1. Conceptos Introductorios.

1.1 La arquitectura de 4+1 vistas.

Antes de hablar de la arquitectura de 4 +1 vistas, es importante conocer sus antecedentes, para lo cual primero se debe mencionar a la Arquitectura de Software.

La Arquitectura de Software consiste en la manera de resumir un conjunto de decisiones respecto a la forma en que se organiza un sistema de información, dichas decisiones se enfocan en los elementos del sistema: quienes son, que son, que hacen, como se comportan, como se relacionan y como se combinan; con lo cual esta arquitectura proporciona un marco de referencia para la construcción del software haciendo uso de un conjunto “de patrones”.

El término “patrón”, surge del mundo de la Ingeniería Civil, y se debe al arquitecto Christopher Alexander, que en 1979 en su libro titulado The Timeless Way of Building, propone la realización de un conjunto de patrones de diseño para edificios, estos patrones recopilan la experiencia de generaciones de arquitectos en la construcción, con lo cual se garantiza una mejor calidad, por lo que un patrón propone una posible solución general pero correcta a un problema de diseño dentro de un contexto en específico.

Así esta idea es trasladada al mundo del desarrollo de software, y con ello se crean principalmente 3 categorías de patrones: de arquitectura con los cuales se propone un esquema organizado de la estructura fundamental de un sistema; los de diseño, los cuales se enfocan a las estructuras de diseño de alto nivel y sus relaciones, es decir, aquellos que definen de forma explícita pero general el comportamiento del sistema; y los de idioma que son los que se enfocan en estructuras de bajo nivel y son dependientes del lenguaje de programación y el entorno en donde será ubicado el sistema.

Philippe Kruchten, quién desarrollo en 1995 el modelo de la arquitectura de 4+1 vista define

la arquitectura de software como:

“Una herramienta que es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño (requerimientos funcionales) de un sistema; así como requerimientos no funcionales, tales como: la confiabilidad, escalabilidad, portabilidad, y disponibilidad”.

Respecto al modelo de vistas, así como en la arquitectura y en la ingeniería civil, es frecuente el uso de modelos, planos y maquetas para la visualización de las diferentes facetas del diseño de una casa o edificio, en la arquitectura de software se tienen diferentes vistas del diseño de un sistema. Por ejemplo en la construcción de una casa, la instalación hidráulica muestra cuantos elementos la componen y como se relacionan entre ellos, pero ¿a quién le interesará principalmente esta “vista”?, la respuesta sería al arquitecto y los plomeros, en cuanto a la instalación, sin embargo al usuario, es decir los futuros habitantes de la casa, les importará su funcionalidad. Lo mismo sucede en el desarrollo de un sistema de software, se tiene un conjunto de personas cada una de ellas se enfocará a una “vista” en particular del sistema de software.

Para Philippe Kruchten un sistema puede ser visto de 5 perspectivas, las cuales él llama 4+1 Vistas, ya que para el una de dichas vistas es en la que se centran las otra cuatro, la figura 1 muestra la representación visual de este modelo.

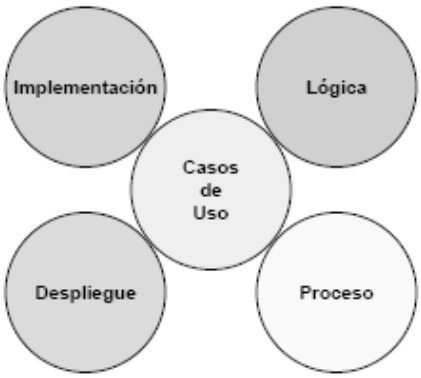


Figura 1. Representación Visual del modelo de 4+1 Vistas.

La del centro la **Vista de los Casos de Uso**, en la cual se centran las demás, es la perspectiva que tendrán o tienen los usuarios del sistema, describe los requerimientos del sistema, así como, la funcionalidad que éste debería producir según los usuarios que se pueden llamar actores externos. Un actor es aquella entidad que inicia una secuencia o sucesión de eventos. Un caso de uso se puede definir como una secuencia de acciones que realiza el sistema y que son generadas por un actor cuando éste interactúa con el sistema, es decir, es una narración que sirve para ejemplificar la utilización de un sistema, en una de las múltiples formas en que el sistema será utilizado.

La **Vista Lógica** describe como la funcionalidad del sistema es provista. Esta vista está destinada a los desarrolladores, mira dentro del sistema, describiendo ambos aspectos de un sistema estático y dinámico. En el estático describe los objetos, clases y relaciones. En el dinámico describe como estos objetos están interactuando y cooperando entre ellos al enviar mensajes en respuesta a eventos externos o internos.

La **Vista de Proceso** también llamada vista de concurrencia está destinada a los desarrolladores e integradores, describe la división de los sistemas en procesos y procesadores. Dicha división permite un uso eficiente de los recursos, ejecución paralela y el manejo de eventos asíncronos, así como, la disponibilidad, la tolerancia a fallos y la integridad, siendo éstos algunos requerimientos no funcionales.

La **Vista de Despliegue o Vista Física**, describe el despliegue físico de los sistemas, muestra la configuración de las unidades de hardware y software, con las que el sistema funciona en tiempo de ejecución, es decir se centra en los requerimientos no funcionales. Desarrolladores, integradores y probadores son los que hacen uso de esta vista.

La **Vista de Implementación o Vista de Desarrollo** describe la implementación de los módulos y sus dependencias, los módulos se verifican mutuamente para asegurar que todos los requerimientos se encuentran en el código. Dichos módulos son paquetes de software como librerías de programa, subsistemas, componentes, etc., que se organizan en capas por lo general jerárquicas, con la intención de dar una gestión adecuada al desarrollo del software teniendo un balance en el reparto del requisitos al formar equipos de trabajo con una adecuada planificación y una correcta evaluación de los costos, involucrados en el desarrollo, así como el poder monitorear el progreso del proyecto, anticipar su reutilización, su portabilidad, seguridad y las limitaciones que tendrá debido a las herramientas y el lenguaje de programación utilizado en el desarrollo impuestas por su entorno. Esta

vista está destinada principalmente a los desarrolladores.

Por otra parte es importante aclarar que dependerá del esquema formal de desarrollo las vistas que se considerarán, aún así en cualquiera de ellos, por lo regular siempre se tienen en cuenta 3 vistas o modelos para el desarrollo:

La vista que define las partes o componentes que conforman el sistema, llamada **vista estática**.

La vista que hace una descripción de la funcionabilidad de cada componente, conocida como **vista funcional**.

La vista que da un seguimiento a lo largo del tiempo de cómo cada componente actúa e interactúa con las demás, llamada **vista dinámica**.

También cabe señalar que al igual que en el campo de la arquitectura existen diferentes tipos de construcciones, como lo son las casas diferentes ellas de los edificios; dentro del campo del software existen a su vez tipos de arquitecturas de acuerdo al estilo de desarrollo de software el cual a su vez depende del contexto en que una aplicación de software va a ser empleado. Una clasificación de estilos y de sus arquitecturas podría ser la siguiente:

Estilo arquitectónico	Modelo de desarrollo o arquitectura empleada	Descripción
Flujos de Datos	Tuberías y filtros	Las componentes de software actúan como filtros para los datos, transformando éstos en salidas, que a su vez servirán de entradas a otra u otras componentes, para lo cual dichas componentes se comunican entre sí, a través de tuberías, y por lo tanto cada componente posee interfaces de entrada y salida. Este tipo de estilo forza a un procesamiento de tipo secuencial, es simple. Este estilo se basa los sistemas operativos UNIX , sin embargo también es utilizado en los compiladores los más antiguos sobre todo(ver figura 2), también se usa en el tratamiento de documentos XML , en algunos mecanismos de determinados motores de servidores de bases de datos, entre otros.

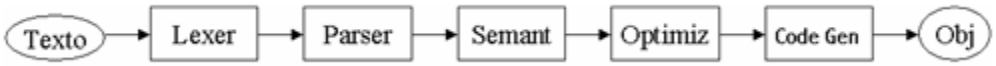


Figura 2 Ejemplo de Tuberías y Filtros

Centrado en Datos	Arquitecturas de Pizarra o Repositorio	Se centran en las estructuras de datos principalmente en su acceso y actualización. Ejemplos de este estilo son las bases de datos, aquellas arquitecturas basadas en hipertexto, los repositorios y las arquitecturas de pizarra. Los sistemas basados en repositorio y pizarra (ver figura 3), constan de dos tipos de componentes una central (pizarra o repositorio) en la cual se actualiza el estado de un problema y por lo tanto se ve constantemente reflejado los cambios en el sistema, provocados por el otro tipo de componente, un conjunto de fuentes independientes que en conjunto tratan de resolver un problema. y cuya interacción se da a través del repositorio o de la pizarra. También se cuenta con una estrategia, encargada de regular el orden en que operan las fuentes.

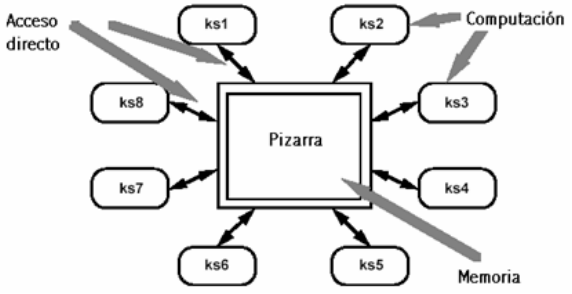


Figura 3. Sistemas basados en pizarra

Llamada y retorno	Dentro de este estilo existen varios modelos de desarrollo, los cuales son descritos uno a uno a continuación.	Se distinguen principalmente por su característica de producir una reacción a alguna acción efectuada por un actor y por lo general son utilizados para sistemas que involucran a más de un tipo de usuario, ya sean estos sistemas grandes o a gran escala
	- Modelo-Vista-Controlador	Este modelo Adoptado en algún momento por Sun para el desarrollo de aplicaciones en Java, es muy similar al modelo de 3 capas y consiste en separar principalmente la programación de la interfaz de usuario de

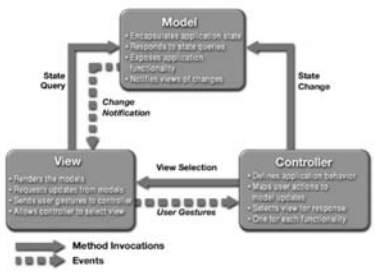


Figura 4. Modelo MVC

la referente a la que realiza el manejo de las bases de datos. Consta de 3 unidades de Software:

- Vista. En la cual se establece la interacción con el usuario.
-Controlador. Es el encargado de manejar las reglas a los eventos solicitados por el usuario, para obtener o realizar cambios a los datos.
- Modelo. En esta unidad se encuentran representados los datos que contiene el sistema.

- Arquitecturas en Capas. Consta a su veces de subclasificaciones que se describen a continuación

Una de las arquitecturas más utilizadas, trata de establecer una separación entre las aplicaciones monousuario, cliente servidor y distribuidas.

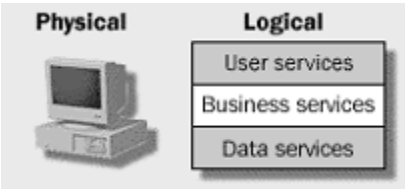


Figura 5. Arquitectura de Capa monolítica

La arquitectura de capa monolítica, a sido utilizada en aplicaciones monousuario, que no requieren comunicación con otros equipos. Y en su programación no existen unidades separadas que distingan el manejo de los datos, su validación y respuesta a eventos del usuario.

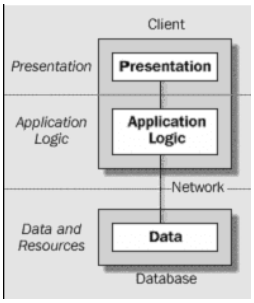


Figura 6. Arquitectura de dos capas modelo cliente-servidor

Utilizada aún antes de la Internet, la arquitectura cliente-servidor por lo regular maneja separadamente la vista del usuario, de la base de datos la cual se encuentra en un equipo distinto del utilizado por él o los usuarios. Estos usuarios pueden tener diferentes “vistas”, de la aplicación en distintos equipos al mismo instante de tiempo.

Esta arquitectura, que tuvo algo de

Figura 7. Arquitectura de tres capas

reticencia a ser utilizada por los programadores al principio, fue convenciéndoles finalmente, ya que a pesar de la sensación de que las líneas de código se triplican, la facilidad que provee en el seguimiento y mantenimiento del código, tiene sus grandes recompensas. Al igual que la arquitectura MVC, consta de 3 capas.

Presentación, Provee la interfaz de Usuario.

Negocio. Encargada de manejar las reglas de negocio, como son la validación de los datos y el manejo y control de eventos.

Datos. Encargada de manipular directamente o a través de manejadores de bases de datos, la información empleada en la aplicación..

Figura 8. Arquitectura en n-capas

La intención principal de este modelo, es la protección y seguridad de la información., para lo cual los servidores de aplicación se organizan como conjuntos de muchos servidores sencillos n-niveles. De tal manera que los clientes puedan así combinar servicios de diferentes servidores y cada servidor pueda implementar su funcionalidad en base a otros servidores.

- Arquitecturas Orientadas a Objetos

Esta arquitectura se basa en la característica de los objetos referente a su interoperabilidad, así como a la abstracción, Con respecto al polimorfismo y la herencia, recientemente se tiene la tendencia a conformar componentes de caja negra que proporcionen servicios a otros componentes también de caja negra, con lo cual dio paso a la siguiente arquitectura

Arquitecturas Basadas en Componentes	Uno de los modelos más utilizados actualmente debido al incremento del desarrollo de aplicaciones para Internet u otros sistemas distribuidos basados en CORBA, y otros desarrollos como OLE, COM, DCOM y más recientemente .NET, así como los hechos por RMI y J2EE.

Existen otra serie de estilos como los son los de Código Móvil -utilizado principalmente en la creación de máquinas virtuales- y el estilo Peer- to Peer, de la cual la que más se distingue es la Arquitectura Orientada a Servicios, comúnmente llamada por sus siglas en inglés SOA, en la cual varias empresas han enfocado sus esfuerzos como IBM; Microsoft. Dirigida principalmente a sistemas de gran escala como su nombre lo dice la arquitectura SOA está enfocada hacia los servicios que brinda, facilitando a que éstos puedan ser utilizados no importando las técnicas ni el lenguaje de programación empleado, lo realmente importante en ella son las interfaces. Esta arquitectura similar a CORBA, ya que trata de ser una plataforma con su conjunto de reglas y especificaciones a seguir, es más sencilla de implementar. Los servicios interactúan en una asociación débilmente acoplada. Y cada interacción es independiente de cualquier otra. En SOA un servicio no es más que una unidad de trabajo que actúa en nombre de alguna entidad de cómputo, sea esta un usuario humano o algún otro programa o entidad de software, SOA será encargado de establecer la manera en que dos entidades se comunicarán o interactuarán entre sí, ya que como se menciono anteriormente estas pueden ser creadas en plataformas y lenguajes diferentes.

1.2 Desarrollo Orientado a Objetos.

Para definir que es el desarrollo orientado objetos; se requiere antes que nada definir o recordar en caso de que el lector ya haya tenido algún conocimiento previo, algunos de los conceptos referentes a los objetos

Objeto.

Un objeto según la Real Academia Española es “Todo lo que puede ser materia de conocimiento o sensibilidad de parte del sujeto, incluso este mismo” En el mundo de la informática lo que nos

interesa es precisamente ese conocimiento que se puede obtener de “algo” o “alguien”, es por ello que para nosotros dicho objeto estará divido en dos partes. La primera son los atributos o características, los cuales desde el punto de vista de un lenguaje de programación se llaman datos o estructuras de datos. La segunda parte es el comportamiento o la forma en que realiza ciertas acciones las cuales pueden ser convertidas en rutinas de programación y que se conocen como métodos.

Abstracción.

A pesar que en el párrafo anterior al definir un objeto se hizo referencia a como se vería un objeto en términos de programación, en el Desarrollo Orientado a Objetos y principalmente en la etapas de conceptualización y análisis, en lo que menos deben centrarse las personas encargadas de ejecutar estas etapas es en el lenguaje de programación que va a ser utilizado para el desarrollo, ellas deben hacer uso de la abstracción. La abstracción es la acción de centrarse en lo que es y en lo que hace un objeto, no la forma en la que éste será implementado en un lenguaje de programación. Por ejemplo, en un sistema que se quisiera desarrollar para controlar las facturas de venta, las facturas son objetos que obviamente estarán presentes en dicho sistema. Haciendo uso de la abstracción se deben hacer entonces preguntas como: ¿Qué es una factura?,¿Cuál es su principal propósito?¿Está conformada a su vez, por algún o algunos otros objetos?¿Cómo son creadas las facturas?¿Pueden ser modificadas o eliminadas? ¿Quién o quienes tendrían derechos de acceso y actualización a ellas? ¿Cómo se relacionan con otros objetos?, etc.,

Clase.

Nuevamente recurriendo al diccionario de la Real Academia española una clase se define como “Orden en que, con arreglo a determinadas condiciones o calidades, se consideran comprendidas diferentes personas o cosas”, en el mundo de la informática y en particular en la programación orientada a objetos una clase no es otra cosa que una definición de atributos y comportamientos que distinguen a un objeto de un tipo en particular de otro tipo de objeto. Haciendo una analogía con respecto a la definición de nombre común, el cual se aplica a personas, animales o cosas que pertenecen a una misma clase, especie o familia, significando su naturaleza o sus cualidades, una clase es algo similar así cuando le damos un nombre por ejemplo Factura, en éste agrupamos las cualidades, naturaleza y comportamiento que tendrán todas las facturas.

Instancia.

Por su parte una instancia es un ente particular de una clase, es decir toma de su plantilla que define formas y comportamientos, dichos atributos y comportamientos para ser un ente dinámico que llena los atributos con valores específicos con lo cual se distinguirá de las demás instancias de su clase. Podría decirse que una instancia es similar a un nombre propio, él que sin tener rasgos semánticos inherentes, se aplica a seres animados o inanimados para designarlos, pero además su comportamiento dinámico le permite realizar las acciones o ejecutar las rutinas que fueron definidas en su clase.

Superclase.

Es una conceptualización de varias clases de una forma muy general. Por lo general son como patrones, que sirven de base a las clases que toman de éstas características y comportamiento

Las figuras 9, 10 y 11 tratan de realzar las diferencias entre superclase, clases e instancias.

Cabe señalar que en la jerga de orientado a objetos un objeto puede ser una instancia y viceversa, sin embargo aquí se trataron estos términos de manera independiente para enfocarse en la abstracción al hablar de objetos más que en la programación de ellos.

Instancia u objeto

PELUDO
-PELO : PELUDO = LARGO -PESO : PELUDO = 2 KGS
+DORMIR() : PELUDO +CAZAR() : PELUDO +JUGAR() : PELUDO

Figura 9 Ejemplo de una instancia haciendo uso de su diagrama.

Clase

GATO
-PELO : G -PESO : G
+DORMIR () : G +CAZAR () : G +JUGAR () : G +TOMAN LECHE () : G

Figura 10. Ejemplo de una clase y su diagrama

Superclase

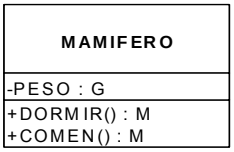


Figura 11. Ejemplo de una superclase y su diagrama

Importancia del uso de objetos.

Considérese una ventana de Windows como objeto:

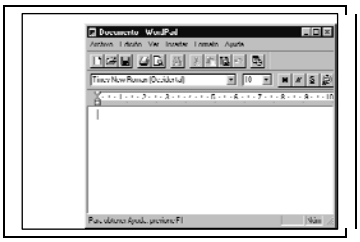


Figura 12 Ejemplo de una ventana vista como objeto

Algunos de sus atributos podrían ser: ancho, largo; distancias con respecto a la esquina superior izquierda de la pantalla: izquierda, alto; color del fondo, estilo del borde, nombre; estilo, color y tamaño de la fuente que en ellas aparecen.

Algunas operaciones que se realizan serían: moverla de lugar, redibujarla, minimizarla, esconderla, maximizarla, mostrarla, en respuesta a mensajes que le son enviados por otros objetos como el ratón cuando se presiona el botón izquierdo al estar el puntero del ratón en la zona de la ventana.

Figura 13. Ejemplo de eventos que puede recibir un objeto ventana.

Si se programa este objeto como una clase garantizando que las respuestas a los mensajes funcionen de la manera esperada sin errores, esta clase ventana podrá ser *reutilizada* por otros sujetos para crear *instancias de ella*, es decir, objetos a los cuales se les podrá otorgar atributos

propios, como por ejemplo color del fondo:= “BLANCO”

Reutilización es la idea atrás de la orientación objetos con ella se obtienen los siguientes beneficios:

1) Ahorra tiempo, al no tener que programar una y otra vez las mismas componentes comunes que son usadas en casi todos los programas

2) Simplificación, al saber como funciona una clase, puede el diseñador y/o programador enfocar sus esfuerzos en problemas más complejos o para los cuales no existen clases.

3) Mejor enfoque, al abstraer o modelar la realidad como un conjunto de entes que responde a eventos, se tiene un diseño más apegado a ésta y por lo tanto más sólido con menor posibilidad de errores.

Pero no sólo la reutilización es lo importante de la orientación a objetos, otro factor que es decisivo para su utilización es **la cooperación** entre objetos. Los objetos son creados con este propósito, ya que bien dice el dicho que “la unión hace la fuerza” y se tiene de esta manera un conjunto de objetos trabajando todos ellos en conjunto y armonía para lograr los propósitos para los cuales fueron diseñados. Es pues que la sinergia, la cual es: la acción de dos o más causas cuyo efecto es superior a la suma de los efectos individuales, que queda plenamente demostrada en el uso de los objetos al desarrollar sistemas informáticos.

Herencia

Un mecanismo muy importante dentro del mundo orientado a objetos, es la herencia, ya que es a partir de este mecanismo también se puede lograr la reutilización, al aprovechar parte de los atributos y sobre todo los métodos de clases ya definidas, dando como resultado una clase derivada a la cual se le pueden añadir atributos y métodos que hacen que la clase derivada se use en una forma más detallada o ad hoc a la aplicación donde se utiliza. También es a través de la herencia que las clases pueden ser organizadas de una manera jerárquica.

Cuando se habla de herencia se pueden establecer dos tipos de relaciones entre clases una de ellas es la generalización y la otra es la especialización.

- **Generalización** .-Es usada al crear superclases que encapsulan la estructura y el comportamiento común a varias clases, consiste en identificar aspectos comunes entre conceptos y en definir las relaciones entre el ***supertipo*** (concepto general) y el ***subtipo*** (concepto especializado).
- Una clase puede no provenir de ninguna superclase, por lo que se le conocerá como *clase raíz o base.* Una clase de la cual no se van a generar instancias se conoce como *clase abstracta*.
- La manera gráfica de representar la generalización es la siguiente:

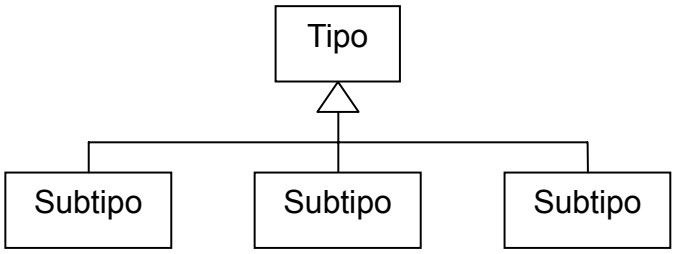


Figura 14. Representación gráfica de la generalización

-
-
- **Especialización**.- Un *subtipo* es la *especialización* de un tipo, es decir, es la relación en la cual las clases están definidas como casos especiales de cada una de ellas, un objeto es un subtipo de su padre
-
-

En resumen la herencia es el mecanismo para compartir automáticamente comportamientos (que llamaremos operaciones o métodos) y características (que llamaremos atributos o datos). La figura 15 muestra de manera gráfica estos 3 conceptos.

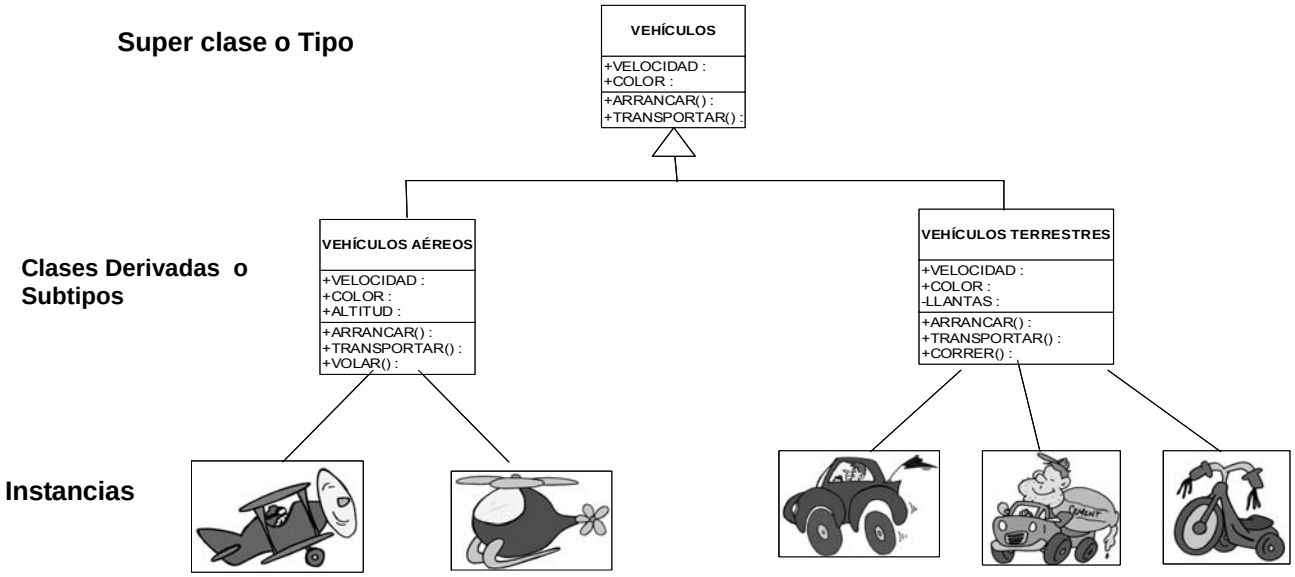


Figura 15. Ejemplo gráfico de los conceptos de herencia, generalización, especialización e instancias.

Estereotipos.

Los estereotipos son utilizados para dar mayor claridad a lo que se quiere representar por ejemplo si tenemos una clase RobotHw y queremos indicar que esta clase va a realizar un control o que pertenece al estereotipo controlador se puede entonces incluir en la representación gráfica de kla clase el nombre del estereotipo encerrado entre los símbolos (marcados entre comillas) “<<” y “>>”, así quedaría <<controller>>

Mensaje

Para que los objetos puedan cooperar entre sí, requieren poderse comunicar, esto lo logran a través del uso de mensajes. Por lo general un objeto envía un mensaje cuando requiere que el objeto receptor del mensaje ejecute una acción determinada, para la cuál puede requerir que el objeto emisor del mensaje envíe alguna información lo cual puede hacerse a través de parámetros. Al terminar la acción solicitada el objeto receptor del mensaje puede retornar alguna información al objeto emisor o quizás no sea necesario. La comunicación por lo tanto puede ser asíncrona si es que el objeto emisor del mensaje puede continuar con su actividad sin requerir una respuesta del objeto receptor para poder continuar, o puede ser la comunicación de tipo síncrona cuando el objeto emisor del mensaje requerirá esperar ya sea una información del objeto receptor del mensaje o una confirmación de que la tarea o acción solicitada ha sido concluida.

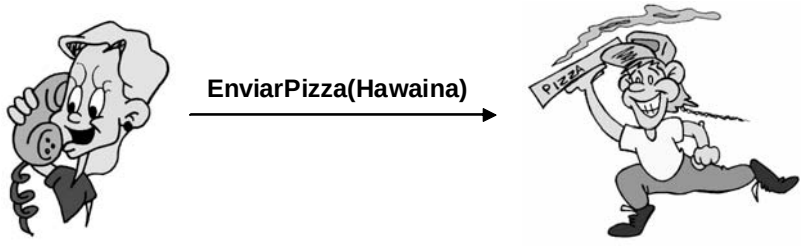


Figura 16.Ejemplo de mensaje entre objetos

Polimorfismo.

Es el fenómeno por medio del cual un mismo mensaje puede dar origen a que objetos de distintas clases que lo reciben realicen acciones que pueden ser completamente diferentes.

Cabe hacer la aclaración -ya que por experiencia se ha observado que algunas personas tienden a confundir este concepto con el de métodos recargados de la programación orientada a objetos-, en la recarga de métodos una misma clase puede tener operaciones o métodos con el mismo nombre, diferenciándose entre sí, en el tipo y la cantidad de parámetros que manejan.

Por su parte el polimorfismo se refiere a operaciones o métodos que tienen el mismo nombre pero se encuentran en clases distintas, aunque estas clases hayan derivado de una clase abstracta los métodos son implementados de manera diferente en cada clase derivada.

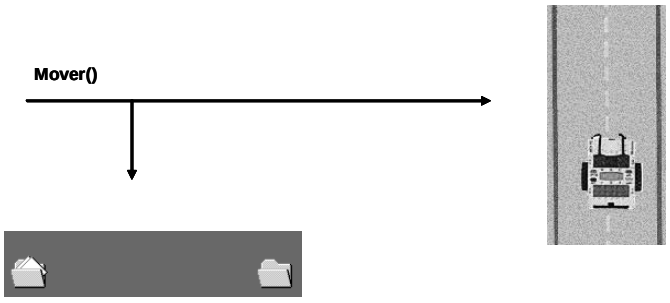


Figura 18. Ejemplo de polimorfismo.

Asociación

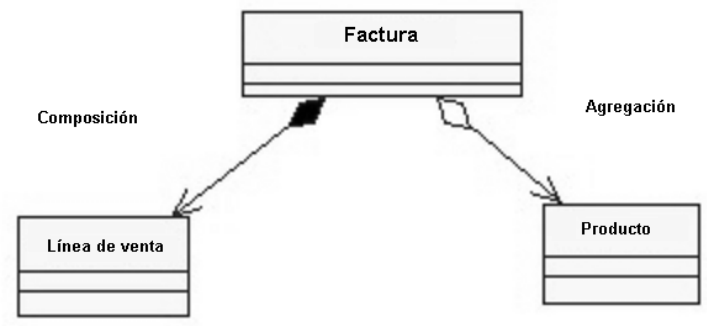
Una manera en que los objetos pueden cooperar los unos con los otros es a través de establecer relaciones entre ellos, ya sea que pertenezcan a su misma clase o a otra, para lo cual hacen uso de

asociaciones que son enlaces físicos o conceptuales entre instancias.

Existen dos tipos especiales de asociación que son utilizadas cuando en una aplicación se requiere el uso de objetos compuestos, estas asociaciones se llaman agregación y composición.

Figura 18. Tipos especiales de asociación: agregación y composición.

- Agregación.- Es un tipo especial de asociación en donde un objeto se relaciona con otro al formar parte de éste, es decir, es una relación todo-parte. Así un objeto que es *agregado* es un objeto base que incluye a otros objetos para su funcionamiento. La **agregación** está relacionada con la **referencia** a un objeto. Por ejemplo en una factura está hace referencia a los productos, pero la destrucción del objeto factura no lleva consigo la destrucción de los productos.
-
- Composición. En este tipo de asociación muy parecido a la agregación ya que también se trata de una relación todo-parte, sólo que la diferencia radica en que en la composición si se retiran los objeto de ella, el todo deja de tener significado, lo mismo sucede a los objeto retirados, si se destruye el todo, Por ejemplo en una factura si se retiran las líneas de venta la factura no tiene sentido lo mismo sucede con destruir la factura se destruyen las líneas de venta.La **composición** está relacionada con el **valor** del objeto, es decir el objeto que lo contiene se encarga de su ciclo de vida (inicialización, mantenimiento y anulación)
-
-



-
- Figura 19. Ejemplo de Composición vs. Agregación

- **Rol**.
- Cada objeto dentro de una asociación va a representar un papel o rol., es común que el nombre de dicho rol sea el de la clase cuando en existe una asociación entre dos clases. Sin embargo pudiese ser que el nombre sea demasiado genérico, por ejemplo una clase que se llama Empleado puede a su vez tener diferentes asociaciones con otras clases, considérese que esta clase puede jugar tanto el rol de Jefe y como el de Subordinado también, ya que es común que un jefe a su vez tenga un superior, el cuando jugará cada rol dependerá de cómo se asocie con otra Clase.
- Un nombre de un rol nunca deberá ser un atributo de la clase, esto se debe a razones de implementación.
-
- **Grados de una asociación**.
- Una asociación entre clases puede ser de más de dos clases, si es de dos se llama asociación binaria, si es de tres ternaria, si n se llamará asociación n-aria
-

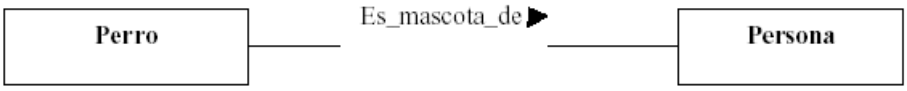


Figura 20 Ejemplo del grado de asociación binario

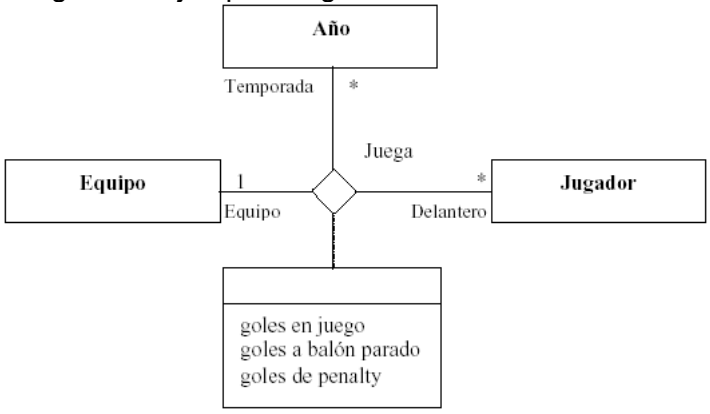


Figura 21 Ejemplo del grado de asociación ternario

-

- **Tipos de asociaciones**
-
- Reflexiva. Las clases no sólo requieren asociaciones con otras clase, en ocasiones es necesario que instancias de una misma clase se asocien entre ellas, a esto se le conoce como asociación reflexiva.
-

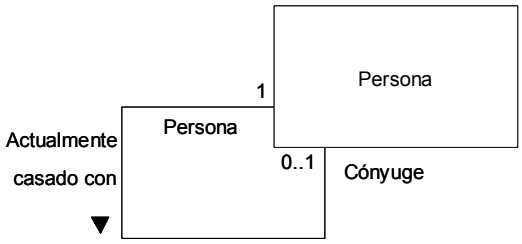
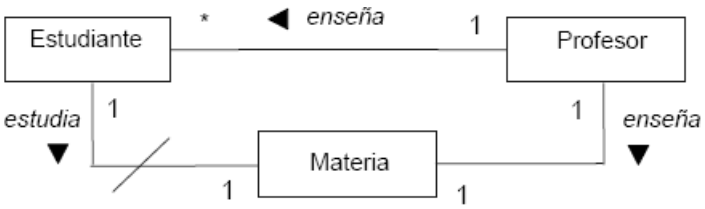


Figura 22 Ejemplo de una asociación reflexiva

- Derivada.- Una asociación es derivada cuando es redundante y por lo general se utilizan para facilitar la búsqueda de información. Es señalada siempre con "/".
-



-

Figura 23. Ejemplo de una asociación derivada

- **Multiplicidad.**- Un importante aspecto de la asociación. Especifica el número de objetos de una clase que están asociados con único objeto de otra clase.
-
-

Figura 24. Ejemplo de Multiplicidad

La multiplicidad puede ser representada como un rango de enteros no negativos, donde un asterisco "*" representará un valor infinito en el lado máximo del rango.

En la figura 25 se muestra algunos ejemplos de multiplicidad.

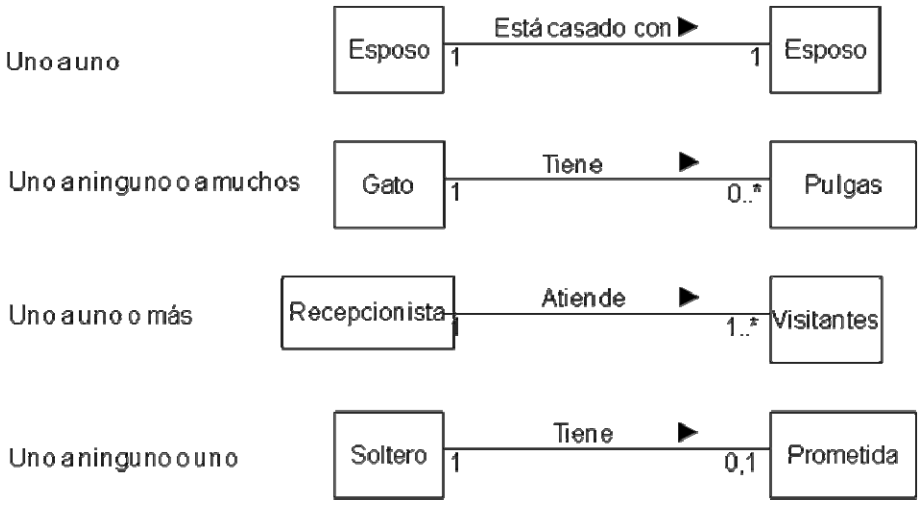


Figura 25 Ejemplos de tipos de multiplicidad.

Dominio.

