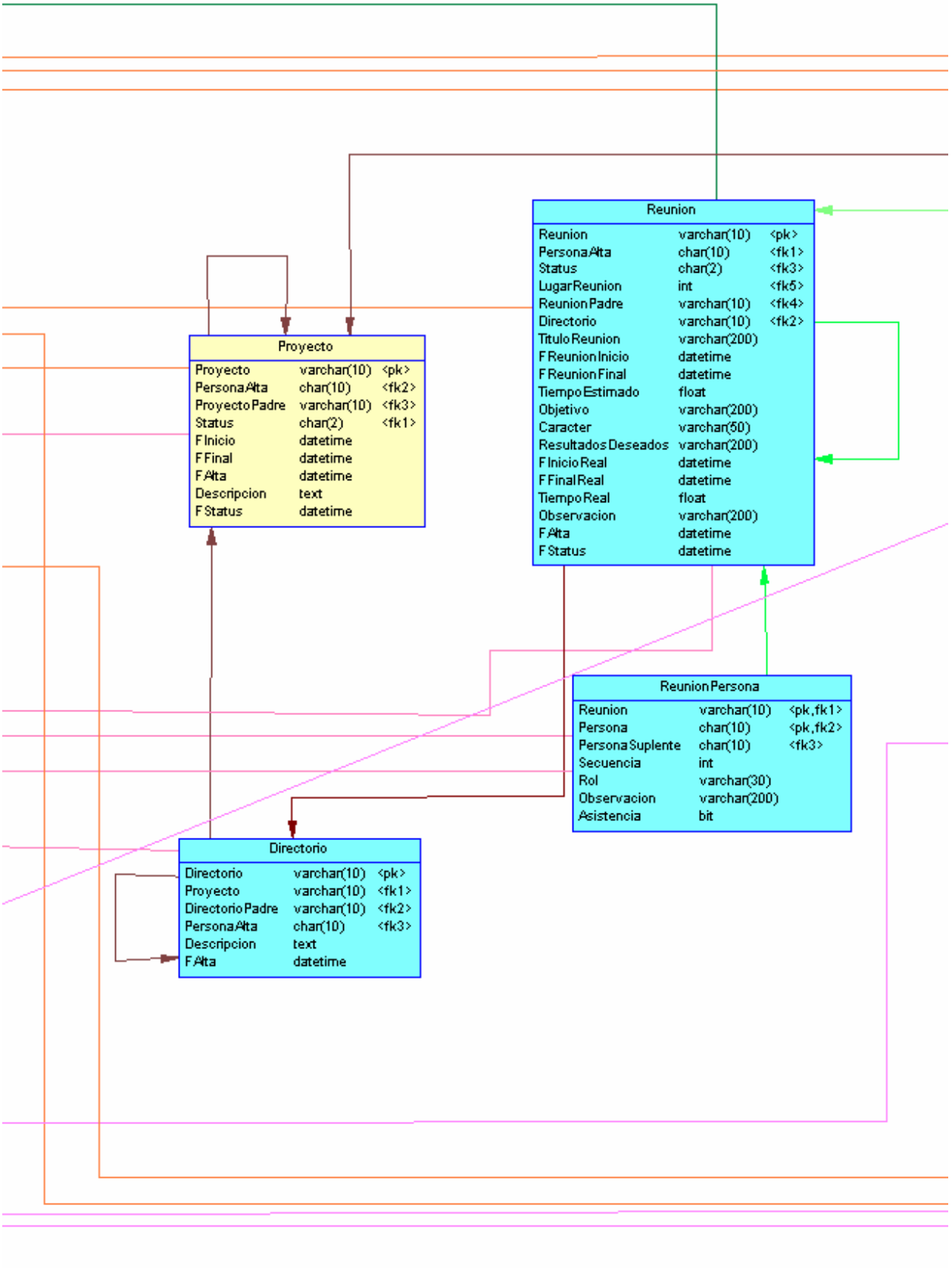

Reunion
Reunion <pi> VA10 <M>
Titulo Reunion VA200 <M>
F Reunion Inicio DT <M>
F Reunion Final DT <M>
Tiempo Estimado F <M>
Objetivo VA200
Caracter VA50 <M>
Resultados Deseados VA200
F Inicio Real DT
F Final Real DT
Tiempo Real F
Observacion VA200
F Alta DT <M>
F Status DT <M>
Identifier_1 <pi>
Proyecto
Proyecto <pi> VA10 <M>
F Inicio DT
F Final DT
F Alta DT <M>
Descripcion TXT <M>
F Status DT <M>
Identifier_1 <pi>
Reunion Persona
Secuencia I <M>
Rol VA30 <M>
Observacion VA200
Asistencia BL <M>
Directorio
Directorio <pi> VA10 <M>
Descripcion TXT <M>
F Alta DT <M>
Identifier_1 <pi>