El dominio puede definirse como un ámbito -el cuál puede ser real o imaginario- en de donde se utilizará un sistema o se desarrolla alguna actividad.
Es muy común el uso del término dominio del problema para referirse al alcance ideal configurado por las cuestiones y los problemas de una o varias actividades o disciplinas relacionadas entre sí que requieren ser sistematizadas.

Escenarios.

De acuerdo al diccionario de la Real Academia Española un escenario es “El Conjunto de circunstancias que rodean a una persona o un suceso”, empleando y aplicando esta definición a ingeniería de software se puede a su vez definir que a un escenario como el conjunto de eventos que describen como un usuario interactúa con el sistema al realizar haciendo uso de él una actividad muy en particular como por ejemplo cuando el usuario hace el registro de una factura al realizar una de venta de contado.

Análisis (antes del enfoque orientado a objetos).

Anteriormente al desarrollo orientado a objetos, las aplicaciones de software eran desarrolladas utilizando un enfoque conocido como “estructurado” el cual se centraba en los procesos que eran llevados a cabo en el ámbito en el cual se requería dicha aplicación.

Por lo tanto bajo este enfoque estructurado, el análisis puede ser definido como una investigación que se centra en el dominio del problema no en definir una solución, así para crear una aplicación de software hay que describir el problema y las necesidades o requerimientos.

Diseño (antes del enfoque orientado a objetos).

Por su parte el diseño en un enfoque estructurado es el hallar una solución lógica que satisfaga plenamente los requerimientos y las restricciones a través de descripciones detalladas y de alto nivel de. Por último los diseños se implementan en software y en hardware.

Análisis Orientado a Objetos

Consiste en situar el dominio de un problema dentro de la perspectiva de los objetos como: personas, cosas, conceptos, eventos etc., y las relaciones presentes entre ellos.

En esta etapa es importante llevar a cabo la realización de un modelo visual del sistema, lo que implica un conjunto de diagramas.

Actualmente los diagramas que más se utilizan mundialmente para realizar dicho modelo visual son los contenidos en el lenguaje unificado de modelado conocido por sus siglas en inglés UML.

Diseño Orientado a Objetos.

Por su parte el diseño orientado a objetos es una etapa en la cual se procura definir los objetos, pero ya no desde una perspectiva conceptual, como en el caso del análisis, sino como objetos lógicos de software definiendo sus atributos y operaciones. Dichos objetos finalmente serán implementados en un lenguaje de programación orientado a objetos.

Por último es importante hacer notar que existen dos tecnologías de objetos básicas dentro del desarrollo orientado a objetos:

1)- **Tecnología Orientada a Objetos** (Object Oriented Technology, OOT) que incluye:

• Programación orientada a objetos.

• Análisis, diseño y modelado orientado a objetos.

2)- **Tecnología Basada en Objetos** (Object Based Technology, OBT), utiliza mecanismos cliente/servidor, incluyendo:

• Programación basada en componentes.

• Análisis, diseño y modelado de componentes (objetos) distribuidos.

• Computo distribuido y concurrente.

Su principal diferencia es que los métodos orientados a objetos utilizan herencia, y son presentados en especificaciones de programas o códigos. Por otra parte, los métodos basados en objetos utilizan herencia de interfaces y son presentados en código binario ejecutable o código basado en DLL.

Ciclo de Desarrollo.

El ciclo de desarrollo no es otra cosa más que el ciclo de vida que tendrá el desarrollo de un proyecto de software y sirve como marco de referencia para indicar que productos o subproductos se deberán obtener y cuando, en cada una de las etapas llamadas fases, a lo largo de dicho desarrollo.

Dado que existen distintos productos de software la manera de agrupar las actividades, los objetivos de cada fase, los recursos empleados, los tipos de subproductos intermedios generados, etc. pueden ser muy diferentes dependiendo del tipo de producto, así como de las tecnologías empleadas. Por otra parte la complejidad de las relaciones entre las distintas actividades crece exponencialmente con el tamaño del producto o sistema de software a desarrollar, es por ello que la división de los proyectos en fases sucesivas es un paso fundamental en la reducción de la complejidad, por eso se elegirán las partes cuyas relaciones entre sí sean lo más simples posibles.

El tener un ciclo de vida definido facilitará enormemente el **control de tiempos** necesario para aplicar recursos de todo tipo (personal, equipos, suministros, etc.) al proyecto. Así como el **control de calidad** también será más sencillo si la separación entre fases se hace corresponder con puntos en los que la calidad deba verificarse (mediante comprobaciones sobre los productos parciales obtenidos).

Las fases en un ciclo de vida se suceden una a la otra, pero no implica que no puedan tener ciclos de

retroalimentación. Cada fase está a su vez compuesta por actividades que a su vez son planificadas.

En la Figura 26 se muestran lo elementos que conforman un ciclo de vida: Las fases y los subproductos que son los resultados que pueden ser evaluados en cada fase .

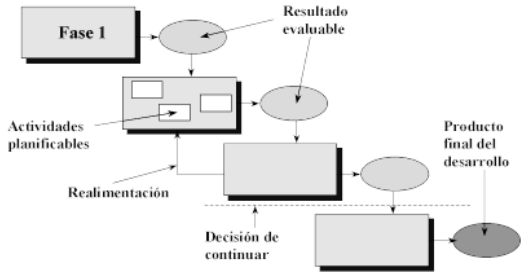


Figura 26 Representación gráfica de un ciclo de desarrollo.

Existen varios tipos de ciclo de vida, los cuales a su vez constan de distintas fases. Sin embargo por lo regular en los distintos tipos de ciclo se encuentran presentes las siguientes fases

Definición o Determinación de requisitos. En la cual se realiza un estudio de viabilidad del proyecto antes de continuar con él, y se hace un estudio detallado de los requisitos que la aplicación deberá tener para asegurarse que requisitos son alcanzables y cuáles no lo son para llegar a establecer acuerdos con los usuarios y llevar a cabo una planificación adecuada y en detalle.

Diseño. En esta fase se deberá especificar de una manera completa la arquitectura global tanto de de las especificaciones de software como de hardware que cumplan en forma óptima en cuanto a tiempo, costo y calidad con cada uno de los requerimientos acordados en la fase anterior, haciendo una correcta asignación de los recursos, así como una validación de las herramientas, técnicas y métodos empleados y si se requiere llevar a cabo un replanteamiento de los requisitos para ajustar las especificaciones de la aplicación o producto de software

Construcción. La cual involucra la codificación e integración de cada una de los elementos de software, verificando y validando cada una de las estructuras de control y de datos, así como de las interfaces de relación, dimensión y algoritmos claves de conforman cada componente de la aplicación, realizando los ajustes necesarios para corregir posibles errores, inconsistencias u omisiones.

Implementación. Poner en funcionamiento el sistema, producto o aplicación de software desarrollado, realizando si se requiere la conversión de programas y datos y llevando a cabo las

instalaciones de componentes tanto de software como de hardware requeridas para el funcionamiento del sistema e implantar la capacitación de aquellos potenciales usuarios del sistema que a su vez sean factibles de convertirse en instructores del uso del sistema o que serán los usuarios finales.

Operación y mantenimiento. En esta fase se debe fijar sólidamente el uso del sistema cumpla con el objetivo para lo que fue realizado y de ser necesario realizar el mantenimiento requerido para ajustar el producto a las nuevas metas que surjan a lo largo del uso del sistema o aplicación de software.

Como ya se dijo anteriormente la razón de tener distintos tipos de ciclo de vida, es que las aplicaciones de software difieren entre sí debido a las distintas características intrínsecas de cada aplicación: tamaño, dominio etc., las cuales marcan las diferencias entre los ciclo de vida siendo las principales, el alcance que tendrá un proyecto, que puede ir desde sólo el estudio de factibilidad hasta su desarrollo y liberación completa; los contenidos o características de cada fase que depende enteramente del tipo de aplicación o producto de software a desarrollar,(se desarrollará un juego de computadora o un aplicación comercial?) y por última la estructura del ciclo de vida.

Es dentro de la estructura del ciclo de vida que se pueden definir algunos tipos.

En cascada.

Para tener adecuadas decisiones al realizar la planificación esta estructura descompone las fases de tal manera que se puede saber con antelación lo que en cada fase ocurrirá antes de comenzar con ella.

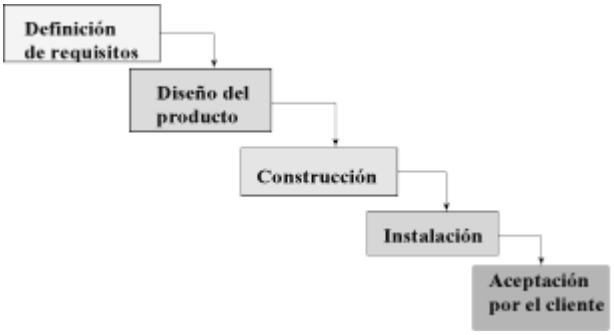


Figura 27. Ciclo en cascada

Por prototipo.

Este tipo de ciclo de vida es idóneo para utilizarlo en aquellas circunstancias en que a los usuarios finales les es muy difícil establecer todas las especificaciones que debería tener un producto de

software al final, es por ello que la creación de un prototipo difiere en el de cascada, ya que en el prototipo las fases de definición, diseño, y construcción pueden ser repetidas hasta obtener especificaciones más completas.

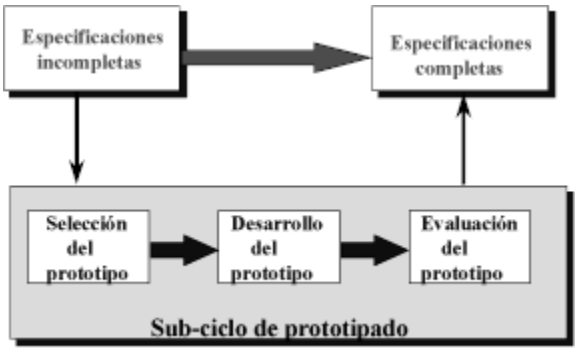


Figura 28. Ciclo en prototipo

Por evolución.

Este ciclo permite que se puedan hacer sistemas dinámicos, en los que los requisitos cambian en el futuro, es decir, después de haber terminado el desarrollo. Con lo cual al cambiar los requisitos se puede realizar toda una nueva versión de manera más rápida que su antecesora.

Incremental.

Bajo ciertas circunstancias se requiere que se vaya haciendo una entrega parcial del sistema dada la premura o necesidad que se tenga del mismo. Esto puede llevarse a cabo al ir desarrollando el sistema de manera incremental, donde una parte del sistema puede ser suficiente para cumplir con ciertos requisitos, y dar tiempo a desarrollar las demás partes.

Espiral,

En este ciclo las fases de definición, donde se realiza la planificación y se definen los requisitos, la de diseño donde se toman las decisiones cuales son los elementos de software más adecuados para la solución deseada, la de construcción en donde se realiza la codificación e integración y la de implementación donde se evalúa al sistema forman una espiral, en la cual al realizar el primer recorrido de la espiral, ya se tendrá un prototipo, él cual puede ser evaluado para analizar e identificar si se cumplen con los requisitos establecidos y la gran ventaja de este modelo es que también en esta

etapa puede llevarse a cabo un análisis de riesgo, para determinar el grado de incertidumbre que proporciona el prototipo y así hacer los ajustes necesarios para disminuir dicha incertidumbre, los cuales serán llevados a cabo en el siguiente recorrido de la espiral hasta ejecutar las cuatro fases, y de esta forma consecutivamente hasta llegar al producto final deseado.

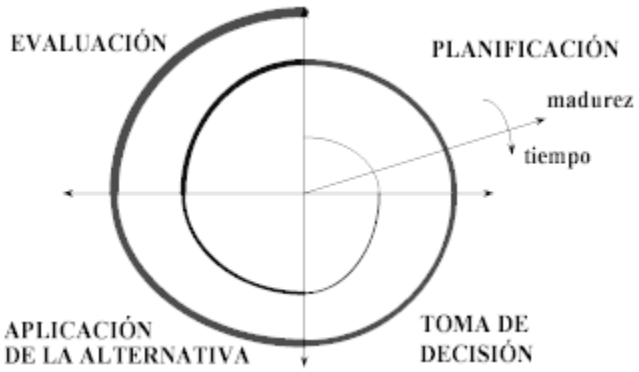


Figura 29. Ciclo en espiral

Metodologías de desarrollo

En el área del desarrollo de sistemas una metodología indicará como un producto o sistema debe ser obtenido así como sus subproductos en cada una de las fases de ciclo de vida seleccionado.
Dicha metodología estará compuesta de un conjunto de métodos (forma de hacer las cosas), técnicas (ejecución de un método) y herramientas (utilizadas en la técnica).

El objetivo principal de la metodología es obtener un producto de software de calidad. Pueden ser utilizadas en pequeños y grandes desarrollos y ser incluso metodologías que no sólo sean un conjunto de métodos para la elaboración de sistemas sino, de hecho muchas de ellas, abarcar la estimación de costos, la gestión y el control de tareas, entregas, recursos, tiempo y calidad.

Desde el surgimiento de la programación orientada a objetos han surgido muchas y diversas metodologías para el análisis y diseño orientado a objetos. Algunas de ellas se mencionan a continuación. Cada una de ellas representa diferentes orientaciones, ya que unas son orientadas a las responsabilidades, otras hacia los estados, otras hacia los datos y otras hacia los casos de uso.

Responsibility-Driven Design, conocido por sus siglas en inglés como RDD, fue concebido en 1990 por Rebecca Wirfs-Brock, quién es una de las pioneras en el desarrollo de metodologías orientadas objetos y que ha aplicado su diseño principalmente para sistemas complejos de software que cambia el pensar en objetos como datos + algoritmos a pensar más bien en los objetos como roles +

responsabilidades, de tal forma que los objetos son más que simples pedazos de lógica, convirtiéndose en proveedores de servicios, portadores de información, modeladores, coordinadores, controladores e interfaseadores, ya que cada uno cumple con un rol bien definido y tiene por lo tanto también bien definidas sus responsabilidades y así como las vías de comunicación con otros objetos para solicitar su colaboración en el cumplimiento de sus metas.

Object-Oriented Analysis and Design, conocido por sus siglas en inglés como OOA&D y desarrollado por Coad y Yourdon en 1991, consiste de cinco pasos, los cuales no necesariamente deben ser ejecutados de manera secuencial y son: Identificar las clases y objetos, Discernir las estructuras de generalización-especialización de las de Todo-Parte (agregación), Clasificar clases y objetos en grupos de acuerdo a sus perspectivas o propósitos dentro del uso del sistema, Definir aquellos atributos en las clases que distinguirán a cada instancia de esa clase, y por último Definir servicios que los objetos proveerán para el funcionamiento del sistema.

Object-Orientd System Analysis, conocido por sus siglas en inglés como OOSA, desarrollado por Sally Shlaer y Stephen J. Mellor en 1992, para esta metodología el análisis y el diseño pueden ser vistos como dos espacios, que pueden ser desarrollados de manera independiente y concurrentemente, considerando al diseño como recursivo. En el análisis el sistema se separa en un conjunto de dominios los cuales son definidos como un mundo separado, real, hipotético o abstracto, habitado por un conjunto de objetos que actúan de acuerdo a las reglas y políticas intrínsecas del dominio, así por ejemplo dentro de un sistema se podrían tener el dominio de la aplicación(reglas de negocio), el dominio de los servicios (por ejemplo interfaces), el dominio de la arquitectura(sistemas operativo, nodos etc.,) y el dominio de la implementación (lenguaje, manejadores de bases de datos etc.,) . Cada dominio con un comportamiento bien definido, característica que hace que dichos dominios puedan ser reutilizables en otros sistemas.

Object-Oriented Analysis & Design, Grady Booch desarrollado en 1994, consta de los siguientes pasos, definición del problema, definición de la posible solución, identificación de los objetos, identificación de los atributos de los objetos, identificación de los métodos asociados a cada objeto y por último determinación de las interfaces entre los objetos. Es un enfoque más bien de estructura estática del sistema.

Object Modeling Technique, u OMT por sus siglas en inglés, desarrollada por James Rumbaugh en 1991, una de las más utilizadas en la década de los 90, hace uso de 3 modelados: Modelado de los objetos., en donde se defines la estructura estática de los objetos su identidad, sus relaciones con otros objetos, sus atributos y operaciones, el Modelado dinámico, para Rumbaugh los objetos no son

sólo estructuras estáticas sino dinámica, ya que se comunican y colaboran entre sí por lo tanto temporización y secuencia de operaciones debe ser modelada; y por último el modelado funcional en el cual se describen las transformaciones de valores de datos, las funciones, correspondencias, restricciones y dependencias funcionales, que pueden ocurrir dentro del sistema

Object Oriented Software Engineering, desarrollada por Ivar Jacobson en 1992. Mientas que las anteriores metodologías se enfocan principalmente en los objetos del dominio de un sistema, para Jacobson lo más importante son los usuarios del mismo, es por ello que esta metodología es orientada a los casos de uso. Consta de realizar modelados a través de cinco fases de un ciclo de vida que son: Requerimientos, Análisis, Diseño, Implementación y Prueba.
Es en el modelado de los requerimientos, donde se plasma la idea de que el sistema debe ser construido desde el punto de vista del usuario, ya que es él, el que lo va a utilizar, en esta etapa, se describen los casos de uso, las interfaces de usuario y se modelan los objetos del sistema.

Algunas Metodologías recientes.

A pesar del gran esfuerzo en el desarrollo de estas metodologías algunas de ellas no resultaron ser muy prácticas ya sea por su rigidez, dependencia del dominio del problema, o amplitud, como el caso de OMT y OOSA de Shlaer y Mellor, es por ello que en algunas empresas como Hewlett-Packard surgieron metodologías híbridas como la de FUSION desarrollada por Derek Coleman en Inglaterra.

Otro de los grandes problemas en el uso de estas metodologías son las grandes diferencias entre la notación que utilizan para representar los conceptos utilizados en la orientación a objetos , es por ello que el grupo OMG (Object Management Group) un consorcio a nivel internacional que abarca a las más poderosas compañías de la tecnología de información Orientada a Objetos, propone e implementa por consenso especificaciones entorno al uso y desarrollo de la tecnología orientada a objetos estandarizo un lenguaje de modelado al que llamo UML (Unified Modeling Language), el cual es un esfuerzo para simplificar y consolidar el gran número de métodos de desarrollo orientado a objetos que habían surgido.

UML en sí fue derivado de tres metodologías: OOAD, de *Booch*, OMT de *Rumbaugh* y OOSE de Jacobson, ya que estos tres desarrolladores unificaron sus métodos cuando laboraron para la misma empresa en 1995, Rational Software Corporation.

La unión de estos tres personajes dio también surgimiento al Proceso Unificado Rational, RUP por sus siglas en inglés y de allí al UP proceso Unificado, el cual actualmente es una de las metodologías mundialmente más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, debiendo principalmente su éxito a que es un proceso de desarrollo configurable que se adapta a través de los proyectos no importando su tamaños o complejidad.

UP consta de cuatro fases:

Puesta en marcha. Conocida como Concepción o *Inicio*, en esta fase se lleva a cado la planificación, la identificación de los riesgos, identificación y elaboración de los casos de uso.

Definición, análisis y diseño. Conocida como *Elaboración* en esta fase se especifican en detalle la mayoría de los casos de uso y se diseña la arquitectura del sistema

Implementación. Llamada *Construcción*, en esta fase se construye el producto para que sea totalmente operativo y eficiente, a la vez que se elabora el manual de usuario.

Fin del proyecto y puesta en servicio: Esta es la última fase llamada *Transición* y es en esta fase que se realiza la construcción final del sistema, se capacita a usuarios y pueden surgir nuevos requisitos a ser analizados.

UP se caracteriza por centrase en la arquitectura ya que desde el inicio establece una arquitectura básica y realiza prototipos y para evaluarla y finalmente refinarla durante el transcurso del desarrollo del sistema.

UP también se caracteriza en ser iterativo e incremental puesto que cada fase es desarrollada mediante iteraciones del ciclo de vida que es en cascada a menor escala. El ciclo de vida es llevado bajo dos disciplinas: la de desarrollo que consta de las etapas de Modelado del negocio, modelado de los requisitos, análisis y diseño, implementación y pruebas y la disciplina de Soporte que consta de las etapas de Administración y configuración de las versiones surgidas en el desarrollo, administración del proyecto, administración el ámbito del sistema y por último distribución. Estas etapas son conocidas como flujos de trabajo. Cada iteración es importante ya que en ella son identificados y especificados los casos de uso revelantes y es creado un diseño basado en la arquitectura seleccionada y el diseño es implementado a través de componentes verificando que

estos satisfagan los casos de uso, una vez que la iteración cumple con todos y cad uno de sus objetivos se procede a la siguiente

La figura 30 muestra un diagrama de las fases y los flujos de trabajo del proceso.

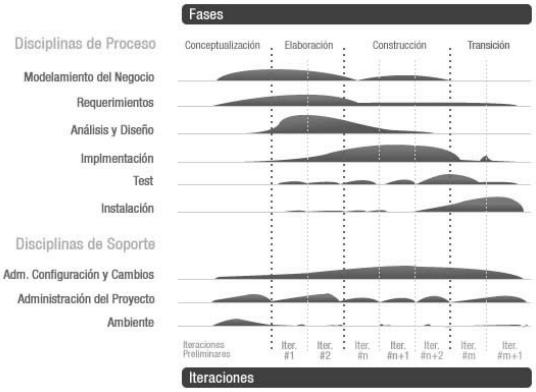


Figura 30. Fases y flujos de trabajo de UP

XP

Las metodologías ágiles a las cual pertenece e**X**treme **P**rogramming, enfrentan la necesidad de realizar desarrollos en un lapso de tiempo corto, y deben responden a los constantes cambios de requisitos por lo cual la interacción con el cliente es una de sus bases. XP es una de estas metodologías, su paradigma fue planteado por Kent Beck fundador y director del Three Rivers Institute, empresa enfocada al desarrollo de software. Xp es usada en proyectos de software.

El ciclo de desarrollo consiste de seis fases que son:

Exploración, en esta fase el cliente especifica los requisitos de software para lo cual se emplean tarjetas de papel en las cuales el cliente describe de manera breve los requisitos funcionales y no funcionales que el sistema debe tener a dichas tarjetas se les conoce con el nombre de *historias de usuario*. Estas historias deben contener la fecha de la historia, si se trata de una nueva funcionalidad, una corrección o una mejora o una prueba funcional, el número de historia, la prioridad técnica y del cliente, alguna referencia a otra historia previa en caso de existir, el riesgo, la estimación técnica, la descripción, notas y una lista de seguimiento con respecto a la fecha, el estado cosas por terminar y comentarios. Las historias pueden especificar el tiempo de programación que no debe superar el tamaño de una iteración, por lo cual deber ser en un rango de una a tres semanas.

Planificación**,** se asigna a cada historia de usuario una prioridad y se hace un cronograma de acuerdo a dicha priorización para hacer una planeación de entregas de software.

Iteraciones, para cada entrega se discuten los objetivos que debe tener la misma y a la vez se definen las iteraciones necesarias para cumplir con los objetivos planteados en la entrega. Una vez concluida la iteración, su resultado (un programa) es entregado al cliente para que en base a su opinión defina las subsecuentes iteraciones, siempre y cuando esté satisfecho con la entrega de no ser así, se debe adaptar el plan de entregas e iteraciones hasta tener la aprobación del cliente .

Producción, se realizar pruebas extras del funcionamiento del sistema antes de liberarlo al cliente.

Mantenimiento, A pesar de dejar un sistema lo más cercano a lo deseado por un cliente después de un tiempo de uso se nota que se requieren nuevas funcionalidades que no se habían previsto o que los requisitos planteados al principio no fueron los más idóneas por lo que es necesario realizar algunas correcciones o inclusiones a lo ya terminado.

Fin. No existen más historias del cliente a ser incluidas en el sistema

Una de las principales cualidades de este método es que al igual que UP se distingue por ser *iterativo e incremental* ya que realiza los cinco pasos siguientes en la fase de desarrollo de manera incremental y luego iterativa hasta lograr el producto que reúne las expectativas del cliente:
Primero.- El usuario final y/o el cliente determinan las reglas de negocio (requerimientos, especificaciones etc.,) que deben ser implementadas.
Segundo.- El desarrollador o grupo de desarrollo estima los recursos tangibles e intangibles necesarios para llevar a cabo la implementación.
Tercero.- El cliente hace una priorización de lo que debe ser construido de acuerdo a las restricciones de tiempo.
Cuarto.- el desarrollador o grupo de desarrolladores elaboran conforme a los acuerdos.
Quinto.- El cliente hace un revisa y evalúa lo desarrollado y en caso de ajustes se regresa al primer paso.

Otra cualidad de este método es que hace uso de *pruebas unitarias* con lo cual se garantiza que cada uno de los módulos por separado funcione correctamente, ya que se prueba el correcto funcionamiento del código en cada módulo.

Otra característica de esta metodología es la de *Programación en parejas*, donde dos programadores son puestos en conjunto a programar un mismo módulo, con la presunción de que él código escrito de esta forma aunque sea lenta será de mejor calidad ya que es revisado y discutido al mismo tiempo

que se escribe.

Otra más y fundamental en esta metodología es que el cliente o un representante del mismo, ya sea éste uno o más usuarios finales que van a manejar y conocen de antemano todas las especificaciones que el sistema debe cumplir, debe *interactuar constantemente con el equipo de desarrolladores*.

Otra característica más es la *depuración de todos los errores* antes de añadir una nueva funcionalidad, para lo cual se deben hacer entregas frecuentes.

Refactoring, es decir el mejorar el código ya escrito, cambiando su estructura interna sin modificar su comportamiento externo, es otra de las características en que se basa XP.

También XP pretende que aunque se programa en parejas se debe tener una *propiedad del código compartida*, ya que de esta manera se presume pueda corregir y ampliar cualquier parte del sistema, pero para ello se deben aplicar frecuentes pruebas de regresión que garanticen que los probables errores de este proceso puedan ser detectados.

Y por último *el código debe ser simple*, para que facilite el mantenimiento o las correcciones y/o ampliaciones que se presenten en las subsecuentes iteraciones.

Desgraciadamente este método tiene grandes fallas, la primera de ellas es la de la programación en parejas, ya que sus creadores al ser ellos excelentes programadores y saber trabajar en equipo, se les olvido el pequeño detalle de que un gran porcentaje de los programadores tienen un alto coeficiente de ego, por lo cual les resulta casi imposible admitir que otro puede programar mejor que ellos, con lo cual el supuesto de ser un metodología que da resultados rápidos puede verse frustrada al verse estancada en la discusión de la pareja al programar. Y aún a pesar que la pareja de programadores no pertenezca a esta categoría, esta práctica puede correr el gran riesgo de provocar resentimientos. Por la misma situación pasa la característica de compartición de código, ya que muchos egos se sentirían afectados por el hecho de que otros programadores se atreviesen a encontrar fallos o sugerir mejoras a un código de otro programador.
Otro factor es la característica de sólo un cliente o representante del mismo sea el único interactuando constantemente con el equipo de desarrollo, ya que si dicho cliente o representante no tiene la experiencia suficiente en los procesos que debe realizar el sistema, los requisitos funcionales y no funcionales, escenarios de funcionamiento y de prueba, éstos pueden llegar a ser insuficientes o endebles y quizás hasta mal planteados, provocando el tener cambios demasiados frecuentes

haciendo con ellos casi nulo el avance del proyecto.
Por último el hecho de que XP evita cualquier tipo de documentación es una de las más grandes fallas de este método. Ya que a pesar del hecho de que la programación sea simple, el porqué de ciertos requisitos funcionales puede perderse pues no existe un registro explícito de los acuerdos y el mantenimiento después de un lapso de tiempo largo puede resultar difícil, sobre todo si hubo un cambio en el grupo de programadores (ya no existen los que hicieron el desarrollo).

Sin embargo XP parece dar muy buenos resultados cuando las personas que intervienen en el proyecto incluido el cliente y su representante no sobrepasa de 10 esto se debe al número de líneas de comunicación que puede llegarse a dar entre ellas (el factorial del número de personas involucradas) y que sea para proyectos internos, es decir, donde el equipo de desarrolladores pertenece a la misma empresa donde es requerido el proyecto, puesto que existe mayor entendimiento de los objetivos de la empresa y conocimiento de los procesos que se realizan en ella.

Algunas Metodologías Orientadas a Objetos para desarrollo en Web.

Debido al gran auge que ha tenido el uso de la Internet, muchas aplicaciones comerciales se ven inclinadas a ser desarrolladas en este ambiente. Es por ello que han surgido metodologías orientadas a objetos para desarrollo Web, a continuación se describen sólo algunas de ellas.

Scenario - based Object-oriented Hypermedia Design Methodology, mejor conocido por sus siglas en inglés como SOHDM, hace como su nombre lo implica el uso de escenarios para recopilar los requerimientos del usuario. Es apartir de estos escenarios que se elaborá el modelo conceptual del sistema. El ciclo de vida de esta metodología, consta de seis fases: fase de análisis del domino, fase de modelado de los objetos, fase del diseño de las vistas, fase del diseño navegacional, fase del diseño de la puesta en marcha y por último la fase de construcción donde se crea e incorpora al sistema la bases de datos.

Object-Oriented Hypermedia Design Method, u OOHDM por sus siglas en inglés, este método es iterativo e incremental , la vez que hace uso de prototipos en su desarrollo. Consta de cuatro fases: Fase de Conceptos, en la cual se representa de manera conceptual los objetos, sus relaciones y la colaboración entre ellos, Fase de Navegación, en la cual se crean vistas de la navegación de las clases(estructuras de acceso, enlaces y nodos) y los contextos que no son más que los espacios navegables(grupos de clases que harán un recorrido en un orden particular). Fase de diseño abstracto de la Interfaz, en esta se muestran las vistas que tendrán los objetos al a través de la

navegación, así como su funcionalidad, y por la fase de implementación, en la que se debe integrar todo conforme al entorno web.

Enhanced Object Relationship Methodology, llamada EORM por sus siglas en inglés. Consta de cuatro fases:

Análisis, en la cual el modelado de los objetos se hace utilizando la misma notación que usa OMT, sin hacer ninguna consideración de tipo hipermedia (texto, audio, video, etc., que interactúan con el usuario).

Diseño, se añade la apropiada semántica a las asociaciones entre objetos para transformarlas en vínculos hipermedia, mostrando así las posibilidades de navegación que tendrá el sistema.

Construcción, se hace la codificación y se crea e incorpora la base de datos.

1.3 Diagramación.

Un modelo es una representación gráfica de la forma en que se desarrollará un sistema. Dado que existen varias y diferentes vistas a lo largo del desarrollo de un sistema, de la misma manera existirán diferentes modelos del sistema.

La diagramación es utilizada para modelar todo un sistema y como ya se mencionó en el apartado anterior actualmente los diagramas que más se utilizan mundialmente en el desarrollo de orientado a objetos son los contenidos en el lenguaje unificado de modelado conocido por sus siglas en inglés UML, el cual fue estandarizado en 1996 por la OMG.

Su éxito ha sido tan grande que los entornos de desarrollo más extendidos en el mundo como son los de IBM, Microsoft y Borland incluyen herramientas de modelado de UML. Por otra parte OMG ha extendido la potencia de UML con OCL, que es un lenguaje usado para escribir restricciones y expresiones sobre elementos de los modelos, por ejemplo cuando es necesario restringir los valores de ciertos objetos (cuales serán los valores permitidos) o definir algunas invariantes, pos condiciones o precondiciones, con lo cual éstos resultan más precisos y completos. También se ha incorporado a UML una especificación que ayuda en la compartición de modelos, con el intercambio de diagramas conocida como XMI por sus siglas en inglés (XML Metadata Interchange), que sirve para hacer un esquema XML pero graficado de los elementos que serán utilizados en el modelo.

UML ha atravesado por varias versiones desde la estandarizada 1.0 hasta la actual 2.1. Esta versión consta de 13 Diagramas que son divididos en dos categorías

Categoría de Diagramas del Modelado de la Estructura, a la cual pertenecen los diagramas

- Diagrama de Clase o Estructural.- Define la construcción del modelo del sistema en bloques básicos: clases y material en general utilizados para la construcción de un modelo completo.

- Diagrama de Objetos.- Usado para mostrar las instancias de un diagrama de clases en un instante de tiempo determinado en la vida del sistema.

- Diagrama de Estructura Compuesta.- Este diagrama sirve para modelar la estructura interna de un clasificador mostrando su configuración. Un clasificador es un concepto abstracto que tiene identidad, estado, comportamiento y relaciones; y que es usado para representar un actor, a una clase, a una componente, a una interface, a un caso de uso, a un nodo, a un subsistema, incluso a una seño a un tipo de dato y etc.,

- Diagrama de Componentes.- El diagrama de componentes tiene un nivel superior al de clases y por lo general representa la implementación de un conjunto de ellas. Por ejemplo una librería dinámica. Por lo regular las componentes soportan interfaces para comunicarse con otras componentes.

- Diagrama de Despliegue.- Modela la arquitectura que en tiempo de ejecución de un sistema, mostrando los nodos del sistema y las componentes dentro de los mismos.

- Diagrama de Paquete. Este diagrama es utilizado para tener una visión de alto nivel del sistema al observar las interacciones que existen entre diversos paquetes.

Y la categoría de Diagramas del Modelado del Comportamiento, entre los que se encuentran

- Diagrama de Actividad. Tiene múltiples usos que abarcan desde el flujo básico de un programa hasta el mostrar de forma general los puntos de decisión y acciones en cualquier proceso que abarca el sistema.

- Diagrama de Estado.- Usados para representar el estatus quo del sistema en un determinado tiempo de ejecución del sistema.

- Diagrama de Caso de Uso.- Utilizados para modelar las interacciones que el usuario tendrá con el sistema.

- Y los Diagramas de Interacción los cuales a su vez son una subcategoria y está compuesta por los siguientes diagramas

    - Diagrama de Secuencia.- Muestran las líneas de vida de los objetos y la secuencia de comunicación con otros objetos dentro de un escenario en particular de un caso de uso.

    - Diagrama de Comunicación (colaboración).- Muestran la secuencia numerada de los mensajes entre aquellos objetos que colaboran entre sí en un instante en tiempo de ejecución.

    - Diagrama de Vista de Interacción. Es como un diagrama de actividad pero que incluye, los elementos y las ocurrencias de interacción.

    - Diagrama de Tiempo. Son utilizados para mostrar los cambios de los valores de uno o más elementos sobre el tiempo de uso del sistema.

EL grupo OMG influenciado por la gran aceptación que ha tenido UML, empezó en el año 2001 el desarrollo de una propuesta de Arquitectura Dirigida por Modelos, la cual se conoce como MDA (Model Driven Architecture), que actualmente es un estándar y en la que se contempla el Desarrollo Dirigido por Modelos (conocido por sus siglas en inglés como MDD), en lugar de un desarrollo dirigido a generar código, por lo que para MDA el modelado debe ser independiente de la plataforma, con lo cual se busca una mayor reusabilidad e interoperabilidad entre sistemas, así como minimizar en lo posible el tiempo de desarrollo de un sistema. Es por ello que UML puede ser utilizada en una amplia variedad de modelos, como modelar los requerimientos abarcar un diverso número de modelos, pues se puede utilizar para modelar los requerimientos, como el flujo de trabajo del sistema, la estructura lógica del sistema, la comunicación e interactuación, así como el modelo físico.

Los diagramas de UML son utilizados en las vistas de la arquitectura de 4+1 vistas, que fue descrita en la sección 1.1 de este libro, conforme lo muestra la figura

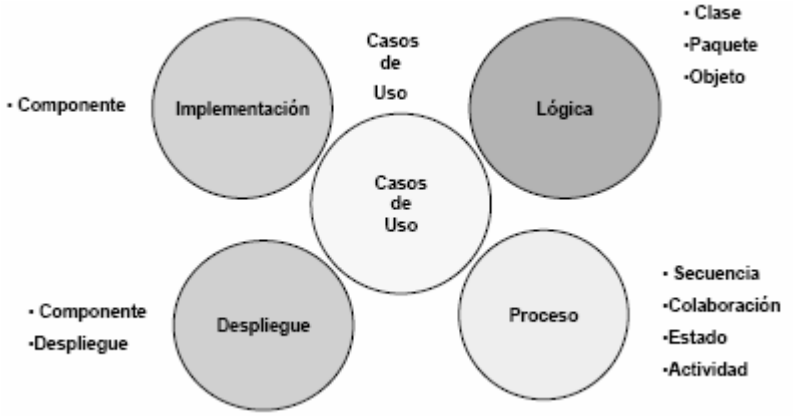


Figura Los diagramas de UML dentro de la arquitectura de 4+1 vistas.

Por otra parte Rumbaugh cataloga a los diagramas en vistas.

Una vista de UML es una forma de enlazar el lenguaje de modelado con el método elegido para el desarrollo, de tal forma que una vista forma un subconjunto de diagramas para abstraer y así modelar un aspecto del sistema

Estas vistas son agrupadas en tres rubros.

- Estructural. En la cual se describen los objetos involucrados en el sistema así como su estructura interna y sus relaciones con otros objetos..

- Dinámico.- En este se muestra el comportamiento y el estatus quo del sistema en el tiempo de ejecución.

- Gestión de Modelo.- Muestra la jerarquía de los modelos del sistema.

Dentro del rubro estructural se tienen las vistas:

- Estática. En donde se usan los diagramas de clases, de objetos y de estructura

compuesta., para mostrar una vista estática del sistema desde un enfoque riguroso.

- Casos de Uso.- Donde se aplican los diagramas de casos de uso, por medio de los cuales queda plasmada la interactuación que los actores tendrán con el sistema.

- Implementación.- Aquí se utilizan los diagramas de componentes, para mostrar las componentes que conformarán el software en forma ya física.

- Despliegue. En esta vista se hace uso de los diagramas de despliegue, con la finalidad de dar un panorama de cómo el software estará distribuido a través de los nodos físcos del sistema.

Dentro del rubro se tienen las vistas

- Máquina de estados. Como su nombre lo indica serán usados los diagramas de estados, para visualizar el estado de los objetos en tiempo de ejecución.

- Actividad.- Hace uso de los diagramas de actividad para representar el flujo de trabajo de un proceso, caso de uso, programa etc.,

- Interacción. Son aplicados en esta vista los diagrama de secuencia, y de comunicación para mostrar la forma y secuencia en que los objetos colaboran entre sí.

Y por último el rubro de Gestión de modelo, que como su nombre lo indica tiene una vista de gestión de modelo en la cual se hace uso de los diagramas de clases que pueden estar contenidos en paquetes para representar subsistemas del modelado del sistema.

Ya que una más amplia explicación del uso de algunos de estos diagramas, así como su vista serán mostrados en el siguiente capítulo, en este subtema sólo se ha incluido su descripción de acuerdo a su versión 2.1, sin embargo cabe hacer la aclaración que a pesar de que a partir de la versión 2.0 todos los diagramas son dibujados dentro de un rectángulo que en su esquina superior izquierda

posee una etiqueta para indicar el tipo de diagrama según la notación mostrada a continuación. En el siguiente capítulo no se usará en la mayoría de los diagramas debido al espacio y la legibilidad ya que ambas pueden verse afectadas al emplear este recuadro.

Etiquetas usadas para el recuadro de los diagramas UML a partir de la versión 2.0

- **act** Diagrama de Actividad
- **uc** Diagrama de Casos de Uso
- **cd** Diagrama de clases
- **sd** Diagrama de Secuencia
- **stm** Diagrama de Máquina de Estados
- **pkg** Diagrama de Paquete
- **cmp** Diagrama Componentes

Preguntas

1. ¿Qué significa la palabra arquitectura desde el punto de vista de desarrollo de software?.
2. ¿Qué significa la palabra patrón desde el punto de vista de ingeniería de software?.
3. ¿Cómo define Philippe Kruchten la arquitectura de software?
4. ¿En que consiste la Vista de los Casos de Uso?
5. ¿En que consiste la Vista de Proceso?
6. ¿En que consiste la Vista de Despliegue?
7. ¿Qué es un estilo arquitectónico?
8. ¿En que consiste la arquitectura de repositorio?
9. ¿En que consiste el modelo Modelo-Vista-Controlador?
10. ¿Cuántos modelos de arquitecturas en capas existen?
11. ¿Cuál es la diferencia principal entre las arquitecturas orientadas a objetos y las arquitecturas basadas en componentes?
12. ¿En que consiste el concepto de abstracción desde el punto de vista de desarrollo de software?
13. ¿Cuál es la diferencia entre objeto e instancia y como se define a ambos términos?
14. ¿Cuál es la diferencia ente una clase y una superclase?
15. ¿En que consiste el concepto de generalización desde el punto de vista de desarrollo de software?
16. ¿En que consiste el concepto de especialización desde el punto de vista de desarrollo de software?
17. ¿En que consiste el concepto de mensaje desde el punto de vista de desarrollo de software?
18. ¿En que consiste la diferencia entre polimorfismo y métodos sobrecargados?
19. ¿Cómo se da la reutilización desde el punto de vista de desarrollo de software?
20. ¿En que consiste el concepto de asociación desde el punto de vista de desarrollo de software?
21. ¿Cuáles son las diferencias entre los diferentes tipos de asociaciones?
22. ¿Cuál es la diferencia entre el diseño estructurado y el diseño orientado a aobjetos?
23. ¿Qué es un ciclo de desarrollo y que etapas lo conforman?
24. ¿Cuáles etapas conforman al método RUP y en que consisten cada una de ellas?
25. ¿A que se debe el éxito de la metodología de Proceso Unificado?
26. ¿En que consiste eXtreme Programming?
27. ¿A que tipo de metodología pertenece SOHDM, y en que consiste?
28. ¿Qué es UML?

29. ¿En que categorías de diagramas se divide UML?

Ejercicios

1. Dentro de su ámbito de estudio o trabajo, investiga que producto de software fue desarrollado en base a componentes y cuales son estas.
2. Dentro de los IDE (Integrated Development Environment) o Entorno de desarrollo que usted utilice para programar,(por ejemplo Netbeans, Microsoft Visual Studio, etc,) investigue la manera que estos crean componentes.
3. Investigue en que consiste el modelo de CORBA.
4. Dentro de sus papeles comerciales busque una factura y describa los objetos de software que encuentre.
5. Basándose en el ejercicio 4, analice y encuentre los atributos de los objetos.
6. Basándose en el ejercicio 4, analice y encuentre las posibles asociaciones entre los objetos?
7. Basándose en el ejercicio 4, analice y determine las clases, atributos y métodos necesarios de cada clase para procesar y reproducir la información contenida en la factura.
8. Basándose en el ejercicio 4, analice y determine si existen o no clases tipo asociación, de ser así diga cual es el atributo que las crea.

Lecturas Recomendadas.

Libros:

L. Bass Et Al. “Software Architecture In Practice”. 2003 ,Addison-Wesley.

Joyanes Aguilar, Luis. “Programación Orientada A Objetos. Conceptos,Modelado, Diseño y Codificación en C++”.1996, Mcgraw-Hill.

Larman, C. "Uml y Patrones: Introducción al Análisis y Diseño Orientado a Objetos". 2003, Prentice Halll.

Stephen R. Schach."Ingeniería Del Software Clásica Y Orientada A Objetos". Sexta Edición.. Mc Graw Hill

Articulos:

Kruchten Philippe, The 4+1 View Model Of Architecture

Páginas Web.

Http://St-Www.Cs.Uiuc.Edu/Users/Smarch/St-Docs/Mvc.Html, Applications Programming In Smalltalk-80(Tm):How To Use Model-View-Controller (Mvc) Bysteve Burbeck, Ph.D

2. Diseño Orientado a Objetos.

Los sistemas de software son cada vez más grandes y complejos, lo cual presenta un gran dilema para los encargados de desarrollar software ya sea de naturaleza comercial o para desarrollos propios para empresas.

El diseño orientado a objetos ayuda a manejar la complejidad, esto se debe principalmente a su habilidad de crear un conjunto de modelos del sistema gracias a los cuales es posible ver a dicho sistema desde distintas perspectivas, las cuales sirven para entender y profundizar en el desarrollo del sistema a cada una de las personas que estarán vinculadas en la planeación y elaboración del sistema, desde los clientes finales, pasando por los analistas, diseñadores, programadores integradores hasta los probadores.

Los modelos bien construidos, son fáciles de comunicar, cambiar, expandir, validar y verificar, de ahí la importancia que han tenido un conjunto de herramientas para modelar los sistemas como la de Rational Rose, ArgoUML, EclipseUML, entre muchas otras.

Por otra parte gracias a la gran aceptación que ha tenido la programación orientada a objetos, se cuenta en la actualidad con un gran número de entornos de desarrollo orientados a objetos muchos de los cuales tienen integrados modeladores haciendo uso de UML, lo cual resulta de gran ayuda al momento de diseñar y construir la aplicación.

Por supuesto de nada sirven las herramientas de modelado ni las entornos de desarrollo, sin una buena metodología, es por ello que la elección de una metodología adecuada al tipo de sistema y al ámbito del mismo es muy importante. Por lo que en la actualidad se cuenta con un gran número de metodologías para cada tipo de desarrollo, así cuando se requiere un sistema de gran tamaño es indispensable contar con una metodología asociada con un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y diagramas, incluyendo modelado y documentación detallada. Por otro lado si el proyecto es pequeño y requiere reducir drásticamente los tiempos de desarrollo, se puede hacer uso de algunas de las metodologías ágiles. Y para aquellos proyectos en que no se cuenta con un gran número de recursos, ni una extensa complejidad, pero si varios tipos de usuarios en una moderada cantidad de servidores y equipos, puede hacerse uso del proceso unificado.

Sin embargo no existe ninguna ley que diga que metodología debe ser aplicada a un específico sistema de software, dada la gran variabilidad existente de recursos humanos, experiencia, equipos, políticas y restricciones marcadas por las empresas u organismos gubernamentales, etc., intrínsecos del ámbito del sistema. Es por ello que por lo regular cada equipo de desarrolladores o empresa optan por establecer su propia metodología, que por lo regular se basa en una o en la combinación de las metodologías existentes, tomando ciertas etapas o pasos de ellas.

Sin embargo no importando la metodología, hay ciertos puntos en que todas coinciden, primero el sistema a desarrollar sustituirá o optimizará procedimientos para procesar información, por lo que las funciones o actividades de dicho procesamiento debe ser analizada y modelada, para lo cual es muy común la descripción de escenarios de dicho procesamiento. Segundo la mayoría de los sistemas cuentan con interfaces de usuario, a través de los cuales los usuarios pueden tomar decisiones sobre puntos de control o de transformación o visualización de la información. Tercero en un paradigma orientado a objetos, los objetos serán los que den soporte a la aplicación y pueden ser analizados y vistos de forma estática y también de forma dinámica y por otra parte serán objetos los encargados de manipular la base de datos ya sea a través de un manejador de base de datos o directamente. Cuarto todo sistema debe ser probado antes de implementarse y quinto ningún sistema es perfecto por lo cual siempre requerirá de mantenimiento.

Es por ello que en este capítulo el primer subtema constará de ilustrar los artefactos empleados en el diseño del sistema en base al análisis de aquellas actividades y escenarios en los que se requiere el tratamiento de información y en el último subtema se mostrarán los artefactos usados en el diseño de las clases que soportarán a una aplicación, tanto desde su perspectiva estática como dinámica.

2.1 Diseño del sistema en base a procesos

Dado que los sistemas informáticos automatizan procesos, al usuario final le interesa que su sistema funcione de acuerdo a los requerimientos funcionales como no funcionales y que desde su punto de vista satisfagan sus necesidades. De tal forma que realmente le simplifiquen su trabajo haciendo que este no requiera de un esfuerzo de aprendizaje del sistema que le lleve mucho tiempo y que la información que el sistema le proporcione sea lo más completa y precisa posible.

Por estas razones es de vital importancia que el grupo de desarrolladores comprendan perfectamente el flujo de información y los procesos que involucran dicho flujo, así como las expectativas del usuario

respecto a la funcionabilidad del sistema por una parte y por otra parte el desarrollar una interfaz de usuario accesible para los diferentes tipos de usuario que tendrán el sistema, además de que esta interfaz sea fácil de usar y aprender. Por lo tanto primero se requiere realizar un ***Modelado del negocio*** y así conocer la organización en cuanto a su misión y funciones, su estructura y tecnología, sus debilidades y fortalezas; para comprender el entorno en el que va a funcionar el sistema, identificando sus procesos, la información, los actores participantes y los papeles que representan cada uno de ellos, en el procesamiento de la información. Una vez realizado este modelado se plasmará en un prototipo antes de proceder a su diseño para verificar que los requerimientos del usuario van siendo cumplidos hasta el momento.

2.1.1 Actividades y casos de uso

Para ayudar a los desarrolladores en la comprensión de las funciones con respecto al procesamiento de información que se desean sistematizar y la funcionabilidad deseada por el usuario UML ofrece para ser utilizado en el análisis de las funciones el diagrama de actividades y para el modelado de la funcionalidad el diagrama de actividad.

Diagrama de Actividad.

Este diagrama es utilizado para representar de manera gráfica una secuencia de pasos a seguir en una actividad los cuales pueden ser ejecutados de manera secuencial como se representa en la figura 31; o llevar a distintos caminos debido a un punto de decisión como puede verse en la figura 32; o ser ejecutados de manera paralela como se muestra en la figura 33, También el diagrama de actividades puede ser utilizado para mostrar los responsables involucrados en una misma actividad de realizar ciertas pasos, como puede ser observado en la figura 34.

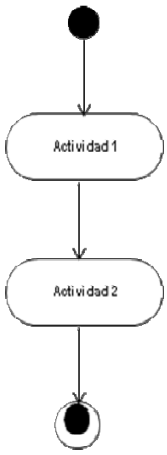


Figura 31 Diagrama de actividad secuencial

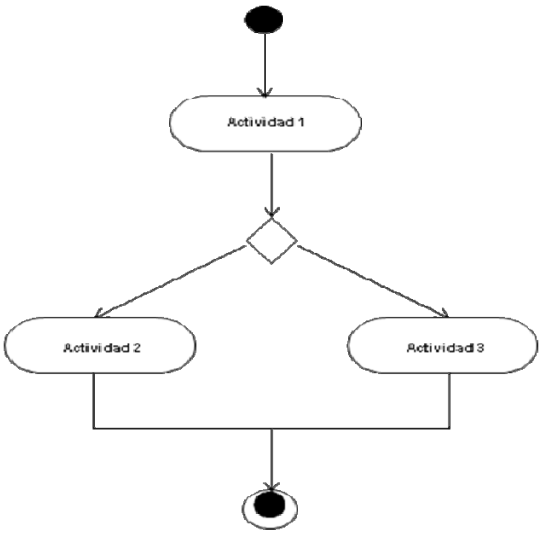


Figura 32 Diagrama de actividad con punto de decisión

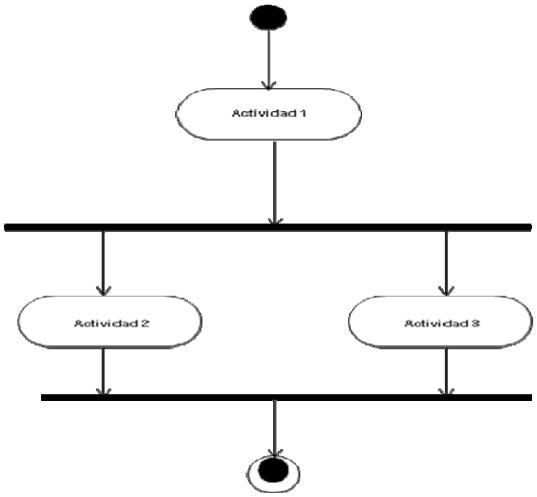


Figura 34 Diagrama de actividades en paralelo

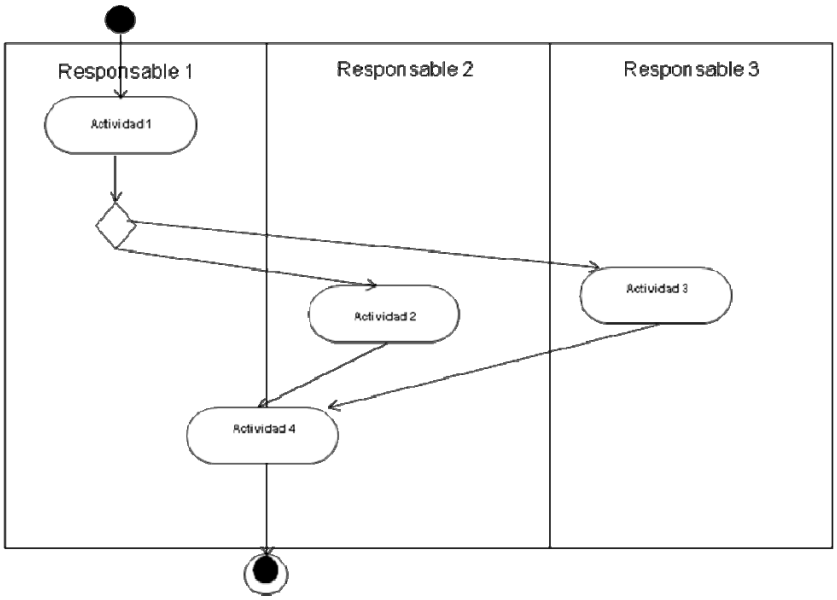


Figura 35 Diagrama de actividad con marcos de responsabilidad

A partir de la versión 2.0 de UML se incluyen nuevos artefactos gráficos al diagrama, como son los tipos de acción que representa cada paso en un diagrama de actividad, ya no sólo se utiliza el rectángulo cuadrado para indicar un paso pueden ahora representarse acciones como por ejemplo el envío de una señal, la aceptación de un evento etc., en sí son más de 40, en la figura 36. a) se muestran algunos de ellos. Lo mismo sucede con respecto a los puntos de decisión llamados “nodos de control” algunos de ellos pueden observarse en la figura 36. b)

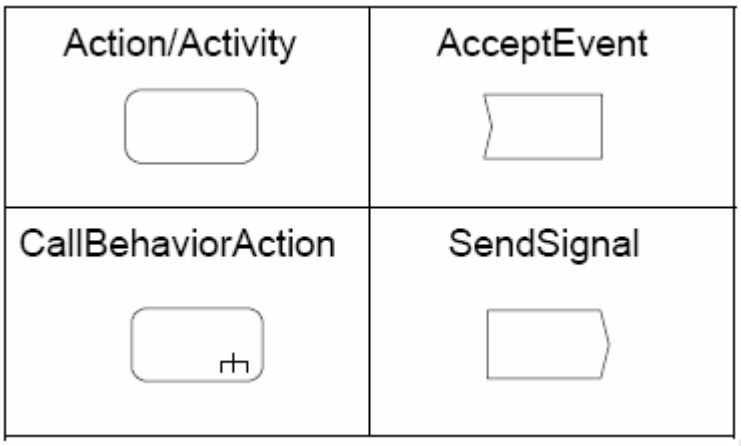


Figura 36 a) Tipos de acción en los diagramas de actividad

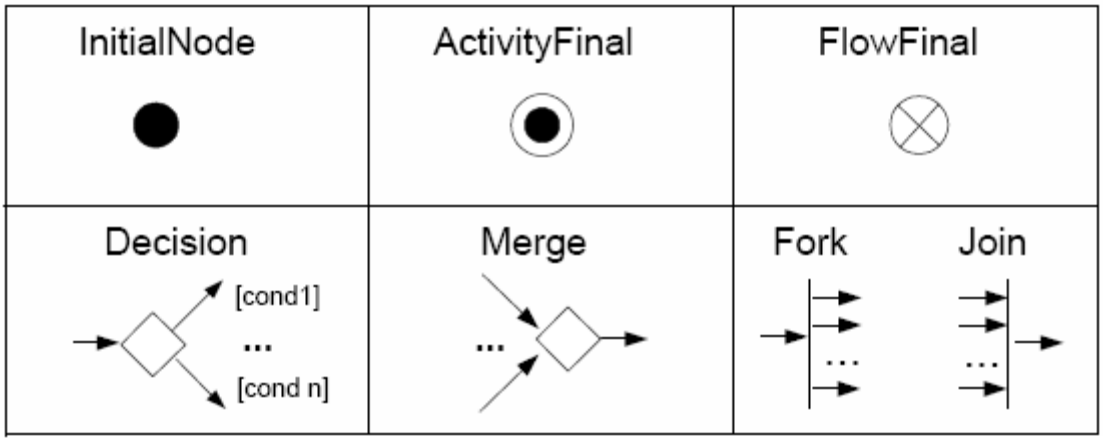


Figura 36 b) Tipos de nodos de control del diagrama de actividad

La aplicación de los diagramas de actividad es para darnos vistas micro o macro de los procesos de un sistema, ya que sirven para ilustrar de una forma simplificada lo que sucede ya sea dentro de una operación –una vista micro de las tareas del sistema- o dentro de un proceso – una vista mayor de los procesos del sistema.

Un ejemplo de uso de este diagrama sería en la operación de solicitud de contraseña para poder acceder al sistema como puede apreciarse en la figura 37 se muestra esta operación haciendo uso de un diagrama de actividad que muestra lo sucedido dentro de ella.

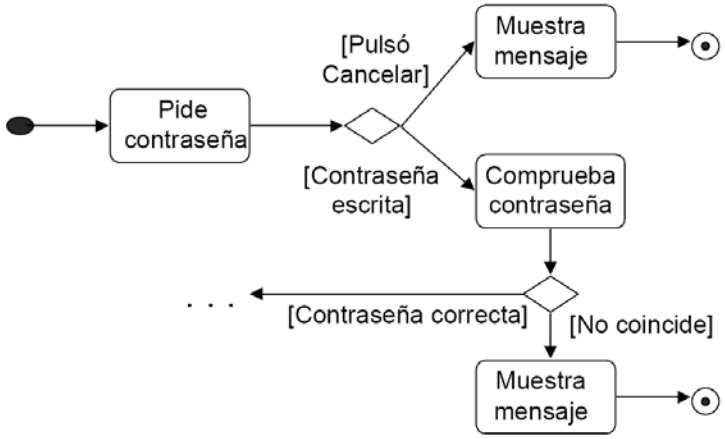


Figura 37. Ejemplo de uso del Diagrama de actividad en una operación.

Un ejemplo más puede ser observado en la figura 37 en la se muestra parte del diagrama de actividades con marco de referencia para ilustrar el proceso de titulación, desde que se solicita información hasta que se expide el acta de examen.

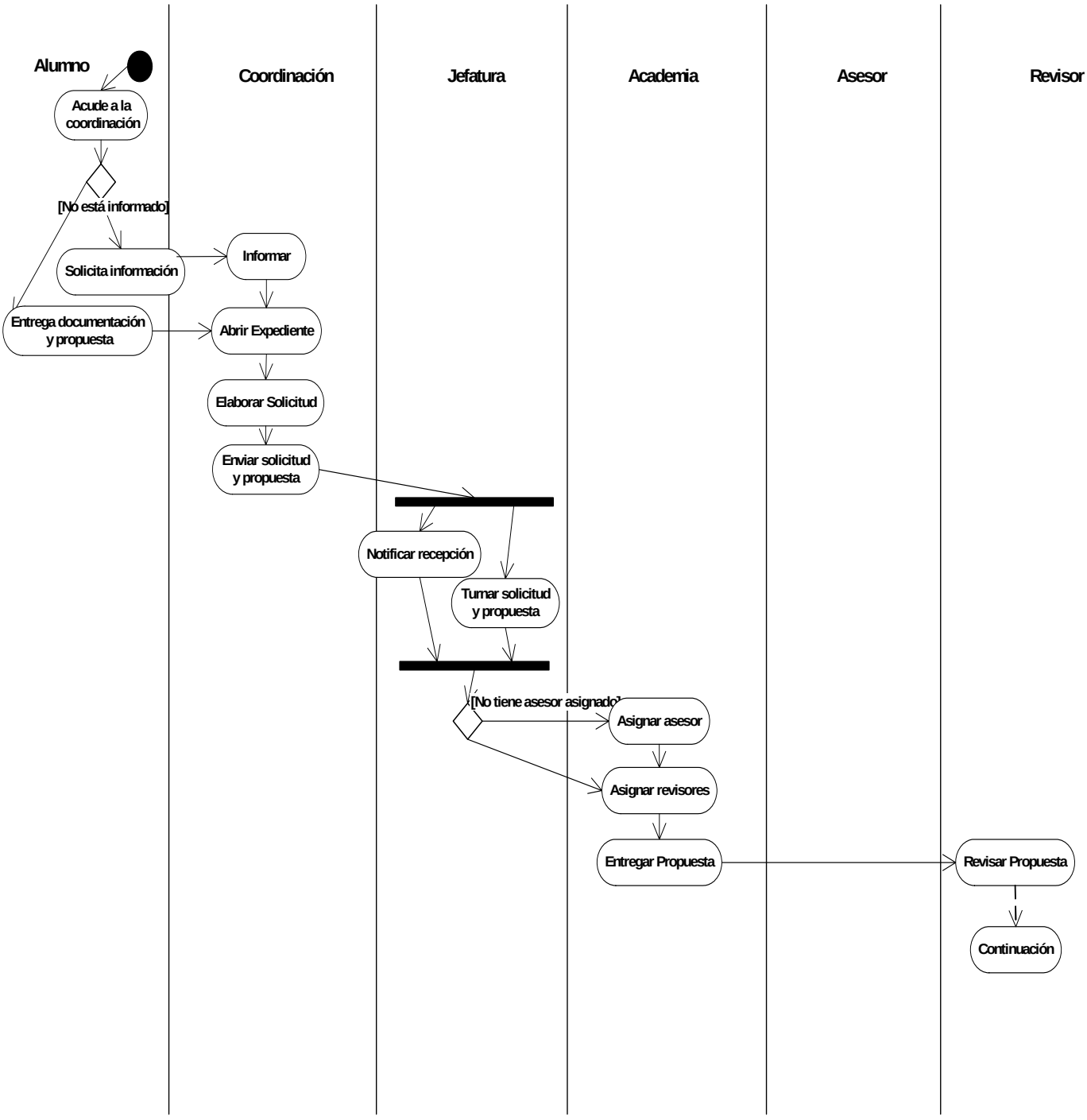


Figura 38. Ejemplo de uso de una parte de un diagrama de actividad con marco de referencia.

Diagrama de casos de uso.

Como ya se había mencionado anteriormente los casos de uso son utilizados para tener una vista de la funcionabilidad que tendrá un sistema cuando los usuarios -que son bautizados con el nombre

“actores”- interactúan con él sistema a través de las distintas interfaces de usuario que proveen dicha interacción en ciertas partes del sistema. A estas partes en las que el usuario interactuará con el sistema se le llaman casos de uso.

Un caso de uso de acuerdo a su creador Ivar Jacobson :
“Es *una forma, patrón o ejemplo concreto de utilización, un escenario que comienza con algún usuario del sistema que inicia alguna transacción o secuencia de eventos interrelacionados”*

Los casos de uso no son con orientación a objetos estrictamente hablando ya que “un diagrama de casos de uso del sistema” mostrará los “casos” o procesos en los cuales el sistema interactuará con los distintos tipos de usuario que puede atender un sistema. Este diagrama es una vista macro del uso del sistema.

Para aclarar el término “tipo de usuario”, considérese un sistema de ventas, así por ejemplo un tipo de usuario será una cajera, aunque el sistema sea utilizado por varias de ellas; o en su defecto un vendedor, aunque el sistema sea usado por varios vendedores. En el diagrama de caso de uso sólo se hace referencia y se graficará sólo al tipo de usuario o el papel que representa -por eso se le llama actor- no al número de usuarios que el sistema atenderá.

La simbología utilizada para el diagrama se muestra en la figura 39.

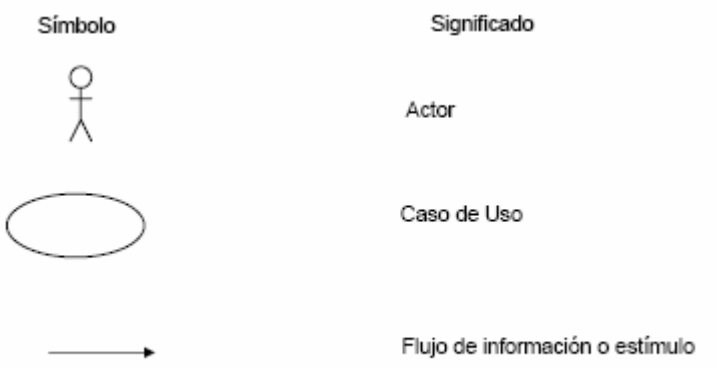


Figura 39: Simbología empleada en el diagrama de caso de uso.

Como ya se menciono anteriormente con el diagrama de casos de uso del sistema se tendría una perspectiva global del funcionamiento que brindará el sistema. En la figura 40, puede verse el diagrama de casos de uso del sistema de un cajero automático, en el cual puede observarse el flujo de información entre los actores y los casos de uso.

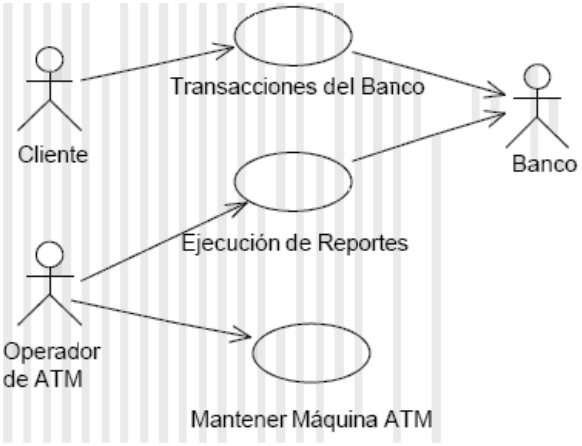


Figura 40. Diagrama de Casos de Uso del Sistema de un cajero automático

Cuando el sistema involucra muchos casos de uso, tantos que no es muy conveniente ponerlos todos a la vista en un único diagrama de casos de uso del sistema, se debe utilizar en su lugar el diagrama de paquetes dentro del este colocar los paquetes que representan a los módulo o subsistemas que conforman a un gran sistema, en la figura 41 puede verse un ejemplo de esto.

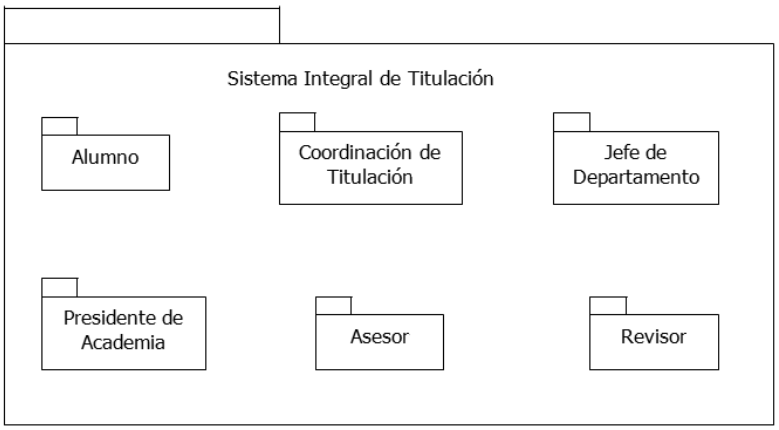


Figura 41. Paquete mostrando los paquetes o módulos que conforman u sistema

Por otra parte para saber qué casos de uso se encuentran en un paquete, la figura 42 muestra un ejemplo de un diagrama de casos de uso del sistema del paquete Alumno de un sistema de Registro y control del proceso de titulación.

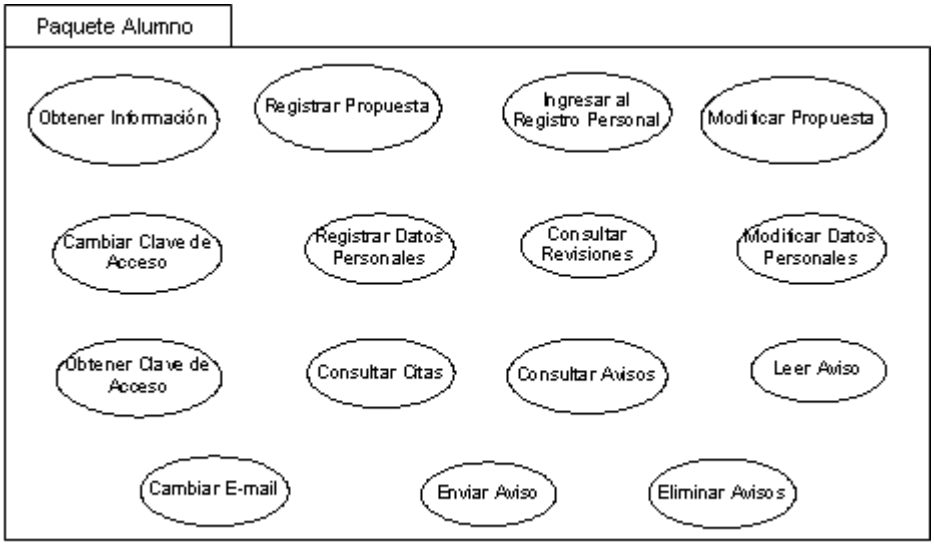


Figura 42. Ejemplo de Casos de uso en un Paquete

Los Casos de Uso también pueden tener relaciones entre ellos, como por ejemplo la de ***"Generalización"***, e la cual un caso de uso puede ser la generalización de uno o más casos de uso,o en su defecto como una especialización en la que de un caso de uso surge otro el cual refina la funcionalidad del original al agregar nuevas secuencias de eventos así como operaciones y/o atributos. La figura 43 muestra este tipo de relación.

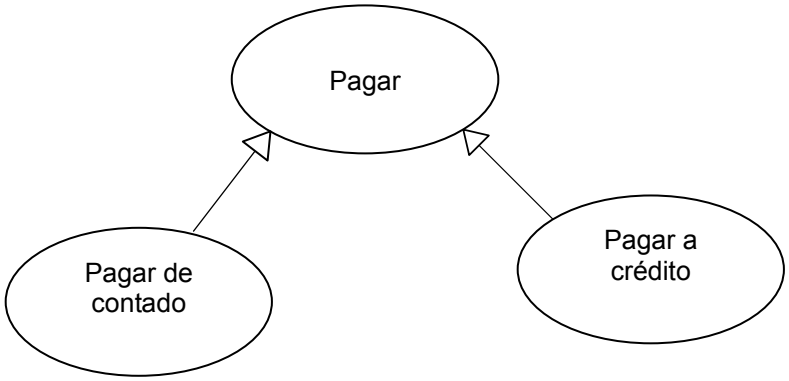


Figura 43. Ejemplo de la relación de Generalización en un Caso de Uso

Otra relación de los casos de uso es la "***Extensión***", la cual es utilizada cuando el caso de uso base requiere de añadirle alguna secuencia de eventos para sumarle un comportamiento adicional, es decir que amplía la funcionalidad mediante la extensión de sus secuencias de acciones, las cuales podrían deberse o no a una condición. Esto suele ser útil para sortear con casos de uso especiales, o para incluir nuevos requisitos durante el mantenimiento del sistema y su extensión. Cabe señalar

que la secuencia de eventos del caso de uso extendido tiene una localización por separado de aquel del que se extiende, esto significa que puede llegar a darse por concluida la secuencia de eventos del caso de uso original puesto que ya no exista la necesidad de concluirlo. La figura 44 muestra un ejemplo de esta relación.

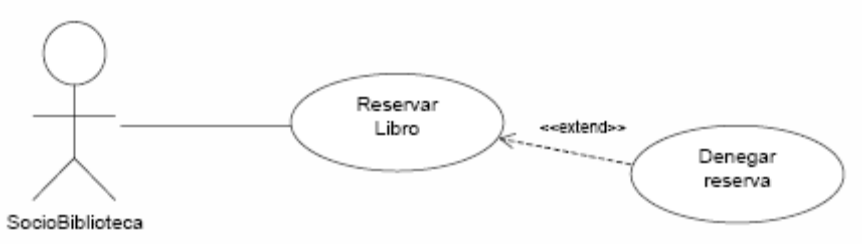


Figura 44. Ejemplo de relación de Extensión de un Caso de Uso

Y por último la relación de “Inclusión” en este caso existe una secuencia de pasos que dado algún punto de decisión en la secuencia de eventos del caso de uso original, es necesario realizar antes de concluir con dicha secuencia. Por lo tanto existe una dependencia del primer caso de uso por el resultado del caso de uso incluido. La figura 45 muestra un ejemplo de este tipo de relación.

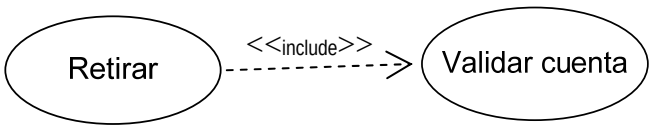


Figura 45. Ejemplo de relación de Inclusión en un Caso de Uso

Una pregunta que sin duda se haría el lector, es que ¿Cómo usando la anterior simbología puede verse la secuencia de interactuación entre el actor y el sistema?, es decir su funcionabilidad en un determinado caso de uso?

La respuesta es que dicha secuencia de eventos son mostrados en lo que se conoce como plantilla

de casos de uso.

Por lo anterior hay que aclarar que dentro de los casos de uso de un sistema se tienen dos artefactos que son por una el diagrama de casos de uso y por otra parte las plantillas de caso de uso.

Y también es muy importante resaltar el hecho de que existen dos tipos principales de casos de uso que son:

- *Casos de uso esenciales*:- Los que deben ser aplicados al inicio del análisis, donde lo más importante es entender la funcionabilidad que debe proveer el sistema no el cómo debe ser implementado, es por ello que estos casos de uso se llaman esenciales, ya que deben reflejar la "esencia" de cómo debe ser una correcta interactuación entre el "actor" y el sistema.

- *Casos de uso reales*. Estos casos de uso son aplicados en el diseño del sistema cuando se requiere conocer los objetos de diseño y sus métodos que deberán acompañar al caso de uso en su implementación

La descripción de ambos tipos de casos de uso se hace a través de ***escenarios.*** **Un escenario** es una descripción del flujo normal de eventos ocurridos durante la ejecución de una instancia de un caso de uso. Para está descripción formal se hace uso de plantillas. Por otra parte los casos de uso suelen tener cursos de eventos o rutas de seguimiento aparte de la normal. Así un caso de uso siempre tendrá un curso de eventos normales, donde la secuencia de dichos eventos resultan sin errores ni excepciones, a este curso suele llamársele ***escenario primario***. Pero también tendrá cursos alternos, a los que se les llama ***escenario secundario*** cuando se presenta algún error ya sea cometido por el usuario como por ejemplo la captura de un carácter en un campo completamente numérico, o el resultado de una excepción como por ejemplo cuando no es posible conectarse con el servidor de la base de datos.

La figura 46, muestra un tipo de plantilla de caso de uso en la cual los cursos de acción se dividen en normal y alterno.

Plantilla de Casos de Uso

Expandido

Caso de Uso:	Nombre del Caso de Uso
Actores:	Lista de Actores
Propósito:	Intención del Caso de Uso
Resumen:	Repetición del Caso de Uso de Alto Nivel
Tipo:	1. Primario, Secundario y Opcional 2. Esencial o Real
Referencias Cruzadas:	Funciones y Casos de Uso relacionadas con éste

Curso Normal de eventos:

Acción del Actor	***Respuesta del Sistema***
Describe la conversación interactiva entre actores y sistema.	