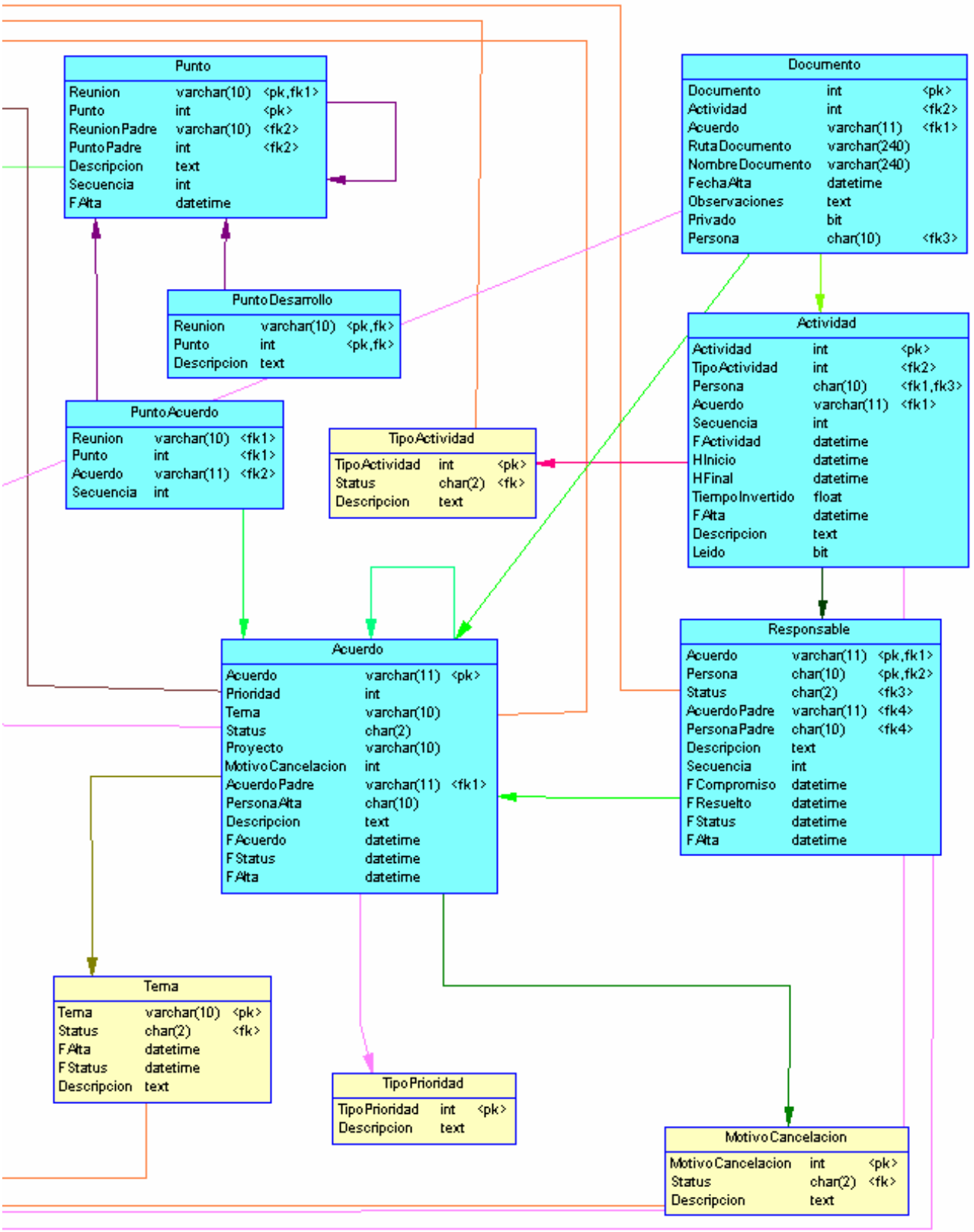

Documento
Documento <pi> I <M>
RutaDocumento VA240 <M>
NombreDocumento VA240 <M>
FechaAlta DT <M>
Observaciones TXT
Privado BL <M>
Identifier_1 <pi>
Punto
Punto <pi> I <M>
Descripcion TXT <M>
Secuencia I <M>
FAlta DT <M>
Identifier_1 <pi>
Actividad
Actividad <pi> I <M>
Secuencia I <M>
FActividad DT
HInicio DT
HFinal DT
TiempoInvertido F
FAlta DT <M>
Descripcion TXT <M>
Leido BL
Identifier_1 <pi>
PuntoAcuerdo
Secuencia I <M>
PuntoDesarrollo
Descripcion TXT
Acuerdo
Acuerdo <pi> VA11 <M>
Descripcion TXT <M>
FAcuerdo DT <M>
FStatus DT <M>
FAlta DT <M>
Identifier_1 <pi>
Responsable
Descripcion TXT <M>
Secuencia I <M>
FCompromiso DT
FResuelto DT
FStatus DT <M>
FAlta DT <M>
TipoPrioridad
TipoPrioridad <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>
TipoActividad
TipoActividad <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>
Tema
Tema <pi> VA10 <M>
FAlta DT <M>
FStatus DT <M>
Descripcion TXT <M>
Identifier_1 <pi>
MotivoCancelacion
MotivoCancelacion <pi> I <M>
Descripcion TXT <M>
Identifier_1 <pi>