Curso Alternos:

Opciones o excepciones (errores) que se pueden presentarse en el curso normal.

Figura 46. Plantilla de Casos de Uso

En donde los tipos se clasifican de la siguiente manera

Primario: Procesos más comunes y/o importantes

Secundario: Procesos menores o raros (no frecuentes)

Opcionales: Procesos que pueden obviarse

Esenciales: Casos expandidos que no hacen referencia a la tecnología y tiene pocos detalles de implementación.

Reales: Describe un proceso a partir de un diseño concreto sujeto a las tecnologías específicas que utiliza.

Un ejemplo del uso de este tipo de platilla puede verse a continuación

Ejemplo: *Comprar libros y pagarlos en efectivo*

Caso de Uso:	Compra en efectivo de libros
Actores:	Cliente, cajero.
Propósito:	Capturar una venta y su pago en efectivo
Resumen:	Un cliente llega a la caja registradora con el o los libros que desea comprar, el Cajero registra los libros a vender y recibe un pago en efectivo, al terminar se marcha con sus libros comprados.
Tipo:	Primario y Esencial.
Referencias Cruzadas:	Funciones : - Registrar la venta - Calcular el total de las ventas (incluido impuesto y descuento) - Obtener la información del libro introduciendo su código - Calcular el saldo del pago

Curso Normal de eventos:

Acción del Actor	*Respuesta del Sistema*
1. El cliente llega a la caja con el libro o libros que desea comprar.	
2. El cajero introduce en el sistema el código del libro. El cajero introduce la cantidad de libros	3. Muestra el precio del libro y su descripción Calcula el subtotal
4. El cajero indica al sistema que concluye la captura	5.- Calcula el total de la venta
6. El cajero indica el total al cliente	
7. El cliente ofrece el pago en efectivo	
8. El cajero la cantidad en efectivo recibida	9. Calcula y muestra la diferencia entre el pago y el total de venta Genera un recibo o nota de venta
10. El cajero deposita el efectivo en la caja y extrae el cambio (si existe) El cajero da el cambio al cliente y la nota de venta	
12. El cliente se va con su(s) libro(s)	

Curso Alternos:

Línea 2. Código de libro no encontrado

Línea 9. El cliente no tiene suficiente dinero y cancela la cuenta

Figura 47. Ejemplo de Plantilla de Caso de Uso

Otro tipo de plantilla puede observarse en la figura 48 en esta plantilla los cursos alternos se llaman rutas y se trata de manera independiente los cursos alternos, de las excepciones, sin embargo se tratan sólo como una narrativa en donde están involucrados tanto las acciones del actor como del sistema sin tratarlas de manera independiente como en la plantilla anterior, es por ella que la anterior es la más usada. Por supuesto existen más tipos de platillas, que pueden ser alguna variación de estas dos.

Otra Plantilla de Casos de Uso

Nombre del Caso de Uso:	Dos o tres palabras
Descripción:	Corta narración, describiendo el estado y el propósito.
Autor del Caso de Uso:	Analista encargado
Actores:	Actores involucrados en este caso
Locación:	Lugar donde se ejecuta el caso de uso
Prioridad:	1 es la más alta
Suposiciones:	Lo que es cierto o es falso respecto a este caso.
Precondiciones:	Hechos que deberán ser verdaderos antes de que este caso de uso y sus rutas internas puedan ser iniciadas. Algunas veces llamadas estados del sistema requeridos para su ejecución.
Poscondiciones:	Hechos que deberán ser verdaderos cuando este caso de uso termina. Algunas veces llamados los estados del sistema requeridos al llegar a su terminación.

Ruta primaria:	Curso de acción normal
Rutas alternas:	Cursos de acción alternos a través del sistema
Rutas de excepción:	Cursos de acción de excepción o condición de error
Reglas de negocio:	**-Acciones restritivas.** Acciones prohibidas debido a las condiciones del sistema
	-Acciones iniciadoras. Acciones que inician una o más acciones cuando una condición es verdadera

Figura 48. Otra plantilla de caso de uso

También en la siguiente figura puede verse un ejemplo de este otro tipo de plantilla.

Ejemplo: *Comprar un artículo deportivo con tarjeta de crédito*

Nombre del Caso de Uso:	Pago con tarjeta de crédito
Descripción:	Un cliente paga una compra con tarjeta de crédito. El pago es validado por un servicio externo de autorización y se registra en un sistema de cuentas por cobrar.
Autor del Caso de Uso:	Analista encargado
Actores:	Cliente, Cajero, Servicio de autorización de crédito, Cuentas por cobrar.
Locación:	Boutique del Rancho Avándaro
Prioridad:	2
Suposiciones:	No se actualiza inventario de artículos, tienda independiente, no forma parte del Consorcio. Captura manual de la clave del artículo. No se lleva el registro de los clientes.
Precondiciones:	El gerente deberá haber iniciado el sistema, el cajero deberá haber entrado al sistema.
Poscondiciones:	Una venta queda registrada
Ruta primaria:	El cliente compra un artículo deportivo pagándolo con tarjeta de crédito
Rutas alternas:	
Rutas de excepción:	El servicio de autorización rechaza la tarjeta de crédito
Reglas del negocio:	**-Acciones restritivas**. No se admite tarjetas American Express **-Acciones iniciadoras.** Si la compra es mayor de $500.00 aplicar el 5% de descuento.

Figura 49 Ejemplo de otra platilla de Caso de Uso

Medios para encontrar casos de uso.

Existe básicamente una excelente forma, por medio de la cual se pueden hallar casos de uso que es a través de los actores. Si se identifican a todos los potenciales usuarios del sistema se debe identificar junto con ellos las tareas que involucran el procesamiento de información y que deberán ser incluidas en el sistema a desarrollar. Por otra parte se deben observar que eventos son externos al sistema pero que se relacionan con los actores para no perder de vistas posibles casos de uso que fuesen relevantes en el sistema.

2.1.2 Interfaces de Usuario

Desde el desarrollo de los primeros sistemas de información el reto por realizar interfaces de usuario ha llevado a la creación de disciplinas como la Interacción Persona Ordenador mejor conocida como IPO/HCI por sus siglas en inglés (Human Computer Interaction), que estudia el diseño, la evaluación y la implementación de los sistemas de información interactivos que utilizan las personas. El propósito de esta disciplina es que la interactuación entre usuario y computador sea eficiente al minimizar errores, incrementar la satisfacción del usuario y con ello disminuya la frustración, logrando con ello que la realización de las tareas que involucran a las computadoras y a las personas sea más productiva. Se basa en los preceptos de que los sistemas interactivos sean usables y accesibles.

Dado que este tema es bastante amplio, en esta sección sólo se tratarán algunos de los aspectos primordiales para la elaboración de interfaces de usuario y se enfocará principalmente a la creación de interfaces a partir de las plantillas de los casos de uso, ya que en ellas queda plasmado el comportamiento del sistema, es decir su funcionabilidad.

La primera consideración que hay que hacer es que una interfaz gráfica de usuario no es otra cosa más que el medio por el cual se comunican entre sí un usuario -al que de ahora en adelante se le llamará actor- y un sistema de información.

Debido a que el ser humano está involucrado en esta comunicación hay que tener en cuenta factores cognitivos como la memoria, la percepción y la forma de procesar la información, que se basa a su vez en los tipos de conocimientos, el nivel de experiencia, las fases y estrategias del conocimiento

que el actor posee y utiliza. Por supuesto también están involucradas las capacidades físicas del individuo principalmente la capacidad motriz y la capacidad visual.

La figura 50 expresa la relación entre estas capacidades

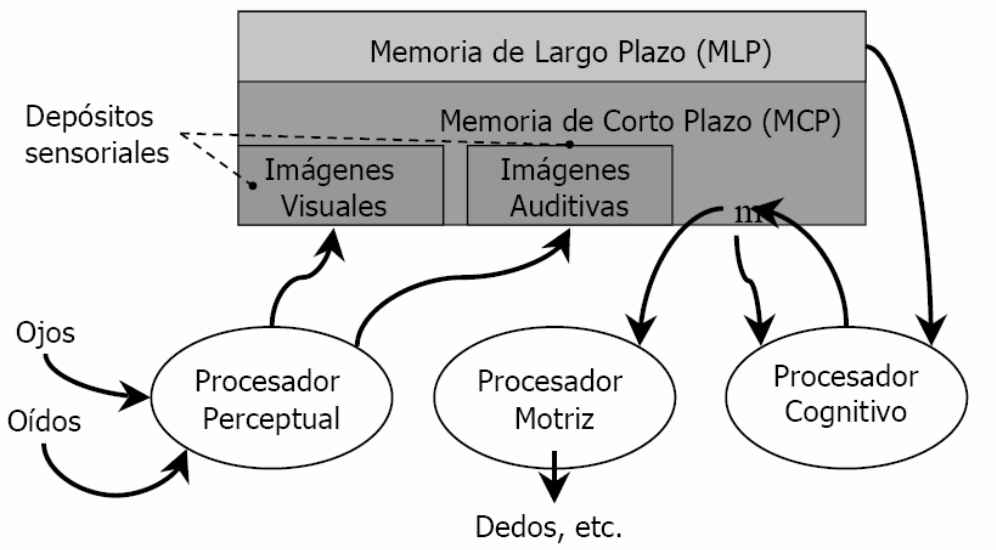


Figura 50. Capacidades involucradas en la interacción Actor- Interfaz Gráfica

Si algunos de estos factores no son tomados en cuenta puede dar como resultado que una interfaz no sea usable, por ejemplo una interfaz que exija al actor retener memoria de una ventana vista unos instantes antes dará como resultado que el usuario le resulte fatigoso el usarla y con el tiempo preferirá no hacerlo. Es por ello que la usabilidad es importante, para que el actor se sienta cómodo con el uso de la interfaz.

Por otra parte las interfaces son creadas para actores diferentes, en el que en cada tipo de actor pueden verse involucradas incluso cientos de personas, como por ejemplo en un sistema de información para alumnos de alguna universidad. Por tal razón las interfaces deben ser pensadas en las características cognitivas de la mayoría y no particularizar a un tipo de actor y a un tipo de persona, aunque dependiendo de las políticas o reglas del negocio sea necesario crear alternativas de accesibilidad para individuos con características especiales como por ejemplos para personas invidentes.

También hay que recordar que tras el diseño de una interfaz gráfica de usuario viene un código involucrado, por tal razón es de vital importancia el considerar el lenguaje de programación que será empleado de acuerdo al ámbito del sistema, puesto que por ejemplo si se tiene que implementar el

sistema en un computador que sólo puede proveernos de una pantalla en modo de texto, no sería muy sensato en pensar hacer uso de íconos.

Por otra parte la norma ISO 9241 indica que la interfaz que muestra un sistema debe adaptarse a las tareas que con él se desarrollan, debe proveer capacidad de auto explicarse y ser tolerante a errores introducidos por el usuario y su aprendizaje debe ser sencillo para que el usuario pueda tener control del él.

Antes de proceder con el diseño de las interfaces de usuario y por supuesto en conjunto con el modelado del negocio se deben tener en cuenta a mi parecer algunos aspectos que describo a continuación.

Conocer de antemano la plataforma hardware que de acuerdo a las limitaciones o políticas o reglas de negocio deberá ser empleada en el desarrollo del sistema, aunque al lector esto le parezca bastante obvio, no es lo mismo diseñar para un dispositivo portátil que para una computadora de escritorio o que para un sistema IBM antiguo y créame, toda vía existen algunas empresas que los tienen y lo usan, debido alto costo de su inversión.
Lo mismo para la plataforma software en la cual por las razones planteadas anteriormente deberá desarrollarse el sistema, ya que también quizás la empresa no desee comprar un compilador de lenguaje visual, pero quiere seguir trabajando en su sistema operativo Windows.

Percatarse de las habilidades cognitivas del o los programadores, principalmente el nivel de conocimiento y experiencia en la programación de las plataformas hardware y software que tendrán que ser utilizadas.

Reconocer que las interfaces más empleadas son las de exploración internet y las de escritorio tipo Windows y con ello quiero dar a entender que la mayoría de los potenciales usuarios están acostumbrados a ciertas metáforas como el escritorio con sus iconos representando carpetas y documentos y el uso de ventanas, botones, ratón etc., que actualmente utilizan la mayoría de los sistemas operativos

Una vez tomadas en cuenta estas consideraciones, se continúa con la descripción de un conjunto de reglas que han sido elaboradas por los expertos en la materia para el buen diseño de interfaces de usuario, orientándolas sin embargo a que las interfaces deberán surgir a partir de la descripción detallada de los casos de uso:

- Comprender la interacción. Dados los requerimientos y secuencia de acciones que debe llevar a cabo un caso de uso en particular, éste puede requerir una o más ventanas e incluso estar inmerso junto con otros casos de uso en una única ventana, los cuales pueden ser casos de uso incluidos o extendidos del primero. Sin embargo deben tenerse en cuenta los factores mencionados anteriormente, ya que quizás los casos de uso no estén bien planteados sino se considero que el sistema y sus interfaces iban a ser ejecutados en un dispositivo móvil cuya pantalla mide unos escasos 6 centímetros cuadrados con resolución de 320x200 pixeles en lugar de una pantalla de 19 pulgadas con resolución de 1440x900 pixeles y también el hecho de que en la memoria a corto plazo el tiempo de retención dura unos escasos doscientos milisegundos aproximadamente y que sólo algunas representaciones son conservadas para su tratamiento futuro, deberá considerarse que estilo de interacción será el más adecuado para la o las interfaces que cubren el caso de uso, las cuales pueden ser algunas, o la combinación de algunas de las siguientes, pero sin olvidar que el diálogo que estas brinden debe ser simple y natural, ya que el humano tiende a acordarse más de las cosas simples que de las complicadas:

    - Interacción a través de lenguaje natural: Concepto maravilloso si funcionará como en la computadora de la nave estelar “Enterprise” de la serie de televisión “Star Trek”, desgraciadamente a pesar del avance en el desarrollo de este tipo de interacción, es un tema que todavía no ha sido resuelto, debido principalmente a la ambigüedad del lenguaje, pero que no debe ser totalmente descartado, puesto que dependerá del dominio del problema, si es que su implementación para ciertos casos de usos es necesaria y factible.

    - Interacción a través de Comandos: Una de las más antiguas formas de interacción, usadas mucho antes que el sistema operativo DOS. Esta forma de interacción puede tener un alto grado de resistencia por parte de los actores, debido a que inicialmente son difíciles de aprender y recordar, sin embargo una vez aprendidas son bastante rápidos además de ofrecer una gran flexibilidad y libertad de movimiento al combinar parámetros dando al usuario un total control. Su uso es recomendado con usuarios experimentados y cuando las limitaciones de las plataformas lo requieran.

    - Interacción a través de formularios: Cuando el uso de un mismo tipo de información es muy frecuente ya sea para su consulta o actualización, el uso de formularios es recomendable, pero deben seguirse ciertos criterios en su uso; como el que la

información que contenga el formulario sea exclusivamente la que para el caso de uso es la pertinente y no colocar elementos que den como resultado una gran aglomeración de información que haga que se pierda de vista la información fundamental a tratar en el caso de uso. Por otra parte la información deberá estar organizada de una manera que resulte natural para el actor su visualización y su uso, por ejemplo si el formulario consta del llenado de una factura y lo primero que el actor está acostumbrado a escribir es el nombre del cliente y éste aparece al final de la forma, entonces dicho formulario no está organizado adecuadamente. También es importante que los elementos que muestra estén integrados adecuadamente, es decir que debe existir una lógica entre las conexiones de los elementos, siguiendo con el ejemplo anterior de la factura, además del nombre del cliente se espera ver también su dirección o por lo menos su RFC. Por último también deben cumplirse requerimientos tales como la precisión de los datos a mostrar, como sería el caso del número de decimales a visualizar.

- Interacción a través de menús: Su uso resulta factible tanto para usuarios novatos como para expertos. Una ventaja es que no requieren de mucho espacio, siempre y cuando no se presenten en el mismo espacio varios menús y subménus con muchos comandos., por lo cual pueden ser utilizados en dispositivos pequeños. Sin embargo debe tenerse mucho cuidado con su uso ya que debe asegurarse que los comandos en los menús representen lo que el actor espera y que se encuentren en donde sea más significativo para el actor.

- Interacción por manipulación directa: La cual gracias al uso principalmente del ratón sobre menús, botones, iconos, etc., es una de las más utilizadas debido a la retroalimentación constante que proporciona sobre la tarea que realiza el usuario, además de requerir poco entrenamiento en el uso de la interfaz.

- Reforzar la consistencia. Uno de los más importantes factores en el grado en que un diseño de interfaz facilita o dificulta el manejo de la misma (usabilidad) es la consistencia, ya que si una interfaz hace uso de la experiencia adquirida por el actor en el manejo de algunas otras interfaces que ya son familiares para él, el aprendizaje de la nueva interfaz se reducirá incrementando también el grado de usabilidad de la misma. La consistencia debe estar presente en los siguientes puntos:
    - Apariencia.- Estilo de las ventanas, colores utilizados, capitalización y tipos y tamaños

fuentes para títulos, campos etc., deben ser iguales en el mismo tipo de dispositivo que despliegue la interfaz, aunque pueden variar si la aplicación va a ser utilizada en diferentes dispositivos, por ejemplo una misma aplicación ejecutándose en una Palm o en una computadora de escritorio el tamaño y tipo de fuentes puede variar. Por lo que no hay que perder de vista que incluso los casos de uso deben ser adaptados a los dispositivos, ya que si la secuencia de acciones es muy larga, existe la probabilidad de que la información tratada en el caso de uso sea amplia tanto que no pueda ser visible en su totalidad en una sola ventana de un dispositivo móvil. De preferencia se recomienda siempre seguir o implementar un estándar.

- Misma reacción ante misma acción. Por ejemplo al pasar el puntero del ratón sobre un botón por lo común hace que resalte el nombre del mismo, no que de cómo resultado que lance alguna ventana.
- Tareas consistentes. Organizar la secuencia de eventos de tal manera que exista una analogía entre el tratamiento de la información como el actor estaba acostumbrado hacer antes del sistema y entre el uso de la interfaz de usuario del sistema. Son muchos los sistemas que fracasan al querer imponer a los usuarios una manera de realizar una tarea que les parezca antinatural como por ejemplo si se debe levantar primero el automóvil para quitar una llanta ponchada, ¿porque queremos que el usuario primero quite la llanta y luego levante el carro?

- Brindar ayuda y retroalimentación pertinentes. Es importante el colocar ventanas de diálogo que ayuden a los usuarios a completar tareas pero debe tenerse en cuenta que toda información adicional compite con la información relevante y disminuye su visibilidad relativa. Debe por lo tanto tenerse cuidado en no encimar las ventanas de diálogo sobre la información fundamental que sea el punto focal para la toma de decisiones en dicho momento. Para brindar la retroalimentación no necesariamente se requiere el uso de ventanas de diálogo, puede usarse de manera alterna una barra de estado o una ventana de mensajes en la parte inferior del área de trabajo o ventana principal, así el usuario se mantendrá informado acerca de lo que está pasando, cuando por ejemplo la aplicación se comunica con un servidor de base de datos para realizar alguna actualización. Algo que debe evitarse a toda costa son la intervenciones innecesarias ya que cuando un usuario está trabajando, su atención debe estar centrada en la tarea que realiza, para reducir la posibilidad de cometer errores. La mayoría posiblemente recordará a los ayudantes de Word, esas figuritas que servían como asistentes para brindar sugerencias de cómo hacer las cosas, que resultaban simpáticas al principio pero que después de un tiempo resultaban un verdadero fastidio además de una fuente de distracción, tanto que en las recientes versiones de Microsoft Office han desaparecido; son un

buen ejemplo de que la ayuda debe ser pertinente y de que menos en ocasiones es mejor, es decir, el diseño de ayuda y retroalimentación siempre debe ser simple y natural. Por otra parte es importante que el tiempo de respuesta en la retroalimentación ya que se considera un límite de 10 segundos para mantener la atención de un usuario hacia una ventana de diálogo.

- Manejo de errores adecuado. Los errores pueden presentarse tanto en el manejo de la interfaz como en la ejecución del código detrás de ella. Por lo que estos dos aspectos deben ser considerados en su diseño, lo cual resulta sencillo al apoyarse en aquellos escenarios de los casos de uso, que tratan con rutas alternas de acción para el manejo de excepciones. Con respecto a los mensajes que deben ser mostrados al actor cuando comete un error en al introducir algún datos, estos deben ser escritos en lenguaje sencillo pero que precise el problema e indique su solución, minimizando en lo posible el espacio que ocupe y que resalte el elemento en donde se cometió el error. También es conveniente agrupar en una sola información de error todos aquellos errores cometidos en una misma ventana, pero sin descuidar aspectos como el tiempo de retención de información, el espacio y el tiempo empleado en su corrección. Por ejemplo considérese un formulario cuyos campos ocupan toda una ventana, en la cual se ha introducido erróneamente datos en más de dos o tres campos, si se despliega encima de este formulario una ventana de diálogo en la cual se indican aquellos campos en los cuales se cometió un error y como remediarlo, se comete la falla de confiar en que el usuario tiene una memoria casi fotográfica, ya que cuando de un clic en el botón de aceptar dicha información ya no será visible. Por otra parte si se presenta una ventana de diálogo por cada error cometido, su corrección llevará mucho tiempo. En cambio si se resalta en un color diferente o inversión de colores en los campos que se cometió el error y al acercar el puntero de ratón aparece un “tooltip”(elemento gráfico que despliega un texto cuando el puntero del ratón pasa por encima de algún componente de interfaz gráfica, como un campo de texto, un botón, etc.,) para explicar la causa y corrección del error, dará como resultado que se enfoque el error, el espacio que se ocupa para señalar el error es mínimo al igual que el ocupado para explicar la causa del error y su posible solución y también resulta más rápida su corrección. Por otra parte siempre debe haber un mensaje detrás del tratamiento de excepciones, así por ejemplo cuando un sistema no puede comunicarse por alguna razón con un servidor de base de datos, lo cual es necesario para dar continuidad a la secuencia de eventos en un caso de uso, dando como resultado su terminación, debe informarse al actor de esta situación a través de una ventana de diálogo y en este caso en particular, dicha ventana será la que deba ser el punto focal del usuario, es decir, se pondrá por encima de las demás.

- Reducir la carga en memoria. Para cumplir con esto es conveniente tener en cuenta la experiencia y conocimientos previos al uso del sistema que tengan los usuarios, lo cual no está especificada en los casos de uso. Es por ello que es importante el uso de técnicas de recopilación de información como los cuestionarios para averiguar si los usuarios finales del sistema, tienen alguna experiencia previa en el uso de sistemas de información computarizados y en el manejo de computadores, así como cuáles metáforas del mundo real sería adecuado emplear dada su experiencia, ya sean estas metáforas organizacionales, como por ejemplo el agrupar ciertas interfaces de acuerdo a una estructura conocida (compras, ventas, devoluciones etc.,) o funcionales, es decir que la secuencia de acciones efectuadas en una interfaz sea similar a lo que el usuario hace en el mundo real sin el uso de la interfaz , o ya que sean visuales como el mostrarle la imagen de una agenda por ejemplo si va a registrar citas. Con ello se logra que el usuario no tenga que estar aprendiendo muchos conceptos nuevos y evitar así el recargo de su memoria. Y por último hacer uso de elementos gráficos que ayuden al usuario a seleccionar en lugar de introducir información.

- Utilizar la tecnología más reciente en medida de lo posible. Actualmente ya existen en el mercado un gran número de entornos de desarrollo a precios razonables, en distintos lenguajes de programación populares o en su defecto paquetes de dibujo; que proveen de una amplia gama de componentes gráficas de interfaz de usuario, como son los botones, cajas de texto, ventanas, menús, listas, etc., y que ayudan a la realización de un prototipo más rápido. En cuanto a los componentes gráficos hay algunas recomendaciones que es conveniente seguir:
    - Organizar en marcos los elementos comunes que se encuentren en una misma ventana. SI la información que se requiere desplegar ocupa demasiado espacio como para ponerla en marcos, es buena práctica el utilizar objetos tab o pestañas.
    - Alinear adecuadamente. Los objetos gráficos deben estar alineados con respecto a la ventana en la cual se encuentran inmersos, si son de igual tamaño es conveniente alinearlos a la izquierda en otro caso a la derecha.
    - Compatibilidad en los datos. El formato de los datos de entrada debe ser compatible con los de salida, por ejemplo si en la captura de cifras monetarias se solita el uso de dos decimales, debe mostrarse dos decimales cuando se visualizan datos monetarios.
    - Espaciar pertinentemente. Un adecuado espaciado entre los objetos gráficos da una mejor sensación de distribución y organización, por lo cual no debe ser muy grande, ni tampoco que los objetos den la impresión de estar encimados.
    - Usar adecuadamente los tipos y tamaños de letra. Para comandos de la barra de menús o en etiquetas se recomienda que el uso de negrita. Se debe procurar el uso de

tipografías estándar para evitar el distorsionamiento o pérdida de claridad del texto en resoluciones pequeñas y sobre todo no usar más de dos tipos de fuentes en una ventana para mantener la claridad.

- Aplicar botones de opciones, cuando la decisión de un valor deba ser excluyente y que el número de opciones sea pequeño y fijo, de lo contrario utilice listas.
- Utilizar botones de chequeo cuando se pueda y deba elegir más opciones en una decisión.
- Los nombre de botone, etiquetas, comandos etc., deben utilizar palabras sencillas y conocidas por el lenguaje del usuario, comenzando con mayúsculas.

2.2. Diseño de la lógica

2.2.1 Clases y Objetos

Diagrama de clases.

El diagrama de clases es utilizado para representar de manera gráfica los objetos que estarán presentes en un sistema así como aquellos que de manera conceptual han sido observados en el dominio. Está conformado por un rectángulo que se divide en tres rectángulos internos (dependiendo de la perspectiva). En el primero se coloca el nombre de la clase, en el segundo inmediato abajo del primero, se coloca un listado de los atributos que pueden ser simples o complejos y en el cual se puede representar aparte del nombre de los atributos, el tipo de dato, la multiplicidad e incluso un valor inicial. En el último se coloca un listado de los métodos que puede incluir a parte del nombre una lista de los parámetros indicando su nombre y tipo de dato, y en caso de que así sea el tipo de dato de retorno del método.

También se representa la visibilidad tanto de atributos como de métodos anteponiendo al nombre del atributo y/o método los símbolos “+” que indicará que el atributo o método posee visibilidad pública, es decir, que puede ser accedido por las demás clases, “#” indicará que es un atributo o método protegido en cuyo caso sólo puede ser accedido por las clases del mismo paquete o sus clases derivadas y “-“ que significará que el atributo o método sólo puede ser accedido desde la misma clase.

Haciendo uso de un grupo de ellos pueden representarse también las asociaciones presentes entre

las clases, incluyendo la agregación y composición; así como, conceptos como herencia, generalización y especialización.

Un diagrama de clases como ya se había dicho va a ser representado como un rectángulo, sin embargo tiene tres perspectivas de representación que son:

Perspectiva Conceptual, en la que sólo se utiliza un simple rectángulo con el nombre de la clase dentro del él y representa únicamente los conceptos dentro del dominio del sistema que se quiere desarrollar. La figura 51 muestra esta figura.

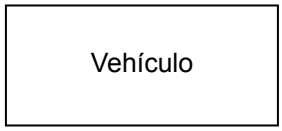


Figura 51. Diagrama de clase desde una perspectiva conceptual

Perspectiva de especificación, en la que el rectángulo es divido en tres partes para representar aparte del nombre de la clase los nombres de los atributos y los nombres de los métodos que conforman a la clase. La figura 52 muestra este tipo de diagrama

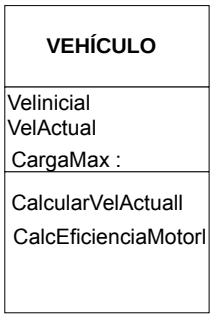


Figura 52 Diagrama de clase desde una perspectiva de especificación

Perspectiva de implementación, en la que el diagrama de clase incluye la visibilidad de los atributos así como parámetros y tipos de datos. La figura 53 muestra el diagrama de clases desde esta perspectiva.

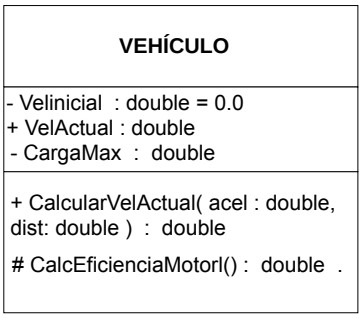


Figura 53 Diagrama de clase desde una perspectiva de implementación

Clase Asociación.

La clase asociación es un tipo especial de clase ya que sólo existe a través de una asociación de clases, pero permitiendo el añadir, tantos atributos como métodos a dicha asociación. La figura 54 muestra un ejemplo de este tipo de clase

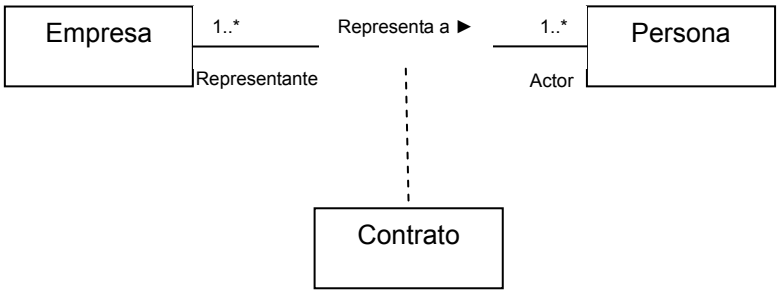


Figura 54 Ejemplo de Clase Asociación.

Asociaciones

Una asociación es un enlace existente entre dos clases o más, es una relación que puede ser reflexiva, binaria, ternaria o n-aria. El tipo de relación más común es bidireccional y toda asociación existe de manera funcional en el mundo real.

Un ejemplo de una relación binaria del mundo real es el de un doctor y paciente ya que en esta

relación, el doctor atiende a dicho paciente, así la relación tiene un nombre que en este caso sería “atiende a” si se ve con dirección del doctor al paciente o en su defecto si se viese de paciente a doctor entonces la relación diría “es atendido por”.

Las asociaciones tienen, por así decirlo sus atributos que son:

- Nombre de la asociación, es decir, frase o verbo que indica el rol.
- Una línea de conexión: No es más que un camino que muestra la liga o asociación existente entre dos clases al final de dicho camino se encuentra lo que en UML se conoce como rol
- Rol: que no es otra cosa más que el papel o rol que juega cada clase en la asociación, es una descripción de dicho papel, algunas ocasiones podría ser el nombre de la clase y a nivel de implementación puede ser útil para utilizarlo para nombrar referencias a clases. Dentro de este rol se pueden indicar o no las propiedades de la asociación que serían:
- La semántica de la asociación o nombre del rol.
- Símbolo para indicar clases especiales de asociación como la agregación o composición, en caso de que esta esté presente.
- Multiplicidad: indica el número de objetos o instancias de cada clase, que estarán involucradas en la asociación. Por lo cual se indica en cada extremo de la asociación. Se expresa en te
- Navegabilidad. Indica el sentido de la asociación.
- Restricción. Las restricciones son utilizadas para indicar ya sea el almacenamiento y manipulación de los objetos inmersos en la asociación, o para indicar que una asociación excluyente (xOR) o para indicar que una asociación es un subconjunto de otra, o que una asociación será permanente.

- Calificador, puede o no estar presente, depende si la asociación se da sobre un conjunto de objetos de una clase referente a un atributo en particular

La siguiente figura muestra estos atributos

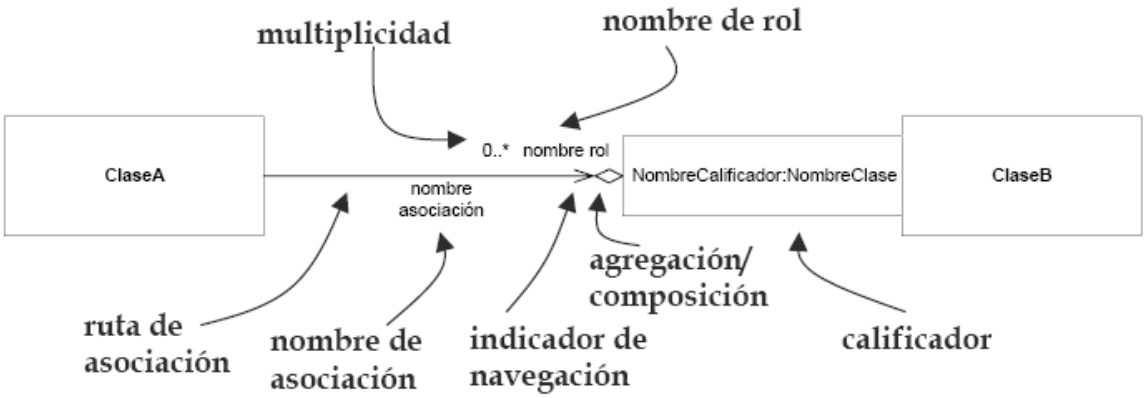


Figura 55. Atributos de una asociación.

Un ejemplo de asociación entre varias clases puede ser observado en la figura 56, como puede verse esta puede servirnos para ver las relaciones de manera conceptual los objetos que están presentes en un sistema, a esto se le conoce como **modelo conceptual**.

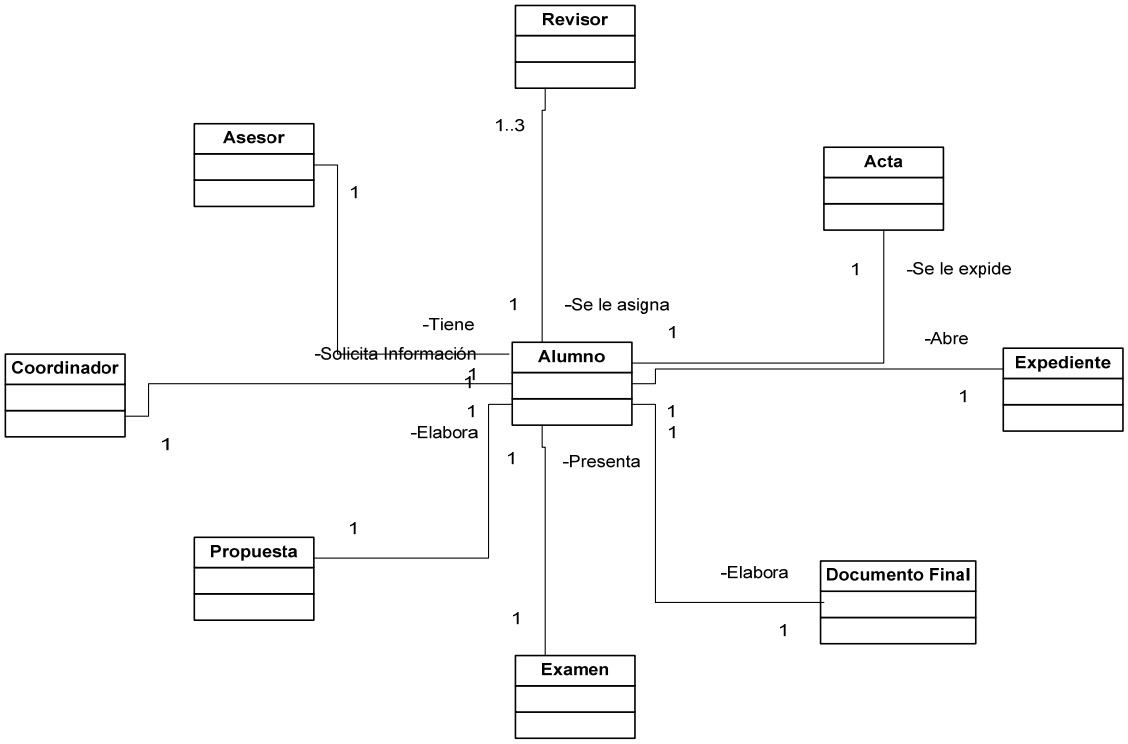


Figura 56. Ejemplo de modelo conceptual

Asociación reflexiva

Es aquella que establece una asociación con ella misma, la siguiente figura muestra este tipo de asociación

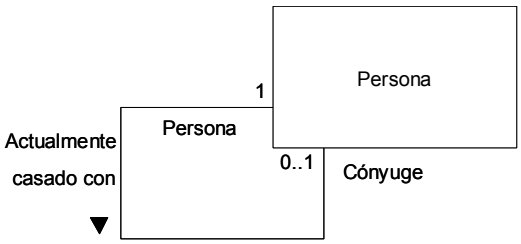


Figura 57. Asociación reflexiva

Agregación y composición.

La agregación es un tipo especial de asociación en donde un objeto se relaciona con otro al formar parte de éste, es decir, es una relación todo-parte. Así un objeto que es agregado es un objeto base que incluye a otros objetos para su funcionamiento.
Sólo puede existir un extremo con agregación en una asociación y no es simétrica, es decir, es decir en el sentido de la asociación sólo una clase será el todo, sin embargo la agregación si es transitiva.

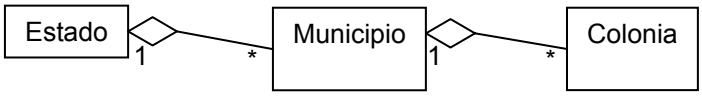


Figura 58. Ejemplo de transitividad en la agregación

La composición es muy parecido a la agregación ya que también se trata de una relación todo-parte, sólo que la diferencia radica en que en la composición si se retiran los objeto de ella, el todo deja de tener significado, lo mismo sucede a los objeto retirados, si se destruye el todo. La composición está relacionada con el valor del objeto, es decir el objeto que lo contiene se encarga de su ciclo de vida (inicialización, mantenimiento y anulación)

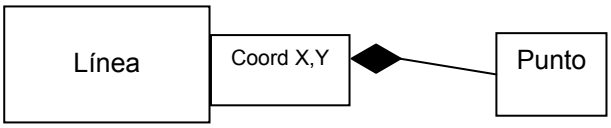


Figura 59. Ejemplo de transitividad en la agregación

Generalización y especialización.

La generalización es una relación que existe entre una clase general y otra más específica, que prácticamente deriva de la primera (a través del mecanismo de herencia) algunas características y

métodos de ella y que contiene nuevos atributos y nuevos métodos, es por lo tanto una especialización de la primera.

En este tipo de relación la clase general se llama padre mientras que la derivada se llama hijo., aunque también es factible que un hijo tenga más de un padre. Así como un padre tenga más de un hijo.
Se caracteriza por ser transitiva y no simétrica.

Es representada por una línea que sale de la clase hijo y que en el extremo de la clase padre tiene un triangulo sin relleno. La siguiente figura muestra un ejemplo de generalización/especialización

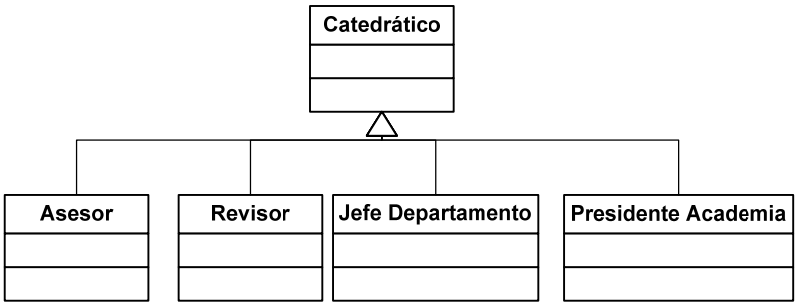


Figura 60. Ejemplo de Generalización/especialización

En la gráfica de una generalización se puede también tener un discriminador si este es el caso a través del cual se puede catalogar los hijos de un padre, la figura muestra la representación de una generalización y el uso de discriminadores 61.

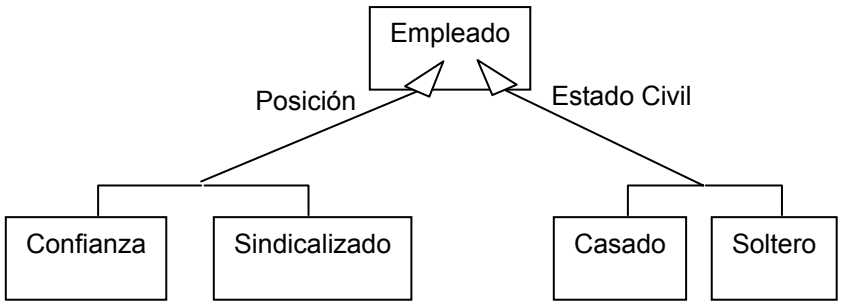


Figura 61. Ejemplo de discriminadores

UML hace uso de restricciones que para ampliar el significado de algún elemento , es decir, su semántica, puesto que a través de las restricciones se pueden anexar o modificar reglas. Las

restricciones en los diagramas de UML son representadas en forma escrita, rodeadas de ambos lados, por llaves "{" puede ser una sola restricción o una lista separadas por coma y son colocadas cerca del elemento al cual se quiere aplicar la restricción,

En la especialización las clases hijas pueden tener restricciones de significado como las siguientes:

Disyunción (Disjoint) Donde no puede surgir una nueva clase hija a partir de dos o más clases (herencia múltiple) que son clases disjuntas entre sí.
La totalidad (Complete) Todas las posibles clases hijas que se pueden derivar de la clase padre han sido especificadas y no puede ser agregada ninguna más.
Un ejemplo de ambas restricciones puede verse en la siguiente figura. Existe disyunción

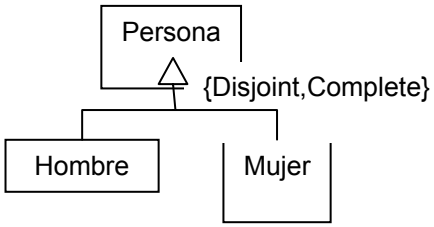


Figura 62. Ejemplo de restricciones de UML en generalización

Parcialidad (Incomplete - incompleto) – Las clases hijas no son todas las que se pueden crear a partir de la clase padre, se esperan más o no han sido especificadas hasta el momento.

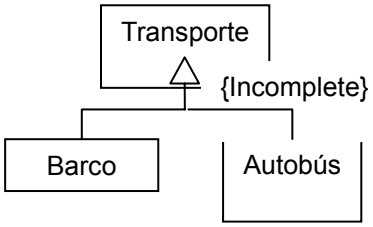


Figura 63. Ejemplo de la restricción "Incomplete"

Solapamiento (Overlapping) Lo opuesto de la disyunción, donde puede surgir una nueva clase hija a partir de dos o más clases (herencia múltiple).

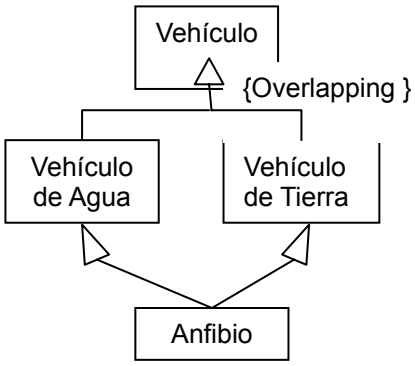


Figura 64. Ejemplo de la restricción “Overlapping”

Diagrama de Objetos

Son utilizados para modelar las instancias de las clases. También sirven para la visualización, especificación, construcción y documentación de las instancias y sus relaciones presentes en un sistema. En los diagramas de interacción se utilizan los diagramas de objetos en su nivel o perspectiva conceptual. La figura 65 muestra el diagrama de objetos en su perspectiva de implementación.

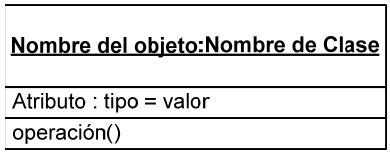


Figura 65. Diagrama de objetos en vista de implementación.

Como ya se había mencionado los objetos en una perspectiva conceptual pueden ser utilizados en los diagramas de interacción en los que si nos referimos a clases en lugar de objetos para tener una vista aún más general, se podrían distinguir tres estereotipos.

1. *Clases Entidad* conocidas como *Clases Dominio*. Representan el núcleo del dominio de la aplicación. Están enfocadas a guardar la información de aquellas entidades persistentes a través del tiempo. Usualmente proveen servicios muy específicos como:

Almacenar y recuperar atributos.
Crear y remover entidades.
Proveer comportamientos que puedan ser necesarios para cambiar de acuerdo
a los cambios que sufran las entidades.

Estas clases por lo común son también utilizadas para derivar la primera vista en la implementación para la Base de Datos física.

2. *Clases Interfase*. Permiten el diálogo entre los actores externos que desean interactuar con la aplicación y las clases dominio. La mayoría de estas clases son componentes de la Interfaz del usuario, las cuales se convertirán en formas y pantallas usadas para interactuar con la aplicación. Para encontrar estas clases se deberá enfocarse en los actores, ya que cada actor su propia interfase en el sistema para interactuar con él.

3. *Clases Control*. Son los coordinadores de las actividades del dominio de la aplicación, razón por la cual se les conoce como controladores, son almacenes de ambos estructuras y comportamientos que no son fáciles de colocar en alguna de los dos tipos de clases anteriores. Típicamente una clase control puede jugar cualquiera de varios roles:

Como un comportamiento en relación a transacciones.
Como un control de secuencia que es específico a uno o algunos casos de uso o rutas a través de un caso de uso.
Como un servicio que hace la separación entre objetos dominio de objetos interfase.

Su ciclo de vida es transitorio, dura el tiempo necesario para completar una interacción.

2.2.2 Interacción

Ya se había dicho que la descripción de los escenarios en las plantillas de casos de uso, sirven para modelar el comportamiento del sistema, está dinámica puede ser representada a su vez de manera gráfica a través de los diagramas de interacción.
Los diagramas de UML que conforman los diagramas de interacción son el diagrama de secuencia y el diagrama de comunicación (colaboración).
El hecho de que los diagramas de interacción son usados como parte del modelado del comportamiento del sistema. No implica que no puedan ser utilizados cuando se desea representar de manera gráfica la implementación del sistema, ya que si estos diagramas representan la dinámica de los casos de uso, hay que recordar que también se cuenta con los casos de uso reales, que como ya se había mencionado con anterioridad, describen una secuencia de acciones o eventos, a partir de un diseño específico sujeto a las tecnologías informáticas que se utilicen.

Diagrama de Secuencia.

Este diagrama es utilizado para mostrar los mensajes en secuencia de tiempo, es decir, exponen la secuencia, duración y comunicación entre objetos que se dan cuando se utiliza el sistema y muestra de una forma gráfica el comportamiento del sistema.

La simbología que se utiliza en este diagrama puede ser observada en la figura 66.

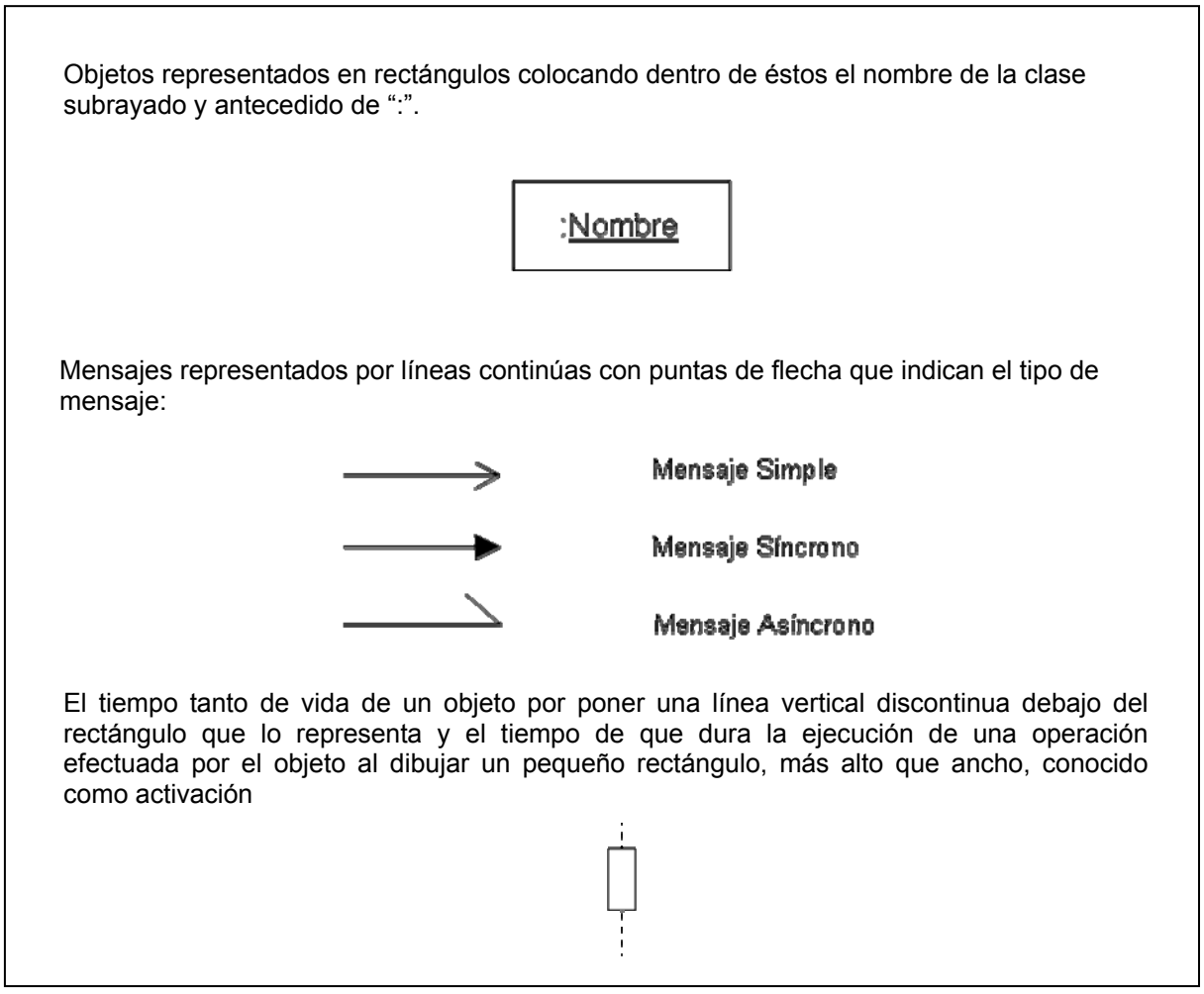


Figura 66. Símbolos usados en un Diagrama de Secuencia.

Los mensajes deben ir acompañados de texto, aunque puede ser opcional en el caso de retorno de valores. Este texto en el mensaje puede ir acompañado de alguna condición e incluso si el diagrama de secuencia esta representación dinámica de un caso de uso real, tener el formato de un método, es decir, el nombre del método, los nombres y tipos de datos de los parámetros y el tipo de dato de retorno del método en caso de que regresa algún valor.

También en los diagramas de secuencia cuando las interacciones son complejas se manejan fragmentos de diagrama los cuales se marcan con un rectángulo a su alrededor y en la esquina superior izquierda de dicho rectángulo se colocan operadores a través de los cuales es posible definir cosas como: ciclos o rupturas en la secuencia o ejecuciones paralelas u opciones y otros más que junto con los ya mencionados son listados a continuación:

alt (Alternativa): Este operador es utilizado para indicar que el fragmento del diagrama contiene alternativas de secuencia, donde sólo se ejecuta la que cumpla con la condición marcada.
opt (Opción): Este operador es utilizado para indicar que el fragmento del diagrama es una opción de secuencia, que se sigue si se cumplió con la condición marcada.

Nota. En el caso de los dos anteriores operadores, sería conveniente antes de emplearlos en un diagrama de secuencia revisar si dichos fragmentos no pueden ser mejor tratados como la representación dinámica de casos de uso incluidos, es decir, hacer diagramas de secuencia independientes.

loop (Bucle): Este operador es utilizado para indicar que el fragmento del diagrama contiene una secuencia de eventos que se ejecuta múltiples veces. La figura a continuación muestra un ejemplo del uso de un fragmento de secuencia que hace uso de este operador

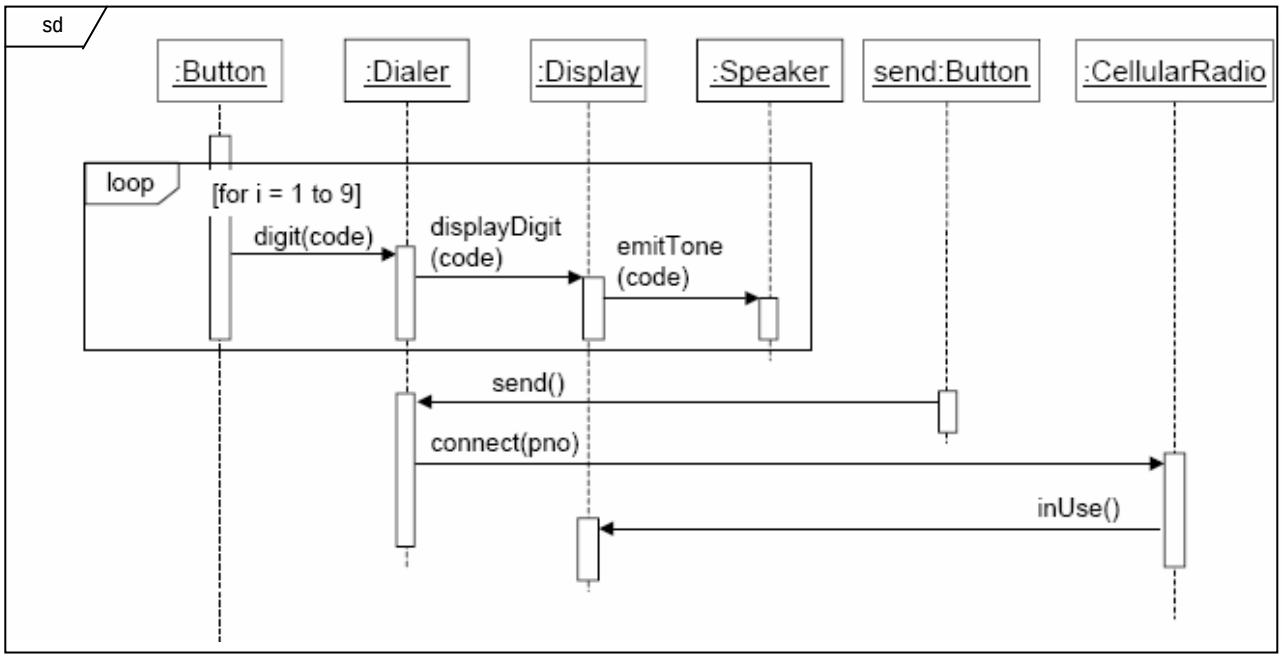


Figura 67. Ejemplo del operador loop en un fragmento de secuencia

neg (Negativa) :Este operador es utilizado para indicar que el fragmento del diagrama contiene una secuencia de eventos resultado de una interacción inválida.

par (Paralelo): Este operador es utilizado para indicar que el fragmento del diagrama puede ser ejecutado por varios hilos de procesamiento de tal forma que resulta que su ejecución sea paralela0

Critical (Región crítica): únicamente un proceso será el que se encuentre ejecutando dicho fragmento es decir, el fragmento debe ser ejecutado de manera simultánea

También en los diagramas de secuencia es posible utilizar los estereotipos, encerrando el nombre del estereotipo entre los símbolos “<<” y “>>”. Como se muestra en la siguiente figura para crear y destruir instancias de los objetos .

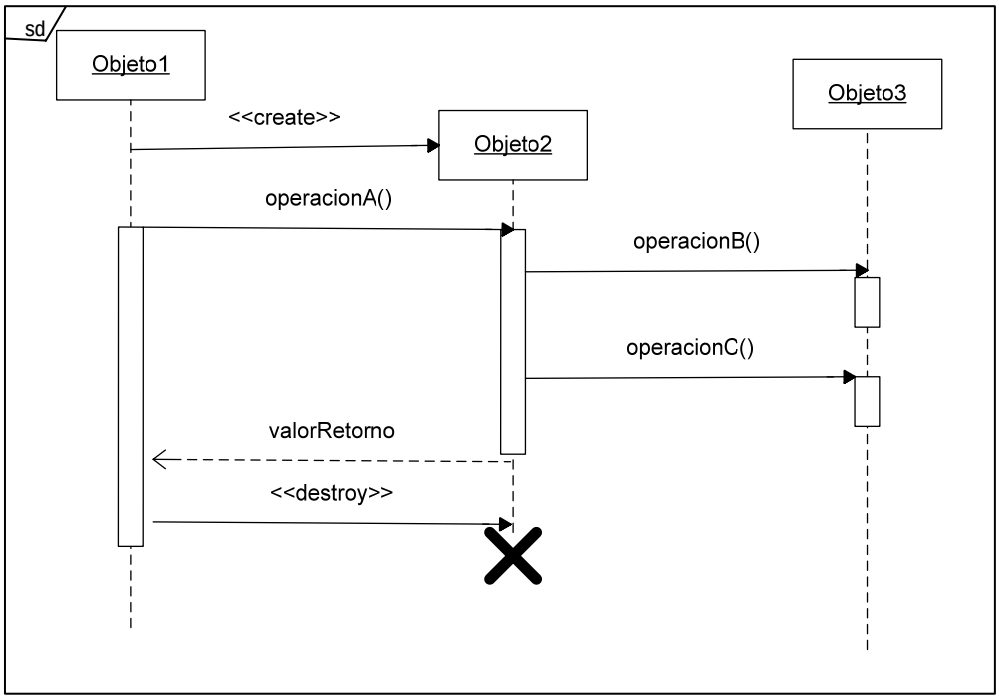
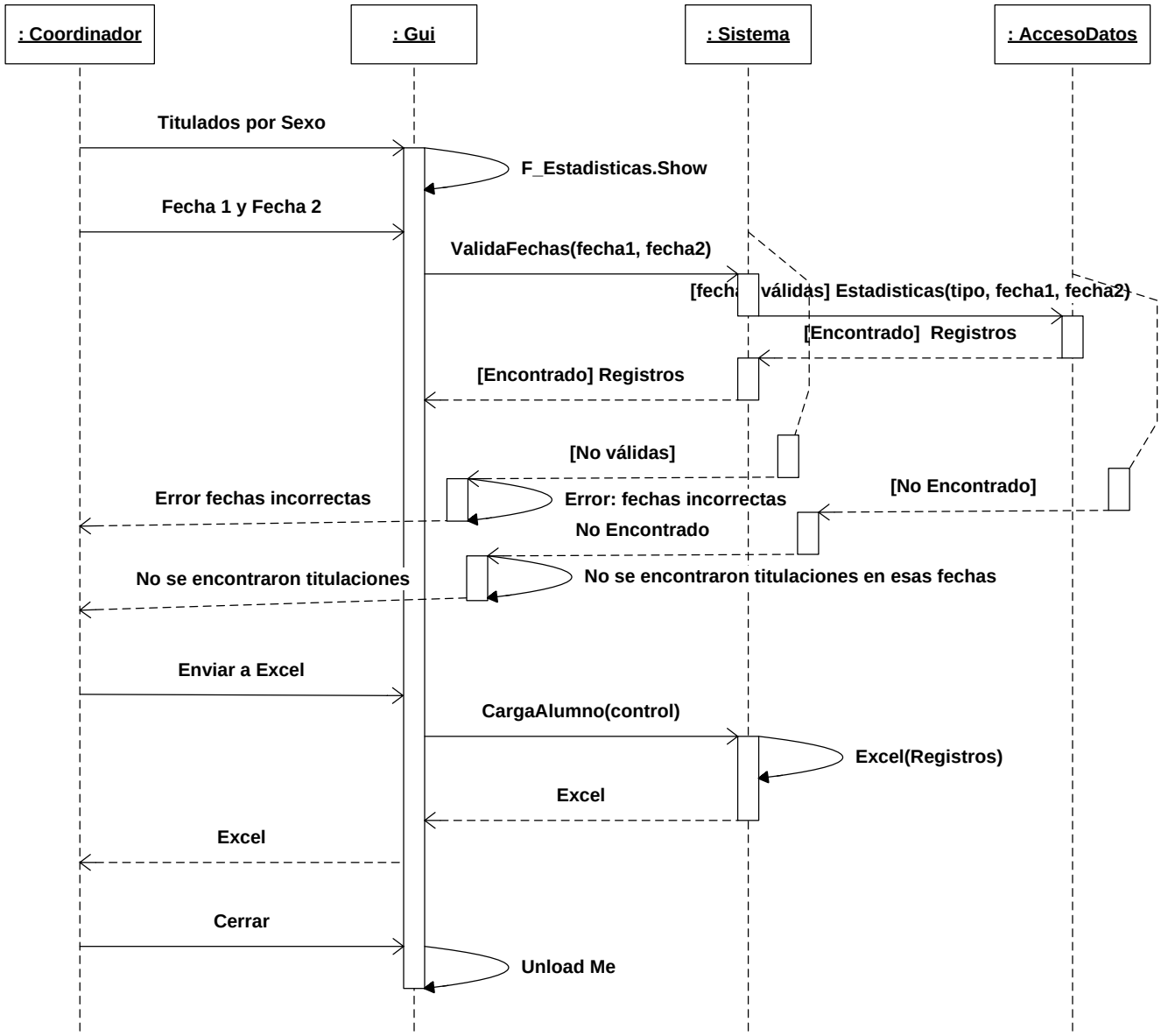


Figura 68. Ejemplo del uso de estereotipos en un diagrama de secuencia

Un ejemplo de diagrama de secuencia al cual le faltan elementos pero cuenta con los suficientes para entender la sucesión en la comunicación entre objetos se muestra en el inciso a de la siguiente figura, que se desprende de la plantilla de caso de uso que puede verse en el inciso b de la misma figura.



a) Diagrama de secuencia

CASO DE USO: Titulados por sexo

ACTORES: Coordinador.

PROPÓSITO: Obtener los datos de los alumnos que se han titulado, clasificados por sexo para poder pasarlos a Excel.

TIPO: Primario Esencial.

RESUMEN: El Coordinador pide al sistema que le muestre los alumnos dentro de un intervalo de fechas.

REFERENCIAS CRUZADAS: ValidaFechas(fecha1, fecha2), Estadisticas(tipo, fecha1, fecha2, gird)

CURSO NORMAL DE EVENTOS.

ACCION DEL ACTOR	RESPUESTA DEL SISTEMA
1. Selecciona la opción “Titulados por sexo” del menú Estadísticas.	
	2. Presenta la pantalla Estadísticas.
3. Selecciona el intervalo de fechas en las que desea buscar a los alumnos titulados.	
4. Hace clic en el botón Buscar	
	5. Manda el mensaje ValidaFechas(fecha1, fecha2), donde se asegura que fecha1 sea menor de fecha2 y que fecha2 no sea mayor a la fecha de hoy.
	6. Si las fechas son correctas, manda el mensaje Estadisticas(tipo, fecha1, fecha2, gris). Que buscará en la tabla propuestas, aquellas que tengan estado 15 y que la fechas de su examen este entre el rango de las fechas mandadas.
	7. Si se encuentran registros, los muestra en pantalla en el control grid.
8. Hace clic en el botón Enviar a Excel.	
	9. Abre Excel y le envía los datos obtenidos con el query.
	10. Muestra el un mensaje indicando que los datos ya se encuentran en Excel.
11. Hace clic en el botón Cerrar.	
	12. Descarga la pantalla Estadísticas.

CURSO ALTERNO.

6. Si las fecha son incorrectas muestra el mensaje de error “Las fechas son incorrectas, favor de verificar”.
7. Si no se encuentra ningún registro, se muestra el mensaje “No se encontró ninguna titulación entre estas fechas”.

b) Plantilla de caso de uso que sirve de base para el diagrama de secuencia

Figura 68. Ejemplo de un diagrama de secuencia obtenido a partir de la plantilla de un caso de uso

Diagrama de Comunicación (colaboración).

Este diagrama es utilizado para mostrar como los mensajes relacionan a los objetos durante una interacción entre ellos, razón por lo cual los mensajes se numeran. Son más finos para describir la interacción entre objetos que los diagramas de secuencia, ya que los mensajes llevan un formato definido que indica claramente dicha interacción, también son mejores que los de secuencia cuando se trata de ilustrar bifurcaciones complejas, iteraciones, así como, compartimientos concurrentes.

La simbología empleada en estos diagramas consta básicamente de 3 elementos:

- Objetos: participantes en la comunicación. Aunque hay que aclarar que alguno de los participantes también puede ser una clase, cuando el mensaje es para invocar un método de una clase en particular.
- Mensajes: son elementos con nombre, para indicar un tipo específico de comunicación en una interacción. La comunicación puede referirse a cosas tales como: invocación de un método, creación o destrucción de una instancia, envío de una señal. El mensaje especificará tanto al emisor como al receptor del mensaje. El orden en el tiempo se organiza de arriba hacia abajo, numerando los mensajes, excepto el primero. Cada mensaje entre objetos es representado como una expresión sobre una línea que tendrá en un extremo una punta de flecha indicando la dirección del mensaje. La sintaxis de dicha expresión es la siguiente:

  <identificador del mensaje> : [condición :] | [<variable> '='] <señal o nombre de la operación> ['(' [<argumento> [','<argumento>] ')'] [':' <valor de retorno>] | ' <argumento> ::= ([<nombre_arg> '='] <valor_arg>) | (<atributo> '=' <nombre del parámetro de salida> [':' <valor_par>]) | ' -'*

  Donde:
  [ ] representa algo opcional
  | representa en lugar de
  iteración.
  condición está formada por [variable signo de comparación valor|variable]
  signo de comparación se cualquiera de los siguientes =, <=, >=, <>, <,>

  Ejemplos de esta sintaxis:

  2: consulta(1,-,2,"precio") , el identificador del mensaje es dos y en este caso el

segundo argumento es indefinido

1.1: num=message(5,kim):200 , el identificador indica que el mensaje cuyo identificador es “1”; tiene caminos que son paralelos Y7o que son mutuamente exclusivos siendo éste “1.1” (uno de uno) el primero de ellos y en este caso el mensaje recibe un valor de retorno (200) que almacena en la variable num

3: [color = yellow] : amarillo(camino) , el identificador del mensaje es 3 y los paréntesis cuadrados indican condición en este caso que el valor contenido en la variable color sea igual al valor contenido en la variable yellow, de ser así se ejecuta la invocación al método amarillo del objeto que recibe el mensaje y cuyo método tiene un solo parámetro en llamado camino , en este caso no se indica el tipo de dato del parámetro.

1: *[i=1..n]:num=nextInt ,el indicador del mensaje es 1, el asterisco indica que la llamada al método nextInt, cuy valor de retorno será depositado en la variable num se repetirá en un ciclo “n” número de veces comenzando por el “1”

1:* [i:=1..10] siguienteLineaProducto(clave) ,el indicador del mensaje es 1, el asterisco indica que la llamada al método siguienteLíneaProducto, cuyo parámetro se llama clave y que no se indica el tipo de valor, será invocado 10 veces.

1:* subTot=CalculaSubTotal , el indicador del mensaje es 1, el asterisco indica que el valor de la variable subTot será el retornado por el método CalculaSubTotal para cada elemento en un conjunto o clección de miembros

También los mensajes pueden contener los estereotipos <<create>> y <<new>> o <<destroy>>, para indicar la creación de instancias o su destrucción.

- Ligas. Consta de una flecha que conecta a dos objetos, o a un objeto y a una clase, indicando la dirección de la comunicación, es decir, quién es el emisor inicio de la flecha sin punto y quién es el receptor punta de la flecha. Y además puede o no indica también la visibilidad. El tipo de flecha indicará si el mensaje es asíncrono, una flecha cuya punta es sencilla o

síncrono una flecha con punta sólida.

En la figura 70 se muestra un ejemplo de un diagrama de comunicación.

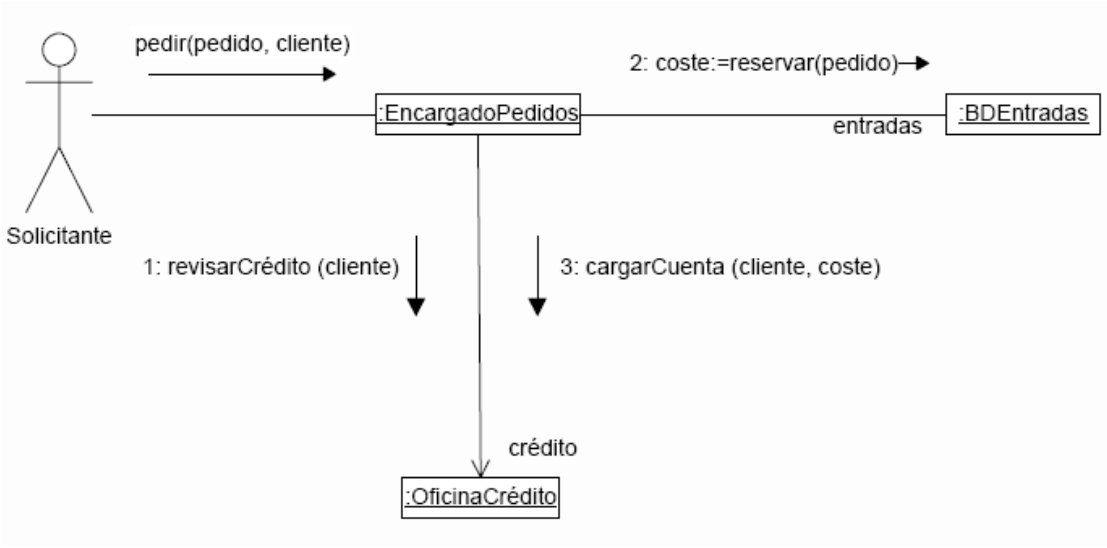


Figura 70. Ejemplo de un diagrama de comunicación.

La siguiente figura muestra un ejemplo de un diagrama de colaboración que incluye un mensaje condicional y el uso de los estereotipos <<create>> y <<new>>n

Figura 70. Ejemplo de diagrama de comunicación con estereotipo y condición.

La siguiente figura muestra un diagrama de comunicación que contiene una iteración para una colección.

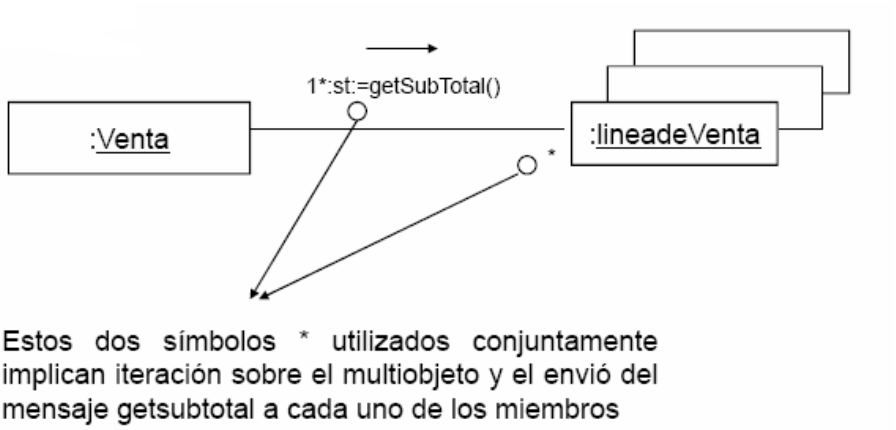


Figura 71. Ejemplo de un diagrama de comunicación con una colección.

En la siguiente figura pueden verse un diagrama de comunicación con mensajes mutuamente exclusivos

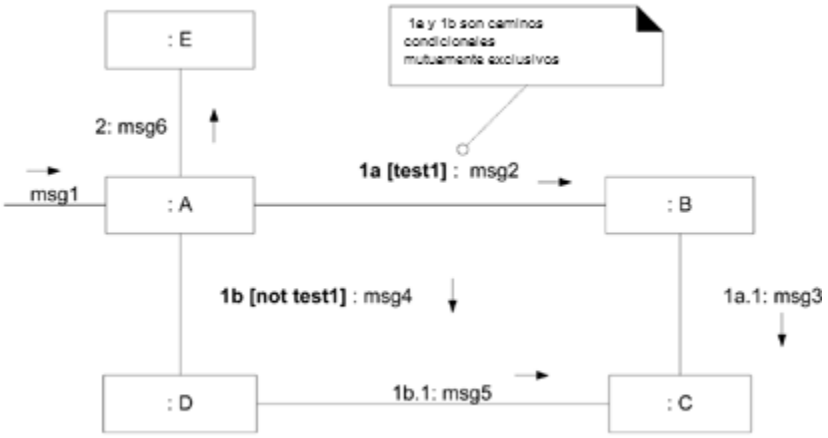


Figura 72. Diagrama de comunicación con mensajes mutuamente exclusivos

En la figura a continuación puede observarse un diagrama de comunicación con un mensaje a una clase

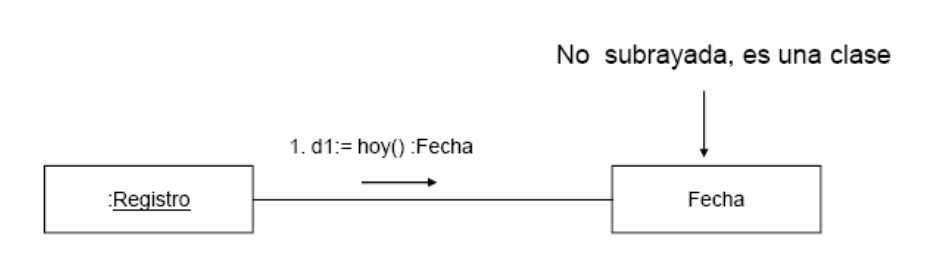


Figura 73. Diagrama de comunicación con mensaje de objeto a clase.

En la figura que sigue se puede ver un diagrama de comunicación en el que se muestra un mensaje asíncrono.

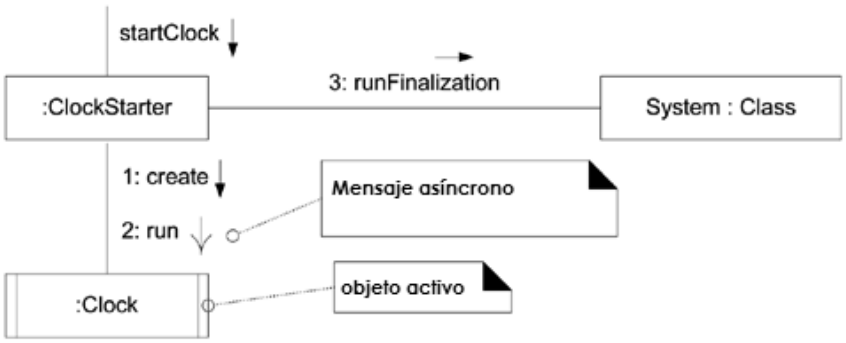


Figura 74. Ejemplo de diagrama de comunicación con mensaje asíncrono.

Y por último la siguiente figura muestra un diagrama de comunicación con una numeración más compleja y que incluye a un actor.