Modelo Físico




TipoUsuario
TipoUsuario int <pk>
Status char(2) <fk>
Descripcion text
Perfil
Perfil int <pk>
Status char(2) <fk>
Descripcion text
PerfilFuncionalidad
Perfil int <pk,fk1>
Modulo int <pk,fk2>
Funcionalidad int <pk,fk2>
Modulo
Modulo int <pk>
Status char(2) <fk>
Descripcion text
Funcionalidad
Modulo int <pk,fk2>
Funcionalidad int <pk>
Status char(2) <fk1>
Descripcion text
Clave varchar(10)


Reunion
Reunion varchar(10) <pk>
PersonaAlta char(10) <fk1>
Status char(2) <fk3>
LugarReunion int <fk5>
ReunionPadre varchar(10) <fk4>
Directorio varchar(10) <fk2>
TituloReunion varchar(200)
FReunionInicio datetime
FReunionFinal datetime
TiempoEstimado float
Objetivo varchar(200)
Caracter varchar(50)
ResultadosDeseados varchar(200)
FInicioReal datetime
FFinalReal datetime
TiempoReal float
Observacion varchar(200)
FAlta datetime
FStatus datetime
Proyecto
Proyecto varchar(10) <pk>
PersonaAlta char(10) <fk2>
ProyectoPadre varchar(10) <fk3>
Status char(2) <fk1>
FInicio datetime
FFinal datetime
FAlta datetime
Descripcion text
FStatus datetime
ReunionPersona
Reunion varchar(10) <pk,fk1>
Persona char(10) <pk,fk2>
PersonaSuplente char(10) <fk3>
Secuencia int
Rol varchar(30)
Observacion varchar(200)
Asistencia bit
Directorio
Directorio varchar(10) <pk>
Proyecto varchar(10) <fk1>
DirectorioPadre varchar(10) <fk2>
PersonaAlta char(10) <fk3>
Descripcion text
FAlta datetime


Punto
Reunion varchar(10) <pk,fk1>
Punto int <pk>
ReunionPadre varchar(10) <fk2>
PuntoPadre int <fk2>
Descripcion text
Secuencia int
FAlta datetime
Documento
Documento int <pk>
Actividad int <fk2>
Acuerdo varchar(11) <fk1>
RutaDocumento varchar(240)
NombreDocumento varchar(240)
FechaAlta datetime
Observaciones text
Privado bit
Persona char(10) <fk3>
PuntoDesarrollo
Reunion varchar(10) <pk,fk>
Punto int <pk,fk>
Descripcion text
Actividad
Actividad int <pk>
TipoActividad int <fk2>
Persona char(10) <fk1,fk3>
Acuerdo varchar(11) <fk1>
Secuencia int
FActividad datetime
HInicio datetime
HFinal datetime
TiempoInvertido float
FAlta datetime
Descripcion text
Leido bit
PuntoAcuerdo
Reunion varchar(10) <fk1>
Punto int <fk1>
Acuerdo varchar(11) <fk2>
Secuencia int
TipoActividad
TipoActividad int <pk>
Status char(2) <fk>
Descripcion text
Responsable
Acuerdo varchar(11) <pk,fk1>
Persona char(10) <pk,fk2>
Status char(2) <fk3>
AcuerdoPadre varchar(11) <fk4>
PersonaPadre char(10) <fk4>
Descripcion text
Secuencia int
FCompromiso datetime
FResuelto datetime
FStatus datetime
FAlta datetime
Acuerdo
Acuerdo varchar(11) <pk>
Prioridad int
Tema varchar(10)
Status char(2)
Proyecto varchar(10)
MotivoCancelacion int
AcuerdoPadre varchar(11) <fk1>
PersonaAlta char(10)
Descripcion text
FAcuerdo datetime
FStatus datetime
FAlta datetime
Tema
Tema varchar(10) <pk>
Status char(2) <fk>
FAlta datetime
FStatus datetime
Descripcion text
TipoPrioridad
TipoPrioridad int <pk>
Descripcion text
MotivoCancelacion
MotivoCancelacion int <pk>
Status char(2) <fk>
Descripcion text

Bibliografía

Pressman, Roger S.
Ingeniería del software un enfoque práctico. Tercera edición
Mc Graw Hill, 1993

Pfleeger, Shari Lawrence
Ingeniería de Software, teoría y práctica. Primera edición
Pearson Education, 2002

Pfaffenberger, Bryan
Diccionario de términos de computación
Prentice hall, 1999

Kulak, Daryl, Guiney Eamonn
Use cases: requirements in context. Second Edition.
Addison-Wesley, 2004

I. Jacobson, G. Booch, J. Rumbaugh
El proceso unificado de desarrollo de software
Pearson Educación, 2000

Date, C. J.
An introduction to database systems, Sixth Edition
Addison-Wesley, 1995

Elmasari Ramez, B. Navathe Shamkant
Sistemas de Bases de Datos, Segunda Edición
Addison Wesley Longman de México, 2000

Griffith Arthur
JAVA Master Referente
IDG Books World, Inc, 1998

Diversas páginas de Internet utilizando el buscador Google
http://www.google.com.mx