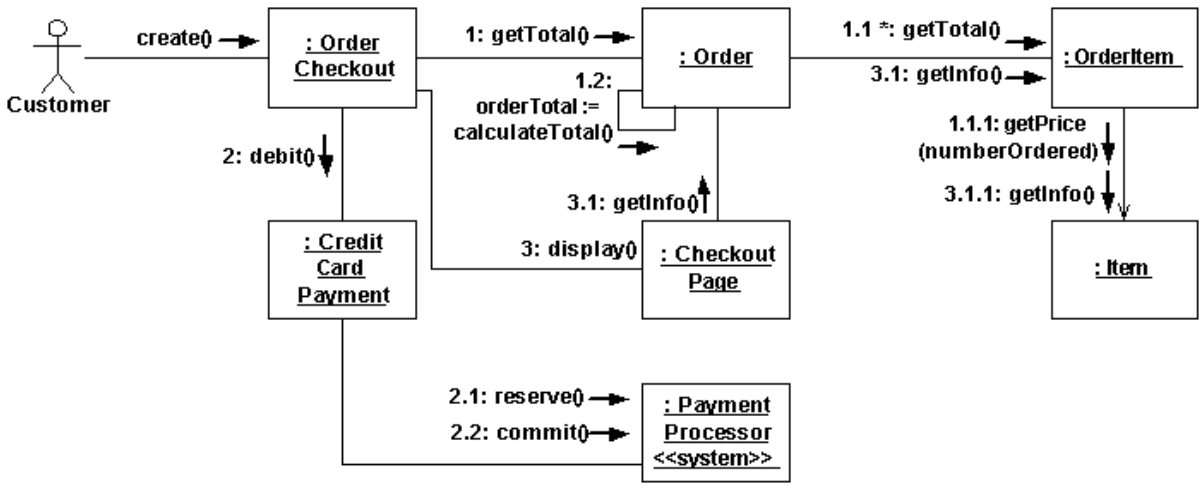


Figura 75. Diagrama de comunicación que incluye a un actor.

2.2.3 Estados y transiciones

En muchas aplicaciones es de suma importancia el considerar los estados y las transiciones entre los mismos de un objeto individual, estas aplicaciones varían desde un juego de computadora hasta aplicaciones como la domótica. Las máquinas de estados finitos conocidas como FSM (Finite Machine State), han sido desde hace mucho tiempo ampliamente usadas para un diverso número de propósitos, puesto que básicamente pueden representar el comportamiento ya sea de un sistema, o de un objeto, que recibe señales de entrada y cuyas señales de salida dependerán de todos los estados de dichas señales de entrada. Las máquinas de estado finitos está conformado por 3 elemneto: estado, transiciones y eventos. Son precisamente estos estados, sus transiciones y los eventos que las provocan los que son modelados a través de un Diagrama de estados.

Diagrama de Estados.

Un diagrama de estados es muy útil para representar objetos complejos y sobre todo facilita su programación, ya que representan su ciclo de vida interno. Provee una vista dinámica que permite modelar el comportamiento manejado por eventos, como el que sucede en una interfaz gráfica de usuario.
EL diagrama de estados puede contener los siguientes elementos, estados, transiciones, eventos y actividades

- Estados: Describe un instante de tiempo durante la vida de un objeto o de una clase

  Los objetos tienen una serie de estados a lo largo de su ciclo vida, en los cuales permanecen un tiempo finito.

  Los estados se pueden reconocer por sus acciones o pueden tener nombre.

  El símbolo de un estado es un cuadro con las esquinas redondeadas, que puede contener el nombre del estado el cual debe ser único aunque se puede omitir dando como resultado un estado anónimo, y puede estar divido en tres partes como el diagrama de clases, esto depende de cómo quiera representarse si es función de sus atributos, es decir estáticamente, por lo regular sólo se usa un rectángulo redondeado, o si es en función de las actividades que ejecuta, es decir dinámicamente, se utiliza un cuadrado redondeado dividido en tres compartimientos en donde:

  - El primer compartimiento llamado compartimiento nombre, valga la redundancia

se pone el nombre del estado, si se trata de un subestado, entonces su nombre debe ir seguido de dos puntos y luego el nombre del estado al que el subestado pertenece.

<nombre del subestado>:<nombre del estado>

- El segundo, llamado compartimiento de actividades internas, en el cual se describen las acciones producidas en el estado, que pueden ser :
  - De entrada (entry): acciones que se realizan cuando el objeto entra a el estado
  - De salida (exit): acciones que se realizan cuando el objeto sale del estado
  - De hacer (do): actividades que se llevan a cabo mientras el objeto se encuentra en el estado.
- El tercero, llamado compartimiento de transiciones internas que contiene una lista de los eventos que dado el resultado de una función booleana serán ejecutados a través de una acción.

La figura 76 muestra estos dos tipos de símbolos

a) Estado en su vista estática b) Estado en su vista dinámica

Figura 76. Representación gráfica de un estado

Los estados a su vez pueden ser compuestos, es decir poseer una subestructura anidada, la cual está conformada por una red de estados que pueden ser concurrentes o disjuntos, llamados subestados, los cuales heredan las variables de estado y las transiciones externas. Un ejemplo de estado compuesto puede ser visto en la siguiente figura.

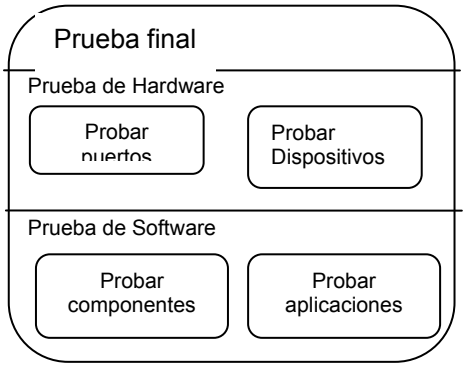


Figura 77. Representación gráfica de un estado compuesto con subestados concurrentes

Existen dos estados en particular cuya representación gráfica no es un rectángulo redondeado, estos son: el estado inicial y el estado final cuyas figuras pueden ser vistas en el inciso a y el inciso b de la figura 78 respectivamente. El propósito de ambos estados es únicamente representar gráficamente el inicio y el fin de un flujo de transiciones a través de uno o más estados. Ambos pertenecen a la categoría de "seudo estados"

a) Estado Inicial b) Estado final

Figura 78. Estados inicial y final

Otros seudo estados son: Fork/Join, Junction, Choice, Entry/Exit Point, Terminate, DeepHistory/shallow History.

Fork. Es para representar un vértice que divide una transición de entrada en dos o más transiciones de salida hacia diferentes regiones de un estado compuesto.
Join: Es para representar un vértice que mezcla un conjunto de transiciones salidas por lo regular de diferentes regiones de un estado compuesto y que no cuentan con eventos disparadores, ni condiciones de disparo, para dar como resultado una sola

transición.

Junction. Es usado para representar un vértice que mezcla un conjunto de transiciones con eventos disparadores y condiciones, para dar como resultado otro conjunto de transiciones diferentes o una sola transición.

Choice. Es un vértice en forma de rombo, que se usa para representa la gama de transiciones de salida que tendrán lugar dado el valor de la condición que tiene una transición de entrada.

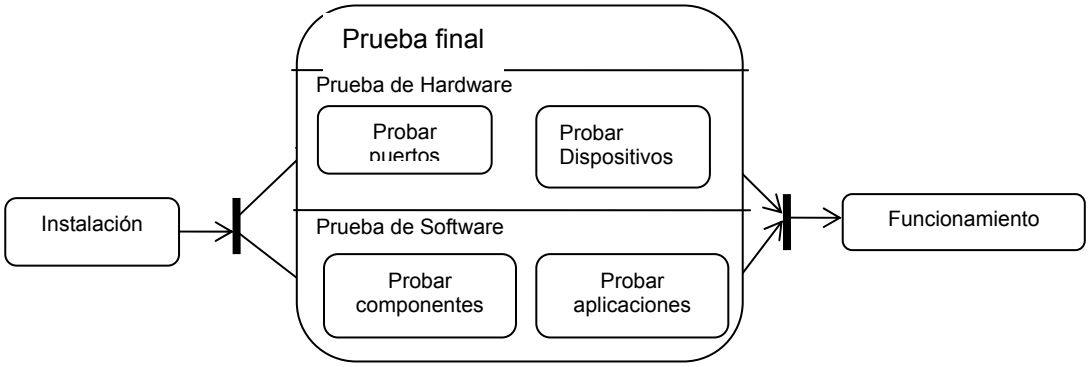
Punto de entrada. Es usado para señalar el punto de entrada de una transición en particular dentro de una región de un estado compuesto.

Punto de salida. Es usado para señalar el punto de salida de una transición en particular dentro de una región de un estado compuesto.

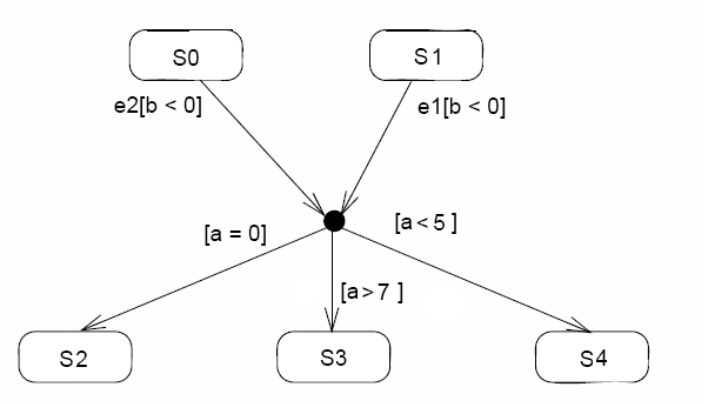
Terminate. Este seudo estado sirve por lo regular para indicar la terminación no normal de un objeto o como su destrucción.

DeepHistory/shallow History. Ambos son muy útiles cuando es necesario saber cual serían los subestados anteriores de un objeto, justo antes de que se interrumpiera su ejecución o se pasará a otro estado. El primero guarda la historia de los subestados de un subestado,es decir, profundiza dentro de regiones complejas en lugar de quedarse en un nivel superior de supestados, mientras que el segundo solamente resume el último estado que tenía, dentro de una región compleja.

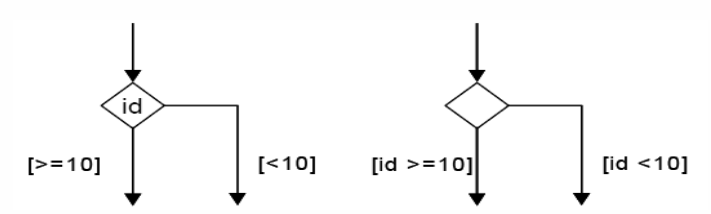
Las representaciones de estos seudo estados pueden ser vistas en la figura 79



a) Fork y Join



b) Junction



c) Choice

d) Punto de entrada y de salida y terminate

f) Deep y Shallow history

Figura 79. Seudo estados

- Transiciones: Son las responsables de desencadenar los cambios de estado en un objeto. También representan las relaciones existentes entre los estados de un objeto, en ellas se muestra el flujo de control.

  Por lo que una transición puede estar conformada por un evento disparador, una función booleana o condición, llamada guarda y una acción de acuerdo al siguiente formato.

  <disparador>[guarda]/acción

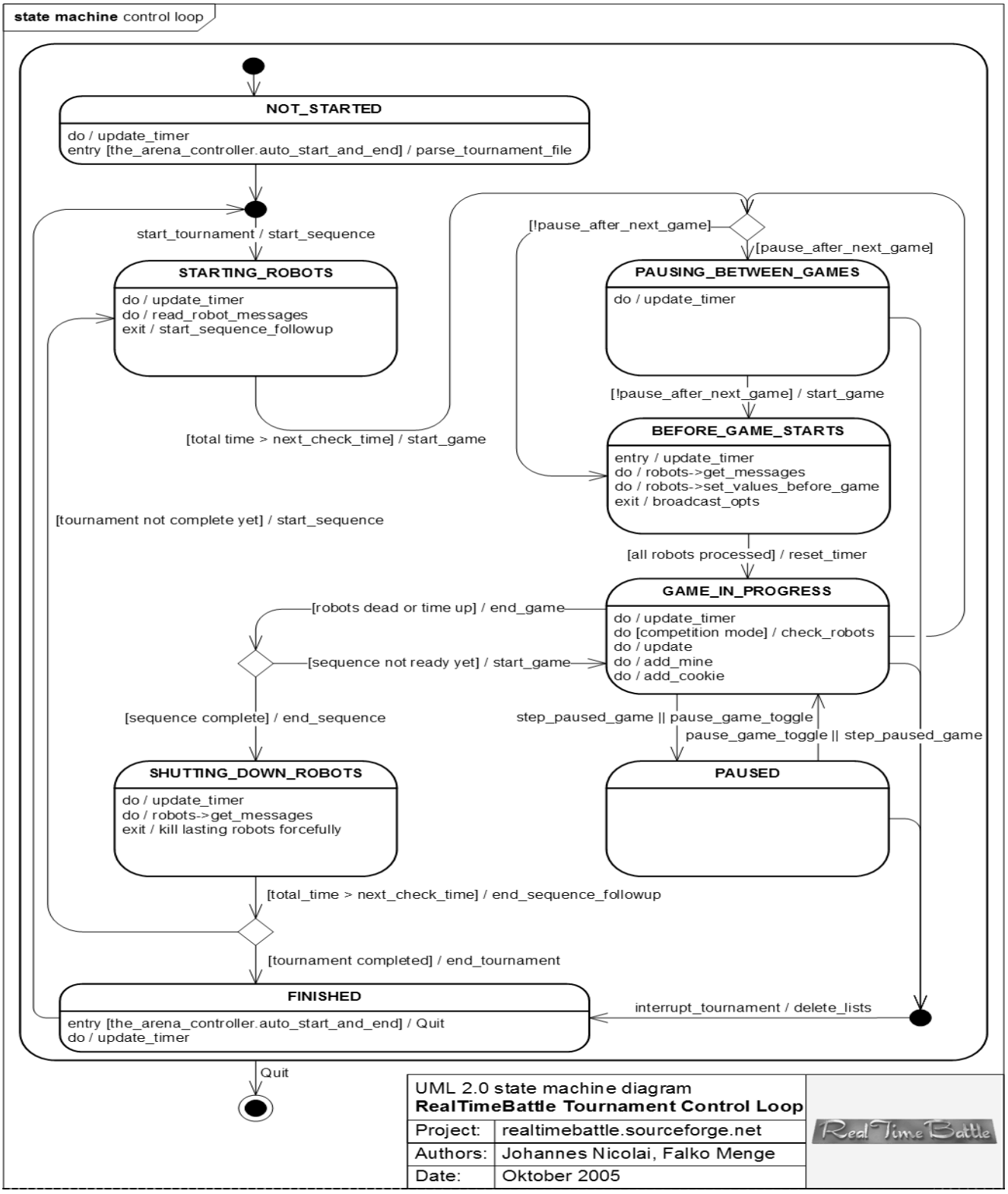
  Donde

  Disparador: es un evento que dispara si se cumple la guarda la transición de un estado origen a otro estado destino Puede poseer parámetros, cuyos valores deben estar disponibles en la transición, y también pueden ser utilizados tanto en las expresiones de la guarda, como en las acciones.

  Guarda. Es una expresión lógica cuyo resultado dispara o no la transición provocada por el evento, si la guarda no contiene operadores relacionales, se considera que la variable implícita en la expresión, es verdadera.

  Acción: por lo regular representa la llamada a un método, el cual puede o no contener parámetros.
- Eventos. Un evento es una ocurrencia instantánea que no tiene duración. Un evento puede pertenecer a alguno de los siguientes tipos:
  - Eventos síncronos, como:
    - Evento de llamada. El cual es el resultado de una invocación explícita a un método del objeto.
    - Evento de excepción. Es el resultado del tratamiento de un error.
  - Evento asíncronos, como:
    - Evento de señal. Es el resultado de una comunicación asíncrona, cuyos parámetros, en caso de poseerlos, son más bien atributos de la señal. En este tipo de evento el estado u objeto que envía la señal continúa con sus actividades.
    - Evento de cambio. Es el que se produce al obtener con valor de verdadero una expresión lógica.
    - Evento de tiempo. Es aquel producido al finalizar un contador.
- Actividades. No son otra cosa que las acciones que serán llevadas a cabo como resultado de una transición de estados.

La siguiente figura muestra un ejemplo de un diagrama de estados elaborado por Johanes Nicolai y Falko Menge en el modelado del marco de trabajo RTB-Team que sirve para programar robots en el paquete de juego RealTimeBattle



La figura 80. Ejemplo de diagrama de estado

Preguntas

1. ¿En que consiste el modelado de negocio?
2. ¿Dentro de un diagrama de actividad que significa un punto de decisión?
3. ¿En que consiste un diagrama de actividades con marco de responsabilidad?
4. ¿Qué son los tipos de acción en un diagrama de actividad?
5. ¿De acuerdo a Ivar Jacobson, qué es un caso de uso?
6. ¿Qué significa un actor para los diagramas de caso de uso?
7. ¿Cuál es la diferencia entre un caso de uso esencial y un caso de uso real?
8. ¿Qué es un escenario secundario?
9. ¿Qué es utilizado para mostrar la secuencia de interacción que existe dentro de un caso de uso entre el usuario y el sistema?
10. Defina que es una interfaz gráfica de usuario (GUI).
11. ¿Cuáles son los factores cognitivos involucrados en la comunicación existente entre un usurio humano y un sistema a través de una GUI?
12. ¿Qué es la perspectiva específica dentro de los diagramas de clases?
13. Cuando se habla de generalización, ¿qué representa la disyunción?
14. ¿Qué es una clase entidad?
15. ¿Qué elementos conforman un diagrama de comunicación?
16. ¿Cuándo debe ser usado un diagrama de estados?
17. ¿ Qué es un DeepHistory/shallow History?
18. Describa de manera general lo elementos que conforman un diagrama de estados.

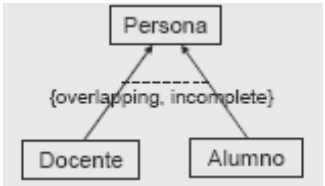
Ejercicios

1. Consiga una herramienta para el modelado que utilice diagramas UML.
2. Considere que usted tiene la tara de realizar la revisión del software instalado en las computadoras de su ámbito o empresa, y debe realizar un reporte del mismo, resaltando detalles como la descripción del equipo, la dirección IP asignada al mismo , el departamento al que pertenece, la persona que está a cargo del equipo, la fecha de último mantenimiento preventivo realizada al equipo para que en caso de tener más de 30 días hacer un reporte correspondiente para su realización, si el software instalado está autorizado, en proceso o no. Realice un diagrama de actividad que modele los pasos realizados al elaborar esta tarea. Utilice la

herramienta de software para modelar que haya conseguido (Racional Rose, StarUML, etc.,)

3. Considere usted que es un Amo o Ama de Casa y que debe pagar las cuentas que genera el llevar un hogar, como renta, luz, etc., Haga un diagrama de casos de uso que muestre el concepto de generalización y especialización. Utilice la herramienta de software para modelar que haya conseguido.
4. Considere que se encuentra como empleado en una tintorería y su trabajo consiste en planchar las prendas de vestir, ilustre haciendo uso de los diagramas de caso de uso los conceptos de extensión e inclusión. Pregunte a su mamá, esposa o cualquier persona experimentada en este menester. Recuerde que las prendas de vestir tienen exigencias de planchado específicas como las camisas de vestir. Utilice la herramienta de software para modelar que haya conseguido.
5. Chang Ling debe contraer matrimonio siguiendo sus tradiciones, para lo cual sus padres le han enviado información de un grupo de mujeres que son hijas de conocidos de ellos y las cuales cumplen completamente con las normas morales que sus padres consideran adecuadas. Chang Ling ha decidido desarrollar un sistema que le permita el procesamiento de la información y le ayude en la toma de decisiones. Realice un diagrama que incluya todos los casos de uso que usted considere necesario para este sistema. Utilice la herramienta de software para modelar que haya conseguido.
6. Tomando en cuenta el ejercicio anterior considere el caso de uso de alta de expediente de candidata, y desarrolle una plantilla de caso de uso considerando tanto ruta normal como alternas.
7. Considere el ejercicio 5 y diseñe la interfaz gráfica de usuario para el caso de uso planteado en el ejercicio 6. Y en caso de ser necesario corrija el ejercicio 6.
8. El siguiente diagrama es incorrecto diga en que y porque

9. Sugiera una clase hija de la siguiente figura



10. Considere un consultorio médico y exprese de manera gráfica las clases doctor y paciente y sus atributo de multiplicidad.

11. Tome en cuenta las posibles clases asociaciones existentes ente una agenda de citas de un dentista.
12. Represente de manera gráfica una asociación reflexiva considerando un automóvil y sus pasajeros.
13. Investigue cuando son utilizados los mensajes del tipo síncrono en los diagramas de secuencia
14. Tomando en cuenta el ejercicio 6. Haga el diagrama de secuencia de cada escenario. Utilice la herramienta de software para modelar que haya conseguido.
15. Considere el funcionamiento de una batería eléctrica de un automóvil y dibuje el o los diagramas de estado correspondientes. Utilice la herramienta de software para modelar que haya conseguido.

Lecturas Recomendadas

Bertrand Meyer. “Construcción De Software Orientado A Objetos”. 1999, Editorial Prentice.

Mark Priestley. “Practical Object-Oriented Design With Uml”, 2000, Editorial Mc Graw Hill.

Perdita Stevens. “Utilización De Uml En Ingeniería De Software Con Objetos Y Componentes”. 1999, Addison Wesley.

3. Construcción.

Como un paso previo a la construcción y durante ella, es necesario tener un modelo de cómo el software será distribuido a lo largo de los dispositivos físicos. Para ello se puede hacer uso de los diagramas de UML de despliegue y de componentes los cuales serán tratados en la siguiente sección.

Por otra parte como ya se había mencionado anteriormente, antes de ni siquiera comenzar a hacer un bosquejo del sistema , debe tenerse en cuenta el ámbito en el cual éste será desarrollado, para poder así establecer el tipo de sistema que debe ser desarrollado y con ello elegir la metodología, técnicas y herramientas más adecuadas; ya que no es lo mismo hacer un sistema que se encargue de transacciones bancarias, que el sistema de control de citas de un dentista, o un juego para un dispositivo móvil, que un sistema que deba controlar el funcionamiento de un robot espacial.
Por lo debe quedar claro que todo sistema computacional, desde el que sólo cuenta con un usuario hasta aquel que tiene usuarios a nivel mundial, posee una arquitectura específica con diferente calidad.
Es esta arquitectura, la que conlleva a tomar las decisiones principales de diseño que se hacen sobre un sistema, las cuales son pensadas en cómo el sistema funcionará y por lo tanto el cómo debe ser desarrollado. Estas decisiones involucran aspectos como la estructura, funcionalidad y usabilidad que garanticen la calidad del sistema, lo cual implica que sólo aquellos sistemas que emplearon una arquitectura adecuada al ámbito del problema y de manera correcta serán los que alcanzaran los estándares de calidad requeridos, así como el tener un sistema que permita enfrentar los cambios futuros debidos a las necesidades propias de la organización donde dicho sistema se encuentra.

De ahí la importancia de las arquitecturas de referencia las cuales proveen una plantilla de solución probada para un particular dominio y por lo cual este capítulo se enfoca a dar a conocer una panorama general de las diferentes arquitecturas para ciertos dominios que han sido desarrolladas.

3.1 Despliegue de componentes y arquitectónico.

En el diseño arquitectónico de la aplicación es importante visualizar tanto los nodos, como los componentes que estarán incluidos en ellos Para lo cual se puede utilizar UML para hacer un modelo físico tanto de componentes como de despliegue. En el primero de ellos se pueden ver de manera detallada las componentes de software que conforman la aplicación. En el segundo se describe la

arquitectura física de modo detallado, la forma en que las componentes serán desplegadas a lo largo de la infraestructura de la aplicación.

Diagrama de Componentes.

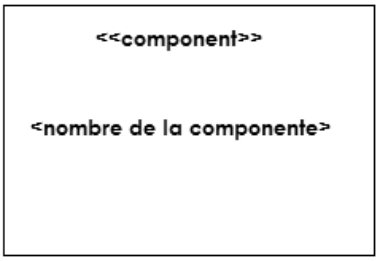
Antes de describir el diagrama de componentes, hay que definir qué es una componente. Una componente es una unidad de software basada en un estándar, que se caracteriza por no pertenecer a un contexto específico, es decir, es independiente de las plataformas de hardware y software, así como del lenguaje; no es modificable y brinda servicios a otras unidades, lo que significa que tiene múltiples usos, por lo que cuenta con interfaces para poder comunicarse con ellas.
Ejemplos de componentes serían una biblioteca dinámica de clases, un código ejecutable, un código fuente, una base de datos, motores de búsqueda, Controles ActiveX, Javabeans etc.,
Existen muchos tipos de componentes y de acuerdo al contexto también diversas clasificaciones, como por ejemplo se puede hablar de una clasificación con respecto aplicaciones en arquitectura de tres capas, en donde las componentes serían clasificadas como:

- componentes de interfaz de usuario, Aquellas unidades de software físicas que interactúan directamente con el usuario
- componentes de proceso o control, Aquellas unidades de software que son encargadas de controlar y procesar todas las peticiones hechas por el usuario
- componentes de acceso a datos. Aquellas unidades de software que se encargan de dialogar con los manejadores de bases de datos o manipular las bases de datos, para dar continuidad al procesamiento de las peticiones del usuario.

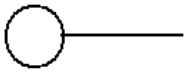
La ventaja de utilizar componentes es clara, ya que su reutilización reduce en tiempo y costo el ciclo de desarrollo de un sistema, a la vez que se incrementa su calidad y permite tener una aplicación que se adapte más fácilmente a los cambios del negocio.

Un diagrama de componentes es un elemento gráfico que muestra la disposición de las componentes así como su interrelación, a través de relaciones de dependencia. Cuando el número de componentes es muy grande o se requiere una vista especifica de las mismas, estas pueden ser incluidas en un diagrama paquete.
Como ya se había mencionado una componente posee interfaces a través de las cuales se comunica con las demás, ya sea para brindar servicios o solicitarlos, dichas interfaces en algunas ocasiones no se encuentran disponibles hasta en tiempo de ejecución, por lo cual una componente puede poseer uno o más puertos, para brindar está facilidad. Es por ello que un diagrama de componentes puede contener cuatro elementos que son: las componentes, sus interfaces para brindar y solicitar servicios, los puertos y las relaciones de dependencia. Los cuales se muestran en la figura 81

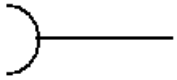
a) componente con un símbolo



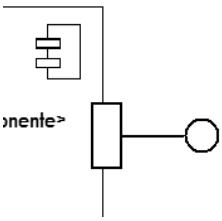
b) componente con estereotipo



b) oferta de interface (provista)



c) petición de interface (requerida)



d) puerto

d) dependencia

Figura 81. Símbolos utilizados en el diagrama de componentes

La figura 82 muestra un ejemplo de un diagrama de componentes

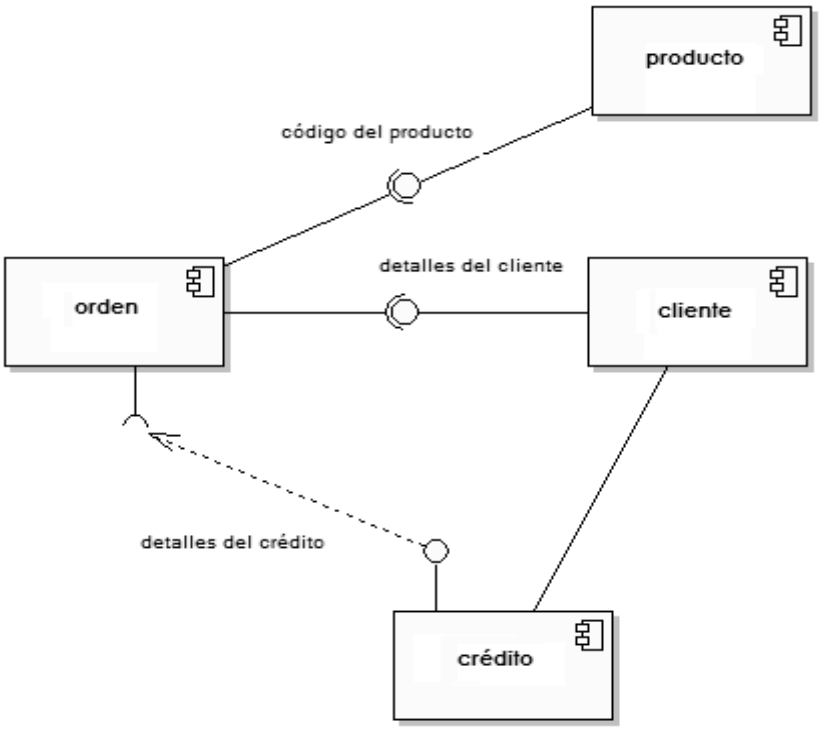


Figura 82. Diagrama de Componentes

Diagrama de Despliegue

El diagrama de despliegue es utilizado para mostrar la disposición arquitectónica de los elementos físicos de un sistema. Dichos elementos son nodos y artefactos.

- Un nodo es un dispositivo físico de hardware con capacidad de procesamiento: como una estación de trabajo, un cluster etc., o también lo es un entorno de ejecución de software como: un sistema operativo, el entorno J2EE, un manejador de bases de datos, etc.

  En su vista general un nodo es un recurso computacional sobre el cual los artefactos pueden ser desplegados para su ejecución, por lo que un dispositivo y un entorno de ejecución son subclases de un nodo.

  El símbolo utilizado para representar un nodo es un paralelepípedo con vista isométrica, dentro del cual se pone el nombre del nodo.

Cuando se quiere aclarar que el nodo es un dispositivo se pone la etiqueta <<device>> dentro, mientras que para distinguir que se trata de un entorno de ejecución de utiliza la etiqueta <<executionEnviroment>>.

Varios nodos conectados representan la topología de red que será utilizada como trayectoria de comunicación.

Un nodo puede contener artefactos y/o componentes, si el nodo es un dispositivo puede contener también nodos de entorno de ejecución.

Puede asimismo relacionarse con otros nodos y/o artefactos. Dichas relaciones pueden ser de asociación, para establecer una comunicación bidireccional o unidireccional, o pueden ser de dependencia.

Para representar una comunicación bidireccional se utiliza una línea entre los nodos que se comunican. Para representar una comunicación unidireccional se utiliza una flecha con punta abierta, la punta dirigida al receptor. Para ilustrar una dependencia se dibuja una flecha de punta abierta con línea achurada.

La siguiente figura muestra un diagrama de despliegue con dos nodos dispositivos relacionados y uno de ellos conteniendo a un nodo de entorno de ejecución, que a su vez contiene artefactos.

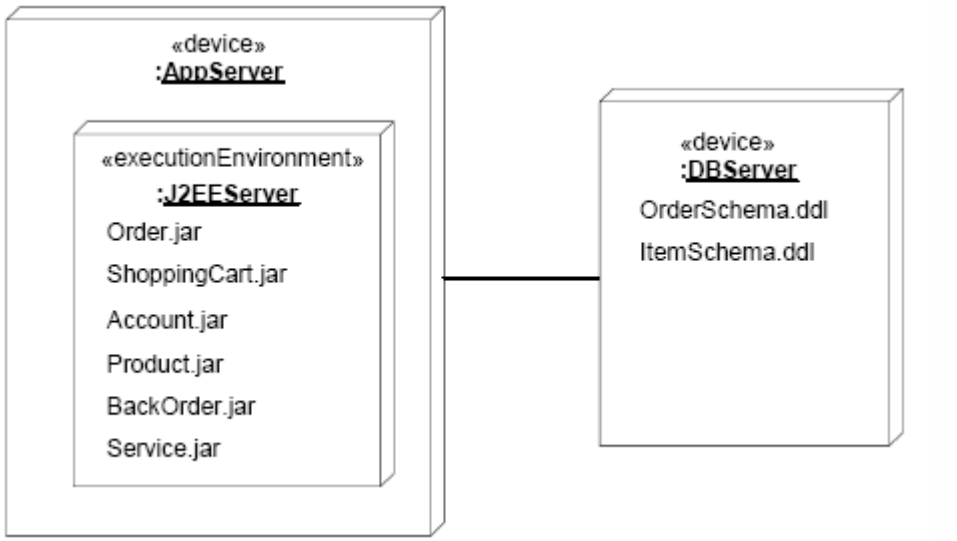


Figura 83. Ejemplo de relación entre nodos

- Un artefacto es la especificación de un elemento físico de información que puede ser utilizada o puede ser producida por el proceso del desarrollo de software, o por el despliegue u operación del sistema, ejemplos de éste serían: un ejecutable, una tabla de datos, un mensaje de correo, un seudocódigo como un jar, un documento fuente etc., puede ser considerado

como una instancia de un componente.

Para representar un artefacto se utiliza un rectángulo con la etiqueta de <<artefact>> dentro de él, también en algunas ocasiones puede ser representado por un ícono.

Para mostrar una dependencia de un artefacto con respecto a un nodo se utiliza una flecha de punta abierta con línea achurada y la etiqueta <<deploy>> sobre la flecha.

La siguiente figura muestra un ejemplo de dependencia de artefactos hacia un nodo.

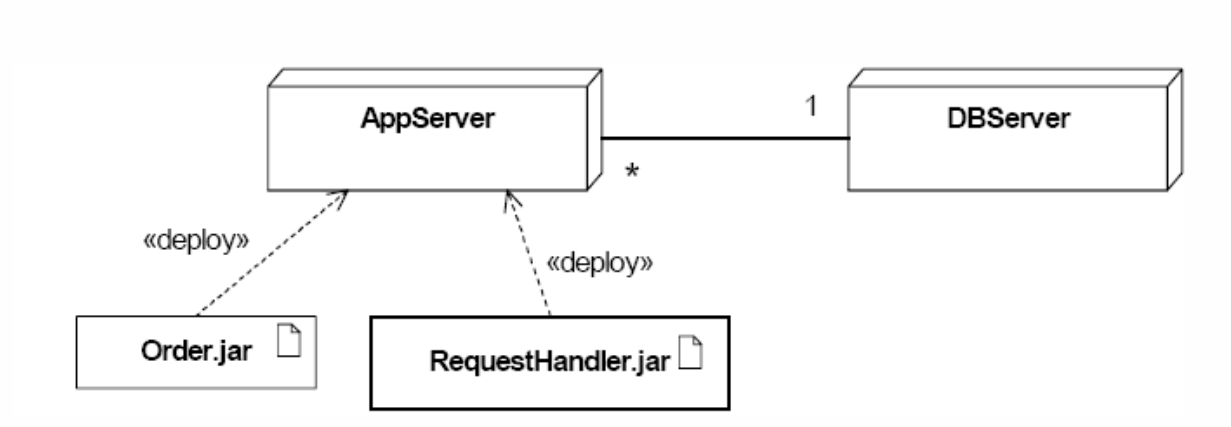


Figura 84. Dependencia de artefactos

La figura 85 muestra un ejemplo de un diagrama de despliegue, que utiliza la notación UML versión 1.4.

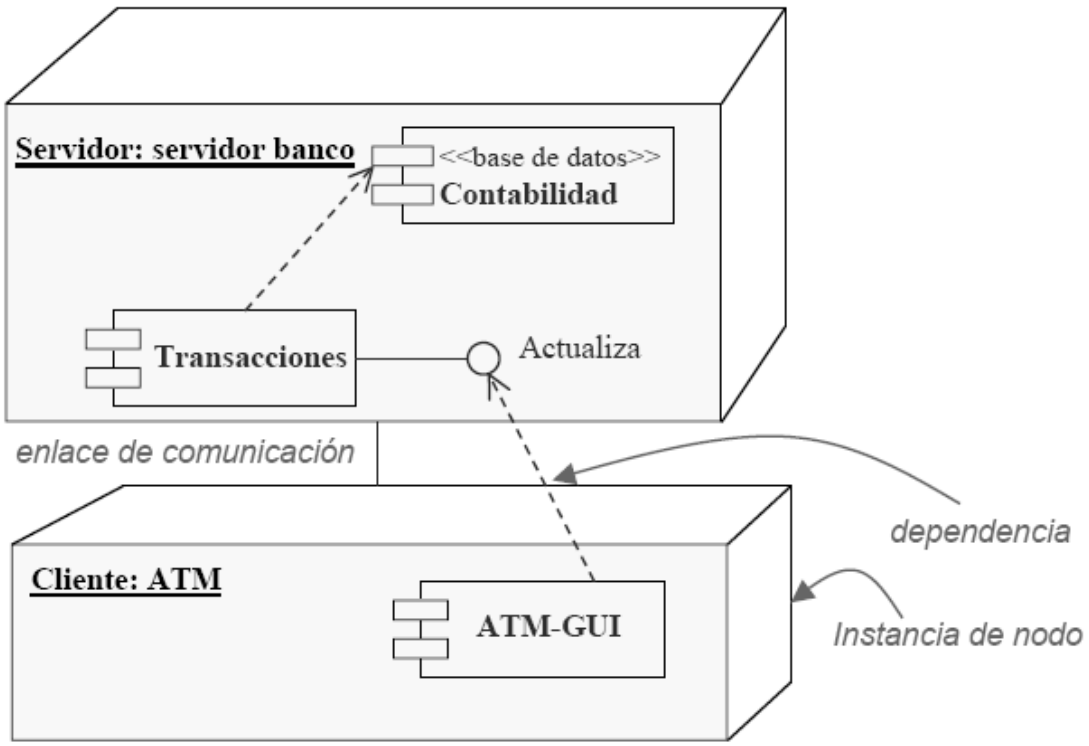


Figura 84. Ejemplo de un diagrama de despliegue UML versión 1.4

3.2 Técnicas de desarrollo de las arquitecturas de referencia en diferentes dominios.

Los sistemas computacionales son encontrados hoy en la actualidad en casi cualquier parte, ya que facilitan el procesamiento de la información y a su vez el acceso rápido a la misma. Es por ello que cada año surgen nuevos productos, metodologías, herramientas y técnicas que fortalecen su desarrollo y a la vez lo aceleran.

El uso del análisis, diseño y programación orientada a objetos incrementó la producción de los sistemas, debido a que este paradigma facilita la abstracción del mundo real para visualizarlo en forma de unidades lógicas que serán las encargadas de cooperar entre sí para procesar la información de dicho mundo real, de una forma más segura y eficiente. Por otra parte, aunque ya en la programación estructurada existía el concepto de reutilización al utilizar las librerías, la programación orientada objetos hizo más evidente que la reutilización era clave para hacer los sistemas en forma más rápida, a la vez que se garantizaba su calidad., esto se debe principalmente a que una clase que ya ha sido ampliamente probada, rectificada en todos sus errores, garantiza su uso, lleva con ella la valiosa experiencia de su desarrollador y por lo tanto no hay que invertir otra vez tiempo en volverla a programar.

Es esta experiencia de muchos desarrolladores de software que por fin ya no se pierde, puesto que así como los objetos colaboran entre sí, los desarrolladores comenzaron a darse cuenta que muchas de las soluciones que daban a ciertos problemas específicos en el desarrollo eran similares a los que los demás desarrolladores daban, es así como surgen los patrones de diseño con la colaboración de desarrolladores con mucha experiencia en la industria de la ingeniería del software. Un patrón de diseño es una solución general a un problema general que pude ser adaptada a una situación en particular. Un patrón de diseño es sola una plantilla, un bosquejo de una solución, no una solución en sí misma para un problema en particular, pero lo importante es que sirve como referencia, como un punto de apoyo del cual partir. Actualmente existen varios patrones de diseño elaborados por los mejores y más experimentados desarrolladores de software, esto dio a entender al mercado de software que otro punto clave a parte de la reutilización era contar con una referencia en que apoyarse para el desarrollo de software.

Dado que existe una gran variedad de ámbitos en los cuales los sistemas computacionales existen, éstos pueden tener diferencias abismales, en cuanto a las plataformas de hardware y software en las que se ejecutan, en cuanto a la seguridad en el manejo de la información, en cuanto al tiempo de respuesta que deben brindar, en cuanto al medio ambiente en donde se encuentran, o en cuanto al tipo de información y con ello los conceptos que se emplean en el mismo, los cuales pueden llegar a ser muy especializados, como en el caso de los sistemas empotrados o de aquellos utilizados en la

exploración espacial, o en la medicina, etc.,
Por otra el costo involucrado en un sistema computacional puede ser demasiado para una empresa, no son pocas las historias de compañías que ha perdido incluso millones de dólares en el desarrollo e implementación de un sistema.
Debido a estas circunstancias resultaba deseable que tanto el desarrollador como las empresas contarán con alguna forma que le indicase, donde es más adecuado y productivo el realizar sistemas dentro de la organización.
Es por esta razón que surgen las arquitecturas de referencia, a través de las cuales es posible establecer un marco de referencia que permite a los desarrolladores crear un entorno extensible, confiable, seguro y fácilmente administrable mediante un conjunto de herramientas, tecnologías y procesos recomendados al dominio.

3.2.1 Los modelos de componentes.

Dado que el desarrollo orientado a objetos ya había mostrado la importancia de la reutilización, la elaboración de componentes permitió que los objetos traspasaran las barreras del lenguaje y de las plataformas de hardware y software, para que esos mismos objetos formasen grupos que dieran servicios a más de un sistema. Es por ello que se desarrollaron modelos de componentes que sirvan a una arquitectura de referencia para tener un modelo de referencia que se encargue de localizar y representar gráficamente la distribución relativa de las componentes de software, así como sus relaciones en un sistema.

El paradigma de desarrollo de componentes ha contribuido a facilitar la construcción de software, debido principalmente a que su nivel de reutilización es mayor en comparación con el sólo utilizar objetos, otra razón importante es que se puede probar de manera independiente cada componente antes de ensamblarlas, también el mantenimiento es más simple puestos que debido a que la dependencia entre ellas no es grande , es decir, que el acoplamiento es débil, son fáciles de sustituir por otras que cumplan con los nuevos requisito, requeridos en el mantenimiento. La calidad y rapidez de construir con componentes es evidente, puesto que una componente en sí es un software ampliamente probado y que por lo regular es mejorado continuamente.

Pero la reutilización no necesariamente involucra la creación de componentes, ya que se puede construir software a partir de partes y/o aplicaciones ya existentes almacenadas en repositorios, sin embargo esta práctica involucra un mayor esfuerzo puesto que dichas partes o aplicaciones deben

ser adaptadas, puesto que no cumplen con todas las características de una componente como por ejemplo puede ser que carezcan de interfaces o sean fácilmente identificables.

Dentro del modelo de componentes existe un concepto importante el cual es llamado granularidad. La granularidad sirve para indicar el nivel de desarrollo involucrado en una componente ya que ésta puede ser desde una simple función o subrutina hasta una aplicación que puede ser usada por alguna otra. La primera por lo regular se llama componente de grado fino y la segunda componente de grado grueso. Sin embargo el conocer la granularidad de una componente no es suficiente para saber que componentes se deben seleccionar, puesto que a lo más da información del tipo de componentes que hay a lo largo del camino de una aplicación, ya que se cuenta con componentes GUI, componentes operacionales, componentes de negocio etc., así que el seleccionar las componentes más adecuadas requiere de estrategias que dependerán principalmente del entorno donde van hacer aplicadas.

Existen básicamente cuatro formas de desarrollar con componentes:

- Construir componentes propias
- Construir con componentes comerciales (COTS, Commercial-Off-The_Shelf)
- Construir con componentes distribuidos
- Construir con componentes de Servicios Web

Todas las formas tienen ventajas y desventajas, si se opta por la primera la ventaja es que el software construido cumplirá con los requisitos específicos para lo que ha sido diseñado sin adaptaciones innecesarias, sin embargo pudiese ser que el costo sea mayor al igual que será mayor el tiempo en su desarrollo. Por otra parte una componente comercial puede ser que ahorrará tiempo, sin embargo quizás fuese necesario hacerle algún ajuste, para que cumpla plenamente con los requisitos de diseño.

La lucha entre componentes propios y COTS, continúo debido a que estos últimos, no resultaron tan económicos, ni tan poco para los vendedores fáciles de desarrollar, adaptar y utilizar, debido principalmente a la falta de una estandarización entre los diferentes vendedores, sólo aquellos componentes de grano fino y cuya funcionalidad es muy conocida a la vez que genérica, como la de algunos componentes gráficos, como por ejemplo un calendario, han tenido éxito.

En los últimos años han surgido modelos de componentes que impulsan el desarrollo basado en componentes para sistemas distribuidos, estos modelos son:

- COM y DCOM de Microsoft. COM (Component Object Model) es un estándar que permite la creación de objetos que ejecuten tareas que resuelven problemas específicos pero comunes a varias aplicaciones que puedan desear hacer uso de ellos. Estos pueden ser invocados por diferentes programas que los requieran, tanto OLE como ActiveX están basados en esta tecnología. La idea es tener un mundo de objetos independientes de un lenguaje de programación. Por ello COM proporciona un estándar para las comunicaciones entre componentes, de tal forma, que una aplicación puede utilizar características de cualquier otro objeto de la aplicación, o del sistema operativo, y permite actualizar el software de un componente sin afectar a la operación de la solución global. COM soporta comunicación entre objetos de equipos de cómputo distintos, en una LAN, WAN, o incluso en Internet.

  DCOM extiende el estándar COM de objetos remotos, para su utilización en redes. Inicialmente se desarrolló para Windows NT 4.0, y posteriormente para Solaris 2.x y Macintosh, así como para diferentes versiones UNIX. Se encarga de manejar los detalles muy bajos de protocolos de red, por lo que el desarrollador se puede centrar en la realidad de los negocios, proporcionando así mejores soluciones a los clientes. La arquitectura define cómo los componentes y sus clientes interactúan entre sí. Esta interacción es definida de tal manera que el cliente y el componente pueden conectarse sin la necesidad de un sistema intermedio. El cliente llama a los métodos del componente sin tener que preocuparse de niveles más complejos. DCOM olvida completamente la localización de los componentes, no importando que estén en el mismo proceso que el cliente o en una máquina en cualquier lugar del mundo. En cualquier caso, la forma en la que el cliente se conecta a un componente y llama a los métodos de éste, es idéntica. No es sólo que no necesite cambios en el código fuente, sino que además no necesita que el programa sea recompilado. Una simple reconfiguración cambia la forma en la que los componentes se conectan entre sí.

  La independencia de localización en DCOM simplifica enormemente las tareas de los componentes de aplicaciones distribuidas para alcanzar un nivel de funcionamiento óptimo. Supongamos, por ejemplo, que cierto componente debe ser localizado en una máquina específica en un lugar determinado. Si la aplicación tiene numerosos componentes pequeños, se puede reducir la carga de la red situándolos en la misma LAN, en la misma máquina, o incluso en el mismo proceso. Si la aplicación está compuesta por un pequeño número de grandes componentes, la carga de red es menor y no es un problema, por tanto se pueden poner en las máquinas más rápidas disponibles independientemente de donde estén situadas. Es completamente independiente del lenguaje. Casi cualquier lenguaje puede ser utilizado para crear componentes COM, y estos componentes puede ser utilizado por muchos más lenguajes y herramientas. Java, Microsoft Visual C++, Microsoft Visual Basic, Delphi, PowerBuilder, y Micro Focus COBOL interactúan perfectamente con DCOM. Puede utilizar

cualquier protocolo de transporte, como TCP/IP, UDP, IPX/SPX y NetBIOS, y proporciona un marco de seguridad a todos estos protocolos. Los desarrolladores pueden utilizar las características proporcionadas por DCOM y asegurar que sus aplicaciones son completamente independientes del protocolo. DCOM está pensado para que el sistema pueda funcionar bajo cualquier tipo de red, ya sea LAN, WAN o Internet, de forma que se solucionen los múltiples problemas que añaden estos entornos.

- EJB de Sun. EJB (Enterprise Java Beans) es un modelo de componentes que permite la elaboración de aplicaciones distribuidas, seguras y portables desarrolladas en lenguaje java del lado del servidor, ya que provee la infraestructura necesaria en sus componentes para cumplir con los requerimientos de alto nivel como son la distribución, transacción, persistencia, manejo de mensajes, seguridad y conectividad con sistemas mainframes o con sistemas ERP. La base de la funcionalidad de las componentes EJB es su contenedor, ya que es éste el que provee de los servicios para el manejo de transacciones, concurrencia, servicios de red, gestión de mensajes, persistencia e incluso escalabilidad, entre otros. Un modelo de componentes distribuido debe cumplir con ciertos requerimientos como son proporcionar mecanismos proxy tanto para el cliente como para el servidor, para proveer la infraestructura básica para recibir las solicitudes del lado del cliente y delegar su tratamiento a los correspondientes objetos implementados en el lado del servidor, así como el informar al sistema cuando un componente ya no es utilizado por un cliente, es por lo que EJB proporciona dos interfaces a cada Bean para cumplir con estos requisitos llamada interfaz EJBHome , la cual proporciona al cliente un medio por el cual puede localizar y también crear instancias de beans, y la interfaz EJBObject, que será la única vía a través de la cual el cliente puede solicitar los servicios a un bean., ambas interfaces son remotas y ambas heredan de la interfaz remote de RMI. Existen tres tipos de beans en EJB:
  - ✓ el bean de sesión o session bean, que como su nombre lo indica sirve para representar el flujo del proceso de negocio en una sesión interactiva con el cliente , y que puede mantener un estado o no. El bean de sesión mantiene una relación uno a uno con el cliente por lo que habrá tantos sesión beans como clientes conectados a la aplicación. Existen a su vez dos tipos de beans de sesión:
    - o con estado. En las variables de instancia de este tipo de bean de sesión son guardados los estados de la conversación cuando se requiere mantener cierta información del cliente para futuras invocaciones a métodos o para algunas otras componentes en la aplicación. Cuando la sesión termina la información almacenada en las variables se pierde, es decir sólo estará presente a lo largo de la sesión. A pesar de ello la actualización de esta información de estado requiere

más tiempo de ejecución que lo que requiere un bean de sesión sin estado.

- Sin estado. En este tipo de bean de sesión, los métodos solicitados son simplemente ejecutados y en su caso devuelven únicamente los valores de retorno. Estos métodos pueden ser concurrentes, ya que en cuanto un cliente termine la ejecución del método la instancia del bean quedará libre para usarse con otro cliente.

- ✓ el entity bean que como su nombre lo indica es utilizado para representar las entidades u objetos que existen en el mundo real de la aplicación, junto con su funcionabilidad que es implementada a través de reglas de negocio y están asociados con las bases de datos que contienen la información persistente de dichas entidades.
- ✓ el message driven bean, este tipo de bean a diferencia de otros beans no hace uso de las interfaces EJBHOME ni de la EJBObject, ya que es un objeto local que hace uso del servicio de mensajería de java, mejor conocido como JMS por sus siglas en inglés. A través de la cual se permite una comunicación asíncrona con el cliente.

A pesar de haber catalogado al EJB como un modelo de componentes distribuidos, nada le impide ser utilizado para desarrollar aplicaciones Web, ya que sus componentes pueden ser desplegados a través de archivos tipo war(en lugar de jar) y de un descriptor de despliegue de tipo XML. Por lo que EJB es uno de los preferidos por aquellos desarrolladores de web que utilizan el lenguaje java

- CCM del grupo OMG. El CCM o CORBA Component Model, es un modelo impulsado por el grupo OMG el cual a su vez desarrolló y estandarizo la arquitectura de desarrollo para objetos distribuidos CORBA, la cual al ser un estándar cuenta con un conjunto de especificaciones, sobre las cuales los vendedores de implementaciones de CORBA, conocidas como Object Request Broker (ORB) se apegan, para facilitar la comunicación con la implementación de diferentes vendedores de objetos distribuidos. La función del ORB consiste en conectar dos partes: cliente y servidor, las cuales pueden quizás ejecutarse sobre plataformas distintas y funcionan con diferentes sistemas operativos. Esto significa que pueden existir diferencias en tipos de datos, el orden de los parámetros en una llamada, el orden de los bytes en una palabra según el tipo de procesador, etc. Es misión del ORB efectuar los procesos conocidos como marshaling y unmarshaling. En caso de que el método invocado devuelva un valor de retorno, la función de los ORB del cliente y servidor se invierte. Éste realiza el marshaling de dicho valor y lo envía al ORB del cliente, que será el que realice el unmarshaling y finalmente facilite el valor en formato nativo. El cliente es implementado a través de un cliente stub y el servidor a través de un servidor skeleton. El Cliente Stub es una entidad de programa que

invoca una operación sobre la implementación de un objeto remoto a través de un Stub cuyo propósito es lograr que la petición de un cliente llegue hasta el ORB Core. Logrando el acoplamiento entre el lenguaje de programación en que se escribe el cliente y el ORB Core. El stub crea y expide las solicitudes del cliente. Un Servidor Skeleton es la implementación de un objeto CORBA en algún lenguaje de programación, y define las operaciones que soporta una interface IDL CORBA. Puede escribirse en una gran variedad de lenguajes como C, C++, Java, Ada o Smalltalk. Y a través del skeleton entrega las solicitudes procedentes del ORB a la implementación del objeto CORBA.

El CCM extiende las capacidades del modelo CORBA permitiendo a los desarrolladores implementar, manejar, configurar y desplegar componentes CORBA que brinden servicios tales como: transacción, persistencia, seguridad y eventos de notificación de servicios en ambientes estándar. Provee una interface de definición de lenguaje IDL, interoperabilidad a través de sus protocolos GIOP (General Inter-ORB Protocol) y el IIOP (Internet Inter-ORB Protocol) para internet. El CCM posee un contenedor para que los componentes de software puedan ser desplegados en él y hacer uso de los servicios que brinda, similar al contenedor de los EJB. Actualmente se encuentra en su versión 4.

Con la llegada de Internet el uso de las computadoras se volvió masivo, y las grandes corporaciones internacionales tuvieron un medio a través del cual podían desplegar algunas de sus aplicaciones. Pero para ello se requería primero que este medio garantizase la confiabilidad, integridad y seguridad de su información, por lo cual la industria del software ha venido desarrollando recientemente un conjunto de protocolos y estándares abiertos, como XML, SOAP, WSDL y UDDI que han permitido la interoperabilidad entre aplicaciones desarrolladas con diferentes lenguajes y que incluso pueden ser ejecutadas en computadoras con diferentes plataformas de hardware y software, pero aún así comunicarse a través de Internet. A estos estándares se les conoció en un inició como servicios web, aunque actualmente este concepto ha ido más allá, por lo tanto hoy en día se considera que un ***Servicio Web*** es una aplicación basada en WEB que posee un identificador URI que hace uso de estándares y protocolos Web para brindar una prestación a través de interfaces y referencias bien definidas a otras aplicaciones, lo cual la diferencia de las aplicaciones web comunes, ya que por lo general estas brindan servicios a través de interfaces gráficas a uno o más usuarios humanos. Por otra parte las aplicaciones Web comunes se acceden a través de direcciones de páginas en Internet, es decir, direcciones URL (Uniform Resource Locator), que sirven como identificadores de dichas páginas. Los servicios Web quedan identificados por su URI (Uniform Resource Indetifier) que es un identificador más general que URL y también por su interfaz, la cual es de vital importancia pues sólo a través de ella es como se puede acceder al Servicio Web en cuestión. De hecho una aplicación

web podría ser un cliente de un servicio web, considere una aplicación web de compra de libros a través de Internet, esta aplicación puede solicitar un servicio web de tarjetas de crédito.
Los estándares y protocolos conocidos como HTTP, FTP y SMTP se siguen utilizando con los servicios web.
Por su parte XML es una especificación desarrollada por el consorcio triple W (W3C), el eXtensible Markup Language, ayuda a los diseñadores a especificar una serie de comandos o tags propios para habilitar la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre empresas.
Mientras que el Protocolo Simple de Acceso a Objetos (SOAP), y que se basa en XML especifica el formato de los mensajes que envían los dispositivos cuando interactúan entre ellos a través del protocolo HTTP o el protocolo SMTP, es decir está por encima de estos protocolos.
Por otro lado el Lenguaje de Descripción de Servicios Web (WSDL) se encarga de especificar tantos los mecanismos como la sintaxis del intercambio de mensajes, ya que para que una aplicación pueda comunicarse de forma automática con un servicio, tanto la dirección del servicio como su interfaz deben estar especificadas lo que se hace con WSDL.
Y por último el protocolo Universal de Descripción, Descubrimiento e Integración (UDDI) que funciona como un directorio en el cual se publican las aplicaciones Servicios Web.
Podría decirse de a manera de resumen que XML es utilizado para especificar comandos para el tratamiento de los datos, SOAP es usado para transferir dichos datos, WSDL se usa para describir los servicios disponibles y UDDI para listarlos.

Además estos estándares de manera similar que el modelo ISO conforman cuatro capas que se pueden catalogar como:

- Servicios de Transporte, donde se tiene los protocolos HTTP, SMTP, FTP y BEEP, siendo este último especificado en XML y además permite la comunicación P2P, y también recientemente a esta capa se ha incorporado JMS que es un servicio de mensajería que sirve como una API de Java.
- Servicios de Mensajería, en esta capa se encuentra SOAP.
- Servicios de descripción. Donde se encuentra WSDL
- Servicios de descubrimiento, en esta capa se tiene a UDDI

Gracias al avance de los Servicios Web, la Arquitectura Orientada a Servicios conocida por sus siglas en inglés como SOA ha tomado nuevos bríos, esta infraestructura desarrollada a finales de los noventa, proporciona un marco de trabajo para que las organizaciones a través de la aplicación de tecnologías de información integren la lógica de negocio y los datos de sistemas separados. Con el

uso de los servicios web se pueden tener aplicaciones débilmente acopladas y con un interoperabilidad alta. Para SOA tanto las aplicaciones como la información son como bloques, a los cuales debe brindárseles de las componentes necesarias que permitan que se conviertan en bloques modulares de tal forma que la interoperación entre ellos sea fácil y rápida. Lo cual se dice fácil pero es un proceso largo y complejo, que sin embargo a futuro tendrá proveerá a la empresa con grandes beneficios ya que la inversión realizada en las aplicaciones y bases de datos antiguas no se perderá. Por lo que si se opta por implementar SOA debe ser de una forma escalonada.

Como puede deducirse de este tema, las aplicaciones monousuario han quedado muy atrás, es por ello que las nuevas generaciones deben ver más allá de su computadora personal y estar consientes que las aplicaciones que han de desarrollar no sólo serán multiusuarios sino multiempresarias.

3.2.2 Arquitectura de referencia para sistemas de tiempo real.

Los sistemas en tiempo real nacieron cuando nació la necesidad de tener un sistema que actualizará la información de tal forma que pudiese volver a ser utilizada casi inmediatamente, por más de un usuario y en lugares distantes, como por ejemplo, la venta de boletos de líneas aéreas, en este caso los usuarios de un punto de venta de boletos requieren saber si existe cupo en un vuelo de un avión de una aerolínea con un destino y tiempo de salida determinados, así como el número de asientos su ubicación etc., y cuando un boleto sea vendido debe actualizarse inmediatamente la base de datos e impedir que otro usuario en otro punto de venta distinto tenga acceso a ella o a un grupo de registros en las tablas de datos en particular.

Sin embargo con el desarrollo presente de los equipos computacionales, así como, de los sistemas operativos y manejadores de bases de datos, este concepto ha cambiado, ya que a este tipo de sistema ya no se le considera un sistema en tiempo real, sino un sistema de información distribuido. ¿Entonces actualmente en qué consiste un sistema en tiempo real?

En la actualidad un sistema en tiempo real es un sistema constituido por un conjunto de procesos que trabajan concurrente o paralelamente, y el sistema monitorea y a la vez controla el entorno para el cual fue diseñado, respondiendo a éste en lapsos de tiempo muy corto. Un sistema en tiempo real físicamente se encuentra asociado a dispositivos hardware de dos tipos: sensores y actuadores. Los

sensores son los encargados de monitorear o recopilar la información del entorno, mientras que los actuadores están programados para desarrollar tareas en repuesta a la información recopilada en un tiempo cuyo rango por lo regular se encuentra en segundos o menos, lo cual dependerá de la aplicación para la que ha sido diseñado.

Se podría decir a manera de broma que un sistema en tiempo real sigue la tercera ley de Newton ya que a cada acción corresponde una reacción, ya que este sistema funciona bajo un binomio que es estímulo-respuesta. El estimulo puede darse de dos maneras distintas:

- Síncrona: Cuando el estímulo es periódico, es decir, se da siempre tras el mismo intervalo de tiempo.
- Asíncrona. Cuando el estimulo no es en un tiempo cíclico sino a intervalos irregulares

Como puede desprenderse de la definición actual de un sistema en tiempo real, estos tienen características muy distintas de los sistemas de información normales, ya que necesitan de un hardware especial, incluso podrían carecer de teclado y pantallas normales como el caso de los sistemas empotrados que es un tipo de sistema de tiempo real y en los cuales los dispositivos de entrada y salida pueden llegar especialmente a ser diseñados y elaborados para la aplicación, que podría ser por ejemplo los sistemas que controlan las comunicaciones en los trenes modernos, o un sistema encargado de controlar los ciclos de lavado y secado de un centro de lavado y secado para el hogar. Su programación también requiere de un conocimiento especializado del entorno en cuestión.
Otra característica que distingue a los sistemas de tiempo real es que necesitan un entorno especial para su desarrollo como el que proporciona un sistema operativo en tiempo real, que a diferencia de los sistemas operativos normales, es un sistema que es predecible, es decir, determinista ya que debe conocer con alto grado de certidumbre el tiempo que al sistema operativo reconocerá una interrupción ocasionada por alguno de los sensores y por otra parte el tiempo que el sistema operativo llevará en dar seguimiento a dicha interrupción a través de alguna rutina de tratamiento. Además un sistema operativo en tiempo real debe manejar los fallos de manera que a pesar de que se presente un error, debe continuar su ejecución y mientras tanto tratar sino de corregir el fallo minimizar sus consecuencias. Ejemplos de algunos de estos sistemas operativos reales son: QNX, un sistema basado en Unix y principalmente diseñado para sistemas empotrados. VxWorks, es otro sistema operativo en tiempo real basado también en Unix, y que puede ejecutarse en la mayoría de las computadoras de arquitectura modernas. Otras sistemas operativos en tiempo real son: RTLinux y Portos un sistema operativo de tiempo real de propósitos educacionales y orientado a componentes.

Los sistemas de tiempo real se pueden clasificar de acuerdo al tiempo límites como:

- Sistemas duros. Se define a un sistema de tiempo real duro como aquel que se declarará como un sistema de operación incorrecta, si los resultados esperados que debe proveer, no son dados en el tiempo estipulado en los requerimientos. Por lo que cualquier respuesta fuera de los parámetros de tiempo establecidos no deben ser tolerados. Ejemplos de este tipo de sistemas podría ser un sistema empotrado para trenes que controle el frenado del mismo o un sistema empotrado de un automóvil que controle la expulsión de las bolsas de aire en un accidente.
- Sistemas suaves. Por sus parte un sistema de tiempo real suave será aquel cuya operación se ve como degradada si los resultados esperados no se cumplen dentro de los requerimientos de tiempo especificados. Esto significa que existe un rango de tolerancia mayor en que una respuesta que ocasionalmente no cumpla con las restricciones de tiempo no será considerada como intolerable. Ejemplos de este tipo de sistemas son: un juego de computadora y un sistema empotrado en una lavadora.
- Sistemas firmes. Un sistema de tiempo real firme es aquel en el que a diferencia de los duros permiten de manera ocasional que existan violaciones en los límites de tiempo, pero las tareas involucradas que no hayan concluido su trabajo en dichos límites de tiempo son abortadas. Un ejemplo de este tipo de sistema son los sistemas multimedia.

Los sistemas en tiempo real también pueden clasificarse de acuerdo a la respuesta al estímulo como:

- Sistemas activados por tiempo. Donde el estímulo es cíclico o periódico
- Sistemas activados por evento. Donde el estimulo produce un cambio de estado.

Y también los sistemas en tiempo real pueden ser clasificados como:

- Centralizados. Toda la aplicación se ejecuta en una sola CPU.
- Distribuidos. La aplicación es ejecutada a lo largo de un conjunto de nodos cada uno poseedor de una CPU y memoria propias

El diseño de un sistema de tiempo real no es sencillo, sin embargo existen ciertos factores comunes en los sistemas de tiempo real que deben ser considerados en su diseño como son:

- Tamaño: El número de líneas de código necesarias para su programación puede influir en hacer al sistema complejo, afectando de esta manera su implementación, así como su

mantenimiento. Es por ello que es recomendable el hacer uso de técnicas y herramientas orientadas a objetos, e incluso el uso de componentes, conforme sea su tamaño, para facilitar su diseño.

- Dominio. Es de vital importancia que los diseñadores tengan una amplia experiencia en el entorno donde radicará y funcionará el sistema, para que dominen el lenguaje técnico utilizado en dichos sistemas, así como entiendan a la perfección los requerimientos que el sistema debe cumplir, sobre todo aquellos relacionados al quehacer y tiempos de ejecución de los actuadores, para que puedan elegir los elementos tantos de hardware (CPU, sensores, actuadores etc.,) como de software (sistema operativo, lenguaje, entorno de desarrollo etc.,) óptimos al dominio de la aplicación.
- Errores. Dependiendo del dominio de la aplicación un error es un sistema de tiempo real puede llegar a ser catastrófico, tanto humana como económicamente. Es por ello que la probabilidad de fallo, así como la probabilidad de recuperación de fallo, que deberá tener el sistema deben ser consideradas como uno de los factores más importantes en el diseño del sistema, proveyendo a éste con los necesarios mecanismos de redundancia y recuperación de errores, para obtener un sistema fiable y seguro dentro de los parámetros establecidos.
- Concurrencia. Un sistema en tiempo real por lo regular interactúa de manera concurrente o paralela con varios dispositivos tanto de entrada como de salida, que pueden ser de decenas a miles, como en el caso de un sistema de conmutación telefónica. Es por ello que el sistema debe contar con un conjunto adecuado de mecanismos, tanto para la sincronización como para la comunicación, síncrona y/o asíncrona, entre el sistema y dichos dispositivos. La elección de dichos mecanismos debe ser llevada a cabo en la fase del diseño de la lógica del sistema, para determinar cuáles son los más adecuados lo cual se llevará a cabo en la fase del diseño de la arquitectura física y el análisis temporal. Lo que no necesariamente implica que puedan ser modificados si en la fase de pruebas y medidas del tiempo resultan no ser óptimos.
- Monitoreo y control. Los sistemas de tiempo real deben ejecutar continuamente procesos de monitoreo a través de sus dispositivos hardware llamados sensores, Este monitoreo consiste en recopilar información que por lo general originalmente es analógica y luego convertida a información digital, lo cual es llevado a cabo por circuitos electrónicos conocidos como convertidores analógicos – digitales. Este tiempo debe ser tomado en consideración como un parámetro importante de los sistemas en tiempo real, ya que una elección inadecuada de un sensor puede ser la causa de que a pesar de un correcto diseño y programación del sistema, éste no cumpla con los requisitos de tiempo requeridos. Lo mismo sucede en el caso de los actuadores que son los encargados de actuar conforme a los datos obtenidos de los sensores,

por lo cual la elección adecuada de los mismos es también un factor muy importante a tenerse en cuenta.

Considerando el paradigma tanto en la programación como en el análisis y diseño orientado a objetos, las etapas del ciclo de desarrollo genérico de un sistema de tiempo real podrían ser las siguientes:

- Análisis de los requisitos, especialmente de las respuestas que el sistema debe dar a los estímulos, así como el tiempo límite que debe tener como máximo cada una de las respuestas.
- Diseño de los casos de uso. Dado que un sistema de tiempo real puede llegar a tener interacción no sólo con eventos de su entorno sino que también con seres humanos, es importante identificar a todos las entidades involucrados, ya sea que estas provoquen eventos o interactúen directamente con el sistema, con el fin de definir las más adecuadas interfaces para estas entidades. También deben quedar identificados los escenarios más importantes de interacción para puntualizar los casos de uso que darán soporte a cada binomio de acción – reacción.
- Diseño de la lógica. Una vez definidos los casos de uso, se deben determinar cuales serán los objetos que estarán encargados de recopilar la información de los sensores, así como los encargados de procesar dicha información e informar los resultados a aquellos objetos encargados de comunicarse con los actuadores para indicarles las acciones que deben llevar a cabo de acuerdo a los resultados obtenidos del procesamiento de la información de entrada. Y una definidos dichos objetos deben elaborarse diagrama de estado para tener un análisis detallado y un modelado correcto de todos los probables estados del sistema y de los eventos que dispararan una transición de un estado a otro.
- Diseño de la arquitectura. Ya determinados los objetos, su lógica y estados, se debe determinar que componentes actuarán de manera concurrente o paralela, para considerar su despliegue a lo largo de la arquitectura física, es decir, se debe conocer el sistema operativo, el número y tipo de dispositivos físicos, así como, como el número y tipo de CPU en la cual se ejecutará la aplicación y los componentes y como estos serán desplegados o instalados a través de estos CPU' s.
- Diseño de los diagramas de interacción. Deben quedar especificados la secuencia y el formato de comunicación que tendrán a cabo los objetos durante la ejecución de sistema.
- Diseño y codificación de los algoritmos de procesamiento de los estímulos.
- Diseño del programa de tiempos que debe contener los tiempos límites estipulados para cada

respuesta a un estimulo, así como de los parámetros que garanticen dichos tiempos, tanto como la fiabilidad y la seguridad del sistema.

Dado que algunos sistemas pueden llegar a manejar miles de eventos y tener millones de líneas de código, sobre todo tratándose de sistemas de tiempo real distribuidos, grupos de empresas usuarios de este tipo de sistemas de giros que varían de telecomunicaciones hasta la industria aeroespacial todos ellos entusiastas usuarios de CORBA, convinieron junto con el grupo OMG en establecer lo que se conoce como Real Time CORBA Specification. Esta especificación tiene como propósito proveer un estándar que sirva como una arquitectura de referencia para construir un sistema cuyos protocolos de comunicación aseguren una "predecibilidad de fin a fin" del sistema, especialmente en sistemas con planificadores de corto plazo que manejan algoritmos apropiativos por prioridades. Los vendedores que elaboran productos de software CORBA para el desarrollo de sistemas distribuidos, no sólo impulsaron esta especificación, sino que inmediatamente la implementaron en sus productos pues el mercado tenía tiempo esperando contar con componentes ORB que ayudasen a resolver algunos de los problemas que se presentan en el desarrollo de sistemas distribuidos de tiempo real.
Si una componente del sistema manifiesta un comportamiento no predecible, esto puede afectar seriamente a todo el sistema. Ejemplo de tales componentes son las prioridades de los procesos clientes o de los procesos servidores que tengan planificación apropiativa.
Por lo que para decidir a priori si un requerimiento debe ser atendido, el sistema deberá contar con predictibilidad. Esto es sólo posible cuando las componentes de un sistema son deterministas y son capaces de combinarse de tal manera que entre ellas puedan proveer de predictibilidad. Las interfaces y mecanismo estipulados en la especificación de tiempo real de CORBA garantizan una combinación predecible del ORB y la aplicación. La cual se encarga de manejar los recursos a través de las interfaces proporcionadas por Real Time CORBA y los mecanismos del ORB se encargan de coordinar las actividades comprometidas en la aplicación.

Por otra parte la empresa Sun MicroSystems a través de su lenguaje java ha tratado de llevar a la realidad el tener aplicaciones que puedan ser ejecutadas en cualquier plataforma de hardware y software sin tener que volver a programarlas para cada uno de ellas, y no sólo para las aplicaciones de escritorio y Web con el J2SE y Java Script, también ha sido el caso para aplicaciones distribuidas con su entorno de desarrollo J2EE y para aplicaciones móviles a través del J2ME e igualmente Sun MicroSytems ha entrado al mundo de las aplicaciones de tiempo real con su especificación JSR-1: Real-time Specification for Java, cuya primera edición fue liberada en Mayo del 2000 y que actualmente se encuentra implementada de manera comercial en el producto Real-Time System Java en su versión 2.1, que básicamente modifica la máquina virtual de java para brindar una innovación

en el manejo de tiempo real específicamente en ocho puntos:

- Acceso a memoria Física. Se han desarrollado clases que permitan tener acceso a la memoria, ya sea para implementar el mapeo de memoria física para dispositivos de entrada y salida o para los controladores.
- Gestión de memoria. Una de las ventajas y a la vez problema con Java, es su recolector de basura, el cual se encarga de la destrucción de objetos una vez que ya no van a ser más refenciados, sin embargo esto trae consigo que nunca se sabe con seguridad la cantidad de memoria física disponible en un tiempo determinado, es más se puede crear espacio para los objetos pero no liberarlo por programa como sucede con C++, ya que a pesar de que hacer esto proporciona un alto grado de predictibilidad, constituye un grado muy bajo de seguridad para el sistema, es por ello que RTSJ para tener un balance entre seguridad y predictibilidad implementa una recolección de basuras semi-automática. Por otra parte se puede determinar el tiempo de vida de un objeto, ya sea a través de la codificación, es decir de manera manual, o también de manera automática como normalmente lo hace Java o también por su ámbito de sintaxis.
- Creación y control de hilos en tiempo real. Los hilos con RTSJ pueden ser creados a partir de la clase RealtimeThread, cuyo constructor puede recibir parámetros para: la utilización de memoria que hará el hilo, para la definición de un conjunto de tareas acíclicas, para la planificación a corto plazo, para definir el comportamiento del hilo.
- Manejo de eventos asíncronos. RTSJ cuenta con un mecanismo que permite el enlace de algún evento, el cual puede ser externo o interno, con objetos que pueden ser planificados.
- Planificación a corto plazo. Se manejan 28 niveles de prioridad y se permite la definición del algoritmo de planificación, a través de la clase Scheduler.
- Creación y control de relojes temporales. En RTSJ se permite la creación de relojes a los cuales es posible asociarles funciones como si se tratase de eventos asíncronos.
- Control de recursos compartidos y Sincronización. Con RTSJ se puede implementar tanto una espera y como una notificación que sean deterministas, de tal forma que cuando un hilo desee acceder a un bloque de código sincronizado y éste se encuentre siendo utilizado por otro hilo, el hilo será colocado en una fila ocupando el lugar que le corresponda de acuerdo a su prioridad.
- Transferencia asíncrona de Control. En RTSJ un hilo puede ejecutar una excepción sobre otro hilo, con lo cual se permite que flujo normal de ejecución pueda ser manejado de forma asíncrona.

3.2.3 Arquitectura de referencia para sistemas móviles con conexión a Internet.

Las tecnologías inalámbricas desde un principio han tenido un gran éxito, una de sus mejores características es que su evolución ha sido persistente. Actualmente las tecnologías inalámbricas alcanzan velocidades de hasta de 70Mbps de velocidades de transmisión a distancias de 50 km lejos de su estación base, gracias a la implementación de el estándar 802.16a de la IEEE.

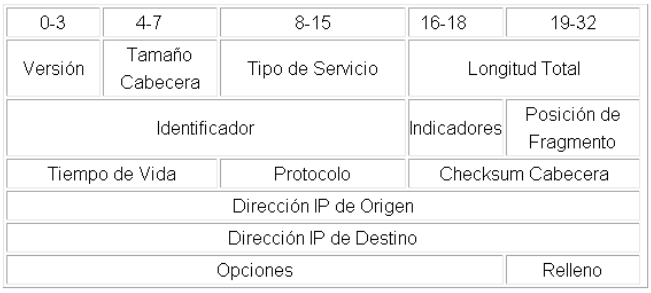
Por su parte Internet que es un conjunto de servicios que se expanden a través de redes WAN alrededor del mundo y hace uso de los protocolos TCP e IP, que fueron desarrollados específicamente para ella.

El TCP o protocolo de control de transmisión, es un protocolo orientado a la conexión que pertenece a la capa de Transporte y que como su nombre lo indica, garantiza que los datos serán entregados en el mismo orden en que fueron enviados y sin errores, en la figura 85 puede observarse la cabecera de este protocolo, que debido a que debe establecer en ella todos los mecanismos del protocolo su tamaño mínimo debe ser de 20 bytes

Figura 85: Cabecera de TCP

El IP o protocolo de Internet es un protocolo no orientado a conexión, En este protocolo los datos enviados a través de él lo se envían en paquetes llamados datagramas, de tal forma que un mensaje si su tamaño es grande puede ser divido en datagramas y estos ser enviados por diferentes rutas, será responsabilidad del destino acomodarles en el orden en que deben estar para que el mensaje tenga sentido, y por otra parte no hay garantía en que el mensaje llegue completo o en buen estado a su destino. Actualmente Ip cuenta con dos versiones la versión 4 con un tamaño de 32 bits, que dada la expansión que ha tenido Internet ya no es suficiente para proporcionar distintas direcciones a todos los dispositivos que requieren tener acceso a Internet y por eso se creo la versión 6 cuyo

tamaño es de 128 bits, En la figura 86 a) y 86 b) se muestran las cabeceras para cada una de estas versiones.



0-3	4-7	8-15	16-18	19-32
Versión	Tamaño Cabecera	Tipo de Servicio	Longitud Total	
Identificador			Indicadores	Posición de Fragmento
Tiempo de Vida		Protocolo	Checksum Cabecera	
Dirección IP de Origen				
Dirección IP de Destino				
Opciones				Relleno

Figura 86 a) Cabecera IP versión 4

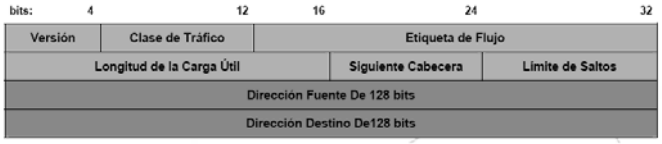


Figura 86 b) Cabecera IP versión 6

Gracias a estos dos protocolos TCP/IP se pueden proveer servicios tanto orientados a conexión como el de dar soporte al protocolo HTTP usado para las páginas Web, el SMTP utilizado para el correo electrónico y el FTP para la transferencia de archivos entre otros.

Cuando el uso de alguna tecnología comienza a expandirse se hace necesaria la implementación de un conjunto de reglas que normalicen el uso de dichas tecnologías con el propósito de llegar a una unificación de criterios que lleven a un mejor entendimiento de las mismas, para conseguir un mayor nivel de penetración o ampliación de su uso.

El modelo OSI por ejemplo fue concebido para tratar de regular la incompatibilidad entre sistemas operativos en cuanto a la conectividad con otras máquinas, procurando que se tuviesen sistemas que permitiesen la interoperabilidad entre computadores de cualquier fabricante, por lo que es utilizado como un marco que sirve de referencia para el diseño de protocolos de interconexión entre sistemas abiertos, normando exclusivamente la interacción entre los sistemas conectados, pero no la estructura interna ni su relación con el sistema operativo. Por lo cual sirve de base para describir la funcionalidad tanto de dispositivos como de protocolos en una red y a la vez es flexible para considerar las necesidades de los protocolos modernos.

Considerando entonces al modelo OSI como la base; los protocolos de Internet quedan ubicados en la capa 3, la capa de red, en específico el protocolo IP, debido a que se encarga de recibir bits de la capa inferior (la capa de enlace a datos) ensambla los bits en paquetes conocidos como datagramas y selecciona la mejor ruta de acuerdo a una métrica. Y dado que IP es sin conexión, cada datagrama debe contener la dirección del nodo destino, la cual es conocida por lo mismo como dirección IP.
También los protocolos de Internet son colocados en la capa 4, la capa de transporte, específicamente los protocolos TCP y UDP. El protocolo de Control de transmisión el cual proporciona una transmisión confiable de datos mediante detección y corrección de datos extremo a extremo, garantizando que los datos serán transferidos a través de la red de manera exacta y en el orden apropiado y retransmitiéndolos en caso de sólo de no ser recibidos, evitando la duplicación de los mismos. Por su parte el protocolo de datagrama del usuario UDP, proporciona un servicio de datagrama no confiable, ya que no existe ni detección ni corrección de error extremo a extremo, tampoco hay retransmisión, ni control de flujo o error, sin embargo se desempeña más rápido que el TCP.

A su vez cada uno de estos protocolos da soporte a un conjunto de servicios manejados por protocolos de aplicación, por lo que para el TCP contamos con el protocolo de manejo de terminales (Telnet), el Protocolo de Transferencia de Archivos (FTP), el Protocolo para Transferencia Simple de Mensajes (SMTP), y el Protocolo de Oficina Postal (POP) y el Protocolo de Acceso a Mensajes en Internet (IMAP).

Por su parte UDP da soporte a los protocolos como el Protocolo de Transferencia de Archivos Trivial (TFTP) el Protocolo de Administración de Red Simple(SNMP), al Protocolo Bootstrap (BOOTP), y al sistema de archivos de Red (NFS) y al Servicio de Nombres de Dominio(DNS)

En la terminología de Internet, un servicio es un conjunto de operaciones que se ejecutan a petición de un usuario, y un protocolo es un conjunto de normas que rigen el formato de los mensajes para que se pueda dar el servicio, de manera que si bien el mismo servicio se puede proveer usando protocolos distintos, el usuario no lo nota.

El principio de funcionamiento de los servicios de Internet es el de “cliente-servidor”. Un “cliente” es un programa que el usuario ejecuta desde su Terminal para acceder a un servicio. Un “servidor” es un programa que se ejecuta desde otra computadora en otra parte de la red que proporciona el servicio solicitado. A esta forma de ver a Internet como una red única de computadoras que desempeñan el papel de clientes y servidores se le define como arquitectura de red cliente-servidor.

Servicios básicos

Los servicios básicos de Internet están disponibles para cualquier usuario. Estos servicios se mencionan a continuación:

a) Terminal virtual o remoto (TELNET)

El protocolo TELNET permite a una terminal local usar los recursos de otra computadora remota como si fuera desde su propia ubicación. La computadora remota en cuestión debe ser multiusuario en el sentido de que deba permitir la operación simultánea de varios usuarios.

b) Transferencia de archivos (FTP)

Mediante el servicio FTP (*File Transfer Protocol*) la terminal del usuario ejecuta un programa cliente FTP que le permite el acceso a los directorios de una computadora remota llamada servidor FTP y así poder descargar sus archivos.

c) Correo electrónico (SMTP)

Es el servicio más utilizado de Internet. En el e-mail se pueden adjuntar archivos que contengan documentos, imágenes, video, etc. El protocolo que permite el intercambio de mensajes y que sólo se usa para enviar correo es el SMTP (*Simple Message Transfer Protocol*), aunque los más utilizados para manipular correo son el POP (*Post Office Protocol*) y el IMAP (*Internet Message Access Protocol*). El POP es el más antiguo y está diseñado de tal forma que la terminal cliente solicita del servidor todo el correo que éste tiene almacenado. A este tipo de acceso al correo se dice que es *offline.* El IMAP, en cambio, está diseñado para un acceso *online* del correo: el cliente no descarga todo el correo sino que los mensajes permanecen en el servidor hasta que el usuario los elimine.

d) Noticias (NNTP)

El servicio es similar a un tablón de anuncios o grupo de discusión. Se pueden leer, poner o contestar mensajes. Como en cualquier tablón de anuncios, el servicio tiene las siguientes características:

- Se sitúan artículos en un tablón público y por ello no necesita inscripción previa.
- Se ordenan por temas.
- Se envían a un grupo en particular

- Dentro de un grupo se ordenan en hilos para que un tema de debate se pueda seguir sin mezclarse con otra

e) Acceso y navegación por páginas Web (HTTP)

Éste servicio se ha popularizado en Internet como el mejor sistema de acceso y consulta de información. Es el más utilizado después del servicio de correo electrónico, y el que más recursos consume. El servicio Web, como se le conoce coloquialmente, facilita una interfaz para usuarios con escasos conocimientos informáticos mediante un programa cliente llamado navegador (*browser*).

El servicio se basa en la existencia de servidores Web, los cuales proporcionan información a los usuarios de la misma forma que se puede consultar un libro, pero de forma que algunos elementos en esas páginas dan acceso a otras nuevas páginas que pueden o no estar en el mismo servidor (hiperenlace). Además, para aumentar aún más la sencillez en el manejo, desde este servicio se puede acceder a los demás servicio de Internet. Los elementos que caracterizan al servicio Web son:

- El “hipertexto” o “hipermedia”.
- El lenguaje HTML.
- Localizadores de recursos.

El hipertexto es información textual de naturaleza no secuencial, es decir que no tiene un orden específico en el que se pueda leer la información de la página. La inexistencia de ese orden en el hipertexto le da una mayor libertad al usuario de elegir a su conveniencia la opción que más interese.

Si bien el protocolo HTTP (*HyperText Transfer Protocol*) permite el intercambio de información entre clientes y servidores, es necesario además un lenguaje común para escribir páginas Web en forma de hipertexto. A éste lenguaje común se le conoce como lenguaje HTML (*Hypertext Markup Language*). A este lenguaje se le caracteriza por usar etiquetas (*tags*) para definir lo componentes que conforman el documento. Las distintas etiquetas pueden ser dibujos, títulos, listas, formatos, enlaces a otros documentos, etc. El HTML es un subconjunto de SGML (*Standard Generalizad Markup Language*), un metalenguaje que permite la descripción formal de un “lenguaje de marcas”. Un lenguaje de marcas es un conjunto de etiquetas o marcas que permiten formatear textos para su presentación.

f) Comunicación interactiva (IRC)

Popularmente conocido como “*chat*", facilita la comunicación a distancia con otras personas mediante la utilización del teclado. Este servicio permite que varias personas de cualquier parte del mundo se comuniquen entre sí en directo. Los c*hat* permiten organizar reuniones en las que se tratan temas específicos para que sean discutidos entre todos los participantes. El protocolo más extendido que facilita el uso del servicio es el IRC (*Internet Relay Chat*).

Comunicaciones corporativas basadas en IP

Todos los servicios de Internet anteriormente mencionados pueden ser prestados en una red privada que cumpla los protocolos TCP/IP, dando lugar a lo que se denomina como una Intranet. Cuando se autoriza a un grupo de usuarios externos a la Intranet de una corporación se dice que se ha creado una Extranet. En el momento de la creación de esta tipo de redes IP, se suele dar acceso a cualquier usuario de Internet con las medidas de seguridad adecuadas.

a) Normas de seguridad

Para que los usuarios den un uso correcto de los recursos que ofrecen las redes IP se utilizan normas de seguridad que satisfacen los requerimientos de seguridad tales como el cifrado, la autenticación y la integridad. Entre las normas de seguridad más destacadas para redes IP están:

-PGP: Es un programa de cifrado de clave pública que permite cifrar el contenido del correo electrónico.
-Kerberos: Desarrollado por el MIT a finales de los 80, permite la autenticación de usuarios mediante claves públicas
-RADIUS (*Remote Authentication Dial-In User Service*): Ésta norma está orientada a la seguridad de accesos remotos y permite la realización de funciones tales como autenticación, autorización y contabilidad.

b) Seguridad mediante cortafuegos (*firewall*)

Un cortafuego monitorea y filtra todo el tráfico en una red corporativa para garantizar la seguridad entre la red interna de la empresa y las redes externas, mediante tres tipos de filtrado:

-IP: Se analiza la información en las cabeceras de los paquetes IP, en cuanto a las direcciones IP de origen y destino de la información.

-Aplicación (*proxy*): Consiste en el bloqueo de todo el tráfico a nivel IP entre la red interna e Internet. El *Proxy* actúa como intermediario de la comunicación, comprobando los permisos de los clientes, y en su caso, realizando la conexión al servidor remoto de Internet.
-Conexión: A diferencia de los tipos de filtrado anteriores, el filtrado a nivel de conexión no reside en el tratamiento sobre los datagramas IP, sino en el control de la conexión entre un cliente y un servidor. No es posible el monitoreo de los datos intercambiados entre el cliente y el servidor, pues una vez establecida la conexión, el cortafuegos actúa de forma transparente.

Nuevos Servicios IP

Además de los servicios básicos de Internet antes mencionados, su creciente convergencia con otros servicios tales como la TV terrenal, por satélite y por cable, y con los servicios de voz, que se ha llegado a considerar a las redes IP como otra forma de acceso a información multimedia.

a) Voz sobre IP (VoIP)

Las redes de datos que cumplen con el protocolo IP pueden transmitir señales telefónicas, ya sea entre teléfonos convencionales, entre los PC o comunicaciones mixtas entre teléfonos y PC. En general, cualquier usuario de Internet con una PC, un micrófono y el programa adecuado puede intercambiar mensajes de voz con otro usuario. Pero la telefonía de Internet presenta problemas importantes tales como el retardo, la variación del retardo y la posible pérdida de paquetes de información. Un retardo por encima de los 250 ms es percibido como una pérdida de calidad. Por ésta razón se prefiere hablar por voz sobre IP más que voz sobre Internet.

Entre las normas internacionales de VoIP (*Voice over IP*) están la H.323 y la SIP (*Session Initiation Protocol*). La norma H.323 permite la creación de una red multimedia con capacidad de conexión mediante *gateways* a diferentes tipos de redes de acceso tales como redes celulares (GSM), red telefónica conmutada (RTC), red digital de servicios integrados (RDSI) o redes corporativas de empresa. Los dispositivos que soportan a la norma H.323 son los siguientes:

- *Gateway*: Proporciona comunicaciones bidireccionales en tiempo real entre terminales H.323 y otras terminales con otras normas de red.
- *Gatekeeper*: Equipo opcional que registra direcciones y proporciona traducción de direcciones. Controla los accesos a la red y el uso del ancho de banda.

- Controlador Multipunto (MC): Efectúa el control para tres o más terminales en una conferencia multipunto. No puede conmutar audio, video o datos.
- Procesador Multipunto (MP): Centraliza el procesado de audio, video y datos en conferencias multipunto. Realiza las conmutaciones bajo el control del MC.
- Unidad multiconferencia (MCU) Tiene la capacidad de que participen tres o más *gateways* en una misma conferencia multipunto. Contiene un MC y opcionalmente un MP.

Parte del conjunto de protocolos que configuran la norma H.323 se resumen a continuación:

-Nivel 5 (sesión)

- H.225: Establecimiento de llamada.
- H.245: Conmutación y creación de canales lógicos
- H.450: Servicios complementarios.
- RAS (*Registration, Administration and* Status): Crea un canal lógico entre el usuario y el *gateway* para solicitar recursos para la llamada.
- RTCP (*Real-time Transport Control Protocol*): Controla la calidad de la sesión.

-Nivel 4 (transporte)

- RTP (*Real-time Transport Protocol*): Transporta el tráfico en tiempo real de datos multimedia.

-Nivel 3 (red)

- Q.931: Mensajes de señalización para establecer y terminar llamadas.

Dada su orientación para procesar información multimedia, se presentan a continuación las normas de codificación de audio y video para la norma H.323 (año/velocidad):

- G.711 (1998/64Kbps): Modulación en el margen de frecuencias de voz.
- G.722 (1988/64Kbps): Codificación de audio por subbandas de 7Khz de ancho de banda
- G.723.1 (1996/5.3 y 6.3Kbps): Predicción lineal excitada por código.
- G.728 (1992/16Kbps): Predicción lineal excitada por código de bajo retardo.
- G.729 (1996/8Kbps): Predicción lineal algebraica de estructura conjugada excitada por código.

Entre las normas de codificación de video para la norma H.323 están:

- H.261 (1990): Para velocidades nx64 Kbps que soporta formatos de imagen QCIF y opcionalmente CIF.
- H.263 (1996): Soporta formatos de imagen SQCIF y QCIF y opcionalmente CIF, 4CIF y 16 CIF.

Los equipos que pueden establecer una comunicación con terminales H.323 son:

- V70: Equipos de transmisión simultánea de voz y datos sobre RTC.
- De voz: Teléfonos convencionales conectados a RTC y digitales conectados a RDSI de banda ancha.
- H.310: Terminales y sistemas de comunicación audiovisual.
- H.320: Terminales de videotelefonía.
- H.321: Adaptación de los videoteléfonos a entornos RDSI de banda ancha.
- H.322: Videoteléfonos para las LAN.
- H.324: Terminales de comunicación multimedia de velocidades bajas.

Las recomendaciones (ITU-I) para la transmisión de fax sobre redes IP son las denominadas T.37 y T.38

-Recomendación T.37

Es principalmente una referencia para las RFC que describe métodos de transferencia y codificación. Existen dos modos de su funcionamiento:

- Modo simple: Solo envía el mensaje codificado. No hay negociación entre las terminales.
- Modo completo: Las terminales intercambian capacidades y mensajes normalizados.

-Recomendación T.38

Permite la comunicación en tiempo real entre terminales de fax. Se utilizan dos tipos de paquetes:

- Indicación: información del estado de transporte de la llamada.
- Datos

Los procedimientos T.38 trabajan con H.323 para:

- Control de llamadas
- Selección de protocolos.
- Transferencia de direcciones de la parta llamada.
- Indicación del progreso de la llamada.

b) Otras normas VoIP

Si bien la aparición de la norma H.323 produjo un importante desarrollo en la industria relacionada con las aplicaciones VoIP, tiene ciertas limitaciones:

- Difícil escalabilidad, ya que un *gateway* asume la conversión de señales y el control de las llamadas. Esto limita el número de usuarios por *gateway*.
- Debido a la ausencia de mecanismos de recuperación del *gateway* en caso de un fallo, la disponibilidad de la red VoIP queda comprometida.

Sin embargo, en el 2000 se introdujo la norma MEGACO. Esta norma elimina los problemas de *gateway* de la H.323 dividiéndola en los siguientes componentes:

- SG (*Signaling Gateway*): Proporciona la función de señalización entre los dominios IP y RTSC
- MG (*Media Gateway*): Traduce medios del dominio IP al dominio RTC.
- MGC (*Media Gateway Controller*): Se sitúa entre el MG, el SG y el *gatekeeper*. Recibe toda la información de señalización RTC desde el SG y la señalización IP desde el *gatekeeper*. Por otra parte, el MGC proporciona la función de procesamiento de llamada al MG, y mantiene la información necesaria del estado de la llamada.

Por su parte la telefonía celular ha brindado el acceso a Internet de banda ancha a través de su tecnología 3G basada en WCDMA o Acceso múltiple por división de código de banda ancha, con velocidades de hasta 3.1 Mps, dependiendo del proveedor de servicio, hace uso del protocolo IP con conmutación por paquetes y con una conexión no dedicada, por lo cual los paquetes pueden ser duplicados e incluso perderse.
En un futuro no muy lejano se espera la telefonía 4G, con la cual se prometen velocidades de hasta 1 Gbps además de proveer calidad de servicio QoS, de Terminal a Terminal con lo cual se contará con

un alto grado de seguridad, que garantice la transmisión de cierta cantidad de datos en un tiempo determinado.

En cuanto al software en CORBA inicialmente no se contaba con soporte a aplicaciones móviles, ni de tiempo real, pero el grupo OMG creo un grupo de extensiones para el desarrollo de aplicaciones para sistemas móviles como por ejemplo Minimun CORBA y la Wireless CORBA, diseñadas y pensadas específicamente para PDA's y teléfonos celulares. Entre las implementaciones de estas especificaciones se tiene a LUAOrb y MICO.

Minimum CORBA

Esta especificación conserva los puntos clave de CORBA como son: la portabilidad de las aplicaciones y la interoperabilidad entre ORBs. Minimum CORBA aunque soporta sólo un subconjunto de interfaces y políticas definidas en el adaptador de objetos portátiles, mejor conocido como POA, no impide el mantener la interoperabilidad entre ORBs, pero no cuenta con activación dinámica de POA. Por otra parte se omiten en MinimumCORBA interfaces como la interfaz de invocación dinámica y la interfaz de Skeleton dinámica.

Wireless CORBA

La arquitectura de wireless CORBA identifica 3 dominios, los cuales son:

El dominio Home: Este dominio para una Terminal dada, es el encargado de hospedar al Agente Localizador del Home, el cual se encarga de mantener un rastreo de la posición actual de la terminal para lo cual está provisto de operaciones para la consulta y actualización de la posición de la terminal y de los puentes de acceso a los que el dispositivo móvil ha accedido a lo largo de su recorrido.

El dominio Visited: Por su parte este dominio es el encargado de hospedar a los puentes de acceso que proveerán de ORB a los dispositivos móviles.

El dominio Terminal: Este dominio consiste de un dispositivo que hospeda un ORB y un puente Terminal a través del cual los objetos sobre la terminal pueden comunicarse con otros objetos en otras redes.
La señalización entre dominios es llevada a través de túneles GIOP. Los mensajes son transmitidos entre el puente terminal y el acceso terminal.

Por otra parte en cuanto a recomendaciones para la construcción de aplicaciones para móviles la organización W3, establece 10 recomendaciones las cuales son las siguientes:

- **Diseñe** para una Web única
- **Confíe** en los estándares Web
- **Evite** los riesgos conocidos
- **Sea prudente** con las limitaciones de los dispositivos
- **Optimice** la navegación
- **Compruebe** gráficos y colores
- **Hágalo** en pequeño
- **Economice** el uso de la red
- **Facilite** la entrada de datos
- **Piense** en los usuarios de la Web móvil

3.2.4 Arquitectura de referencia para sistemas de información.

Como ya se ha mencionado anteriormente una arquitectura de referencia sirve como un marco que contiene un conjunto de guías que aconsejan la mejor manera de construir software, dicho de otra manera es un mapa de referencia para la planificación estratégica. En este caso para los Sistemas de Información. Las metas primordiales para los cuales se requiere el tener una arquitectura de referencia de sistemas información son las siguientes:

- Contar con un modelo que abarque todos los sistemas de información y las aplicaciones
- Contar con una estrategia general para los futuros desarrollos
- El adecuar los sistemas tanto a las más recientes metas organizacionales como a la tecnología.
- Reducir costos en la implementación de las tecnologías emergentes.
- Tener un horizonte claro a corto, mediano y largo plazo.

Los componentes principales de una arquitectura son el modelo de funciones, el modelo de datos, el modelo de flujos de información, dentro de los cuales la Decisión, Gestión y Producción son las principales actividades a tener en cuenta. Sin perder de vista por supuesto la infraestructura sobre la que las arquitecturas estarán montadas, por lo que los servidores, las redes y las comunicaciones, los sistemas operativos y manejadores de datos, deben ser considerados para que los sistemas resulten

además de productivos, confiables y seguros.

Sin embargo el problema real a lidiar es el de adecuar sistemas antiguos a las nuevas tecnologías, y más que los sistemas sus bases de datos, ya que es la información de estos la que realmente interesa. Para lidiar con este problema las nuevas arquitecturas de referencia hacen uso de patrones y estilos arquitectónicos que proyectan los elementos de software y los flujos de datos entre ellos, de tal manera que se permite el tener el diseño del sistema a un alto nivel antes de iniciar su implementación. Obteniendo un modelo arquitectónico, él cual será continuamente refinado al ser ajustado al cumplimiento de los requisitos funcionales.

Por otra parte en cuanto a las bases de datos empresas como Sun y Sysbase proporcionan soluciones a nivel de hardware como la de su Arquitectura de Referencia para Almacenes de Datos Empresariales iForce (Enterprise Data Warehouse Reference Architecture) la cual está basada en servidores y sistemas de almacenamiento de Sun y en Adaptive Server IQ Multiplex de Sybase, que pueden almacenar hasta 48,2 terabytes de datos de entrada en bruto, algo así como 179.000 millones de filas que se guardan haciendo uso de un diseño de almacén de datos realista y con la utilización de sólo 22 terabytes de almacenamiento total al tener una compresión de datos de 0,46, proporcionando además un rendimiento en consultas que se cuyo incrementó es de más de un 94 por ciento al contar con múltiples nodos servidor y con una carga de datos con un retrasó de menos de un 7 por ciento, al mismo tiempo que se pueden ejecutar consultas concurrentes.

La Arquitectura de Referencia demuestra un alto rendimiento y escalabilidad, ya que su capacidad para mantener el rendimiento de consultas en un solo nodo cuando se incorporaron on-line dos nodos adicionales que efectúan más consultas concurrentes contra la misma base de datos, no se ve en lo más mínimo disminuida., al contrario el rendimiento aumenta para procesar más carga de trabajo ya sean estos usuarios o consultas, a medida que se van añadiendo recursos adicionales, sin que la aplicación deje de funcionar.

.

En cuanto al software existen metodologías y especificaciones como la MDA (Model Driven Architecture) o Arquitectura Basada en Modelos, la cual como ya se había mencionado con anterioridad fue desarrollado por el grupo OMG.

La idea de la arquitectura basada en modelos es la de que exista una interconexión y funcionamiento en conjunto de una manera compatible entre los distintos sistemas apoyándose en los estándares del grupo OMG como: MOF, XMI, CWM y por supuesto UML. De tal forma que los modelos que queden así expresados en una notación bien definida serán fundamentales para el buen entendimiento de los

sistemas a nivel organización y una construcción de los mismos con un apoyo formal a la descripción de modelos, en un conjunto de meta modelos, hará fácil la integración y transformación entre los mismos, dando como resultado una base para la automatización mediante herramientas, y sobre todo al contar con la aceptación de un desarrollo basado en modelos se motivará a tener estándares industriales y por lo tanto comerciales.

Para la arquitectura basada en modelos, un sistema debe contar con tres modelos relacionados pero diferentes. El primero de ello es el modelo independiente de cálculo llamado CIM (Computation Independent Model), el cual hace una descripción de los procesos de negocio que debe solventar el sistema con un enfoque independiente de que dichos procesos serán ejecutados por un sistema computacional, es por ello que a este modelo es común llamarlo modelo del dominio o modelo de negocio. En este modelo se crean y organizan los requisitos que el sistema debe cumplir. Los cuales forman un diccionario fácilmente compartido por los otros dos modelos. La función de CIM es la de servir como conducto entre los analistas o ingenieros de requisitos y los desarrolladores de las componentes de software.
El segundo modelo es el Modelo Independiente de Plataforma, llamada PIM (Platform Independent Model) y en la que el sistema se materializa, es decir, en este modelo se plasma la lógica del sistema, así como la forma en que este interacciona con el mundo exterior, pero con la condición que el nivel de abstracción plasmado en él sea tal que no importando la plataforma de desarrollo y la de su implementación, siempre permanezca igual.

El tercer modelo es el Modelo Específico de la Plataforma, conocido como PSM (Platform Specific Model), que a su vez está conformado por los modelos PM (Plataform Model) o modelo de plataforma, el cual especifica la forma en la cual el sistema utilizará una plataforma en particular y el modelo ISM (Implementation Specific Model) o Modelo Específico de Implementación, que se encarga de armonizar las especificaciones realizadas en PIM con los detalles vistos en PM.

En resumen MDA, trata de que los requisitos especificados en el modelo CIM sean transformables en PIM y posteriormente en PSM y viceversa.
Cabe señalar que MDA únicamente proporciona una estrategia general a seguir en el desarrollo de software, más no intenta definir fases del proceso ni tampoco técnicas, es sólo una especificación de desarrollo.

Por otra parte dadas las tendencias cada vez más fuerte hacia la utilización de la Internet, en un futuro no muy lejano casi todos los sistemas de información se encontrarán en ella, es por ello que las arquitecturas que más se han ido imponiendo son las arquitecturas SOA o Arquitecturas Orientadas a

Servicios. Las cuales son un modelo para la estructuración y uso de componentes distribuidas que pueden ser propiedad de diferentes dominios que las tienen bajo su control, y que sin embargo SOA permite una plataforma homogénea para descubrir, interaccionar y usar dichas componentes.
Conforme a SOA, las componentes y sus interfaces deben ser publicadas a través de un registro, al cual puede tener acceso, cualquier consumidor, de tal manera que las dependencias entre diferentes servicios sean mínimas, es decir, que se tenga un esquema de bajo acoplamiento.
Por otra parte los servicios pueden ser modulares, es decir, servicios más complejos pueden ser construidos tomando como base servicios más simples. Es por ello que los servicios deben ser lo suficientemente independientes como para no provocar la modificación de otros sistemas o servicios que quieran hacer uso de ellos, sólo deben encargarse de su propia funcionabilidad, la cual por supuesto deber estar encapsulada.
La condición que estos servicios deben cumplir es la de poderse implementar, gestionar y utilizar, por cualquier desarrollador, proveedor o usuario de una manera simple y más accesible de lo que actualmente se hace con los paquetes de software tradicional.

3.2.5 Arquitectura de referencia para ambientes virtuales de aprendizaje.

Otra de las aportaciones del uso del Internet, es el de los sistemas virtuales de aprendizaje la cual es una modalidad para que un individuo pueda obtener conocimientos para su propio análisis y reflexión.
Sin embargo no involucra exclusivamente una aplicación de software, de hecho un ambiente virtual de aprendizaje está compuesto de varios elementos físicos como son los que conforman el entorno donde el estudiante recibe los contenidos educativos (computadoras, módems, elementos de red, software de apoyo como : pizarra electrónica por ejemplo etc.,), los cuales son sometidos a una evaluación por parte de un asesor con el cual mantiene una comunicación a distancia a través de los medios de comunicación, en este caso a través de la Internet. También deben considerarse al grupo de académicos especialistas en cada tema, el contenido o programa académico que se piensa cubrir, los pedagogos que revisan y reestructuran los contenidos para que sean fáciles de asimilar por parte del estudiante, los expertos en tecnologías de información, encargados de administrar la red e instalaciones que dan soporte al ambiente virtual, etc.,

En sí un ambiente de aprendizaje virtual lo conforman cinco entornos:

Gestión: puede estar o no implementado vía software, prácticamente el propósito de este entorno es llevar un registro y control de todos los trámites administrativos involucrados en un ambiente

académico, como: el trámite de inscripción o reinscripción, registro y control de pagos de colegiatura, registro de calificaciones, historial académico, oficios de certificación de estudios, etc.,

Experimentación: también puede o no ser implementado vía software, ya que involucra el que el alumno pueda llevar a cabo experimentos recomendados en el material didáctico, algunos de los cuales son factibles de realizarse a través de simuladores, sin embargo otros requerirán elementos físicos encontrados únicamente en insolaciones físicas reales.

Asesoría: Este entorno si se encuentra implementado vía software, aunque esta implementación requiera el uso de software de terceros, como podría ser el uso de chat's y correos electrónicos ajenos a la aplicación, que se conocen como comunicación sincrona y asíncrona respectivamente. La cual es necesaria cuando el alumno requiere asesoría de su asesor sobre cualquier cuestión relacionada con el material didáctico proporcionado.

Colaboración. Por lo general un ambiente virtual de aprendizaje debe promover una interacción intergrupal, para que los alumnos se comuniquen entre sí, para lo cual se debe proporcionar un entorno de comunicación como son los foros, o videoconferencias y también comunicación a través de correo electrónico.

Conocimiento. Este entorno es la esencia del ambiente virtual del aprendizaje ya que involucra la interacción interpersonal del estudiante consigo mismo a través de la reflexión y el análisis de los conocimientos otorgados a él por la aplicación a través de un conjunto de elementos de software como archivos con contenidos escritos y multimedia.

Es por ello que una arquitectura de referencia de un ambiente virtual puede estar constituida por diferentes capas cada una de ellas con componentes de software que cubran las necesidades de cada uno de estos cinco entornos.

Desde punto de vista práctico se pueden utilizar como referencia los desarrollos de sistemas colaborativos, un ejemplo de este tipo de sistema es el desarrollado por la empresa mexicana Tralcom, el cual hace uso de herramientas colaborativas como los chats, pizarrón electrónico, videoconferencias y foros y que hace uso del protocolo de comunicación segura SSL y de servidores de Microsoft como: el MS SQL Server y el MS Transaction Server.

3.2.6 Arquitecturas de referencia para líneas de productos.

Los sistemas para el control de producción de líneas de productos requieren conocimientos del funcionamiento y manejo de dispositivos electromecánicos, así como de controladores lógicos programables, mejor conocidos como PCL's, dichos controladores abundan en la industria. Actualmente son capaces de ejecutar operaciones aritméticas y manipular señales analógicas, que les permite controlar el funcionamiento tanto de máquinas, como de plantas y procesos industriales, además de poder establecer comunicación con otros PCL's y estaciones de trabajo conectadas en redes LAN. Su programación cubre un amplia gama de necesidades que van desde la aritmética booleana hasta el manejo de funciones de multiprotocolo que les permite comunicarse con otros dispositivos.

Uno de los modelos más utilizados en el desarrollo de este tipo de aplicaciones es el modelo CIM (Computer Integrated Manufacturing) o Manufactura Integrada por Computadora, el cual fue desarrollado con la idea de incrementar la eficiencia en la planeación de los insumos de manufactura y recientemente en la producción justo a tiempo, conocida por sus siglas en inglés como JIT(Just In Time). La cual resulta muy beneficiosa ya que uno de sus axiomas es el de producir lo que y cuando se necesita, para lo cual se basa en la reducción de inventarios, especialmente de inventarios de productos en proceso, lo que ocasiona una buena reducción en el coste de inventario.

Además del módulo de planeación de insumos, CIM cuenta con módulos para la planeación y organización de las actividades de manufactura, que proporcionen mejores alternativas para la producción y los insumos, módulos que monitorean si dichas actividades se ajustan al plan previo y además módulos que permitan predecir o proyectar los resultados incluyendo resultados financieros.

Dentro del ámbito estrictamente computacional, CIM cuenta con una amplia gama de componentes de hardware y de software incluidas las telecomunicaciones.

Para manejar el control de los diferentes entornos de manufactura, CIM prevé cinco niveles que conforman una jerarquía de control y que son los siguientes de abajo hacia arriba en la jerarquía:

- Controladores lógicos Programables. En este nivel el inferior de todos, se encuentran todos los dispositivos electrónicos encargados de controlar el funcionamiento de las máquinas.
- Control de celdas. Este nivel está conformado por un conjunto de máquinas con tareas independientes pero que laboran en grupo, controladas por un computador central.
- Computador de área. Por su parte este nivel se encarga de monitorear todas las operaciones

dentro de una línea de producción en específico

- Computador de planta. Este nivel cuenta con un computador que se encarga de ejecutar operaciones administrativas, como la supervisión y división de tareas.
- Computador corporativo. El nivel más alto cuenta con el servidor de base de datos y cn programas tanto administrativos como financieros.

Por otra parte las comunicaciones entre los niveles es llevada a cabo bajo protocolos de comunicación como el protocolo MAP ((Manufacturing Automation Protocol) y el protocolo TOP (Technical and Office Protocol), los cuales permiten la transmisión de datos entre programas o procesos en la red interna lo que hace MAP, mientras que TOP se encarga de especificar el procesamiento de información en ambientes técnicos y de negocios.

CIM se apoya en sistemas de información como son:
CAM (Computer Aided System). Esta hace uso de la robótica y el control numérico.
CAD (Computer Aided Design). Utiliza todas las técnicas y elementos para el dibujo y diseño de productos.
CAE (Computer Aided Engineering). En el cual se utilizan técnicas para simular procesos de fabricación.
FMS (Flexible Manufactured System). Apoya en el cálculo numérico usado en la ingeniería mecánica.

Preguntas

1. Explique que es una componente de software.
2. ¿Cuáles elementos conforman un diagrama de componentes?
3. ¿Qué representa un diagrama de despliegue?
4. ¿Qué es un patrón de diseño?
5. ¿Qué significa el término de granularidad?
6. ¿Qué es DCOM?
7. ¿En que consiste EJB?
8. ¿Qué es un ORB?
9. ¿qué un cliente stub?
10. ¿Qué se conoce como Servicios Web?
11. Describa que es un sistema en tiempo real y los componentes que lo conforman.
12. Considere el ejercicio 8. Enfoque su atención a los diagramas de estado ¿Puede ser considerado un sistema en tiempo real?, de ser así a que clasificación corresponde?
13. ¿Qué elementos conforman la cabecera IP V6?
14. ¿Cuál es la razón de haber diseñado el modelo OSI?
15. ¿Cuáles son las principales diferencias entre el protocolo UDP y el TCP?
16. ¿Cuál es el protocolo utilizado en el correo electrónico?
17. ¿En que tecnología se basa la telefonía celular 3G?
18. Describa los modelos que conforman la Arquitectura Basada en Modelos.
19. ¿Cuáles son los entornarnos que conforman un ambiente de aprendizaje virtual.
20. ¿Qué es Manufactura Integrada por Computadora?

Ejercicios

1. Visualice alguna aplicación que involucre a diferentes tipos de usuarios y de preferencia distribuida a lo largo de varios equipos computacionales, en su defecto puede considerar como sería una aplicación de su escuela como la que lleva el control de calificaciones o la administración del proceso de titulación, también podría conseguir alguna aplicación que coste de varias librerías y módulos elaborada con un entorno de desarrollo de un lenguaje de programación como el lenguaje C, y utilice la herramienta de modelado obtenida y dibuje los

diagramas de componentes correspondientes.

2. Visualice alguna aplicación que involucre a diferentes tipos de usuarios y de preferencia distribuida a lo largo de varios equipos computacionales, en su defecto puede considerar como sería una aplicación de su escuela como la que lleva el control de calificaciones o la administración del proceso de titulación. Utilizando la herramienta de modelado obtenida dibuje el diagrama de despliegue correspondiente.
3. Investigue que es un patrón creador, un patrón de alta cohesión, un patrón de bajo acoplamiento.
4. Investigue en que consiste el concepto de serialización para los beans del leguaje Java
5. Investigue en que consiste el marshaling y el unmarshaling y como son realizados en Borland DELPHI.
6. Realice un programa cliente y servidor, en donde el cliente envía dos números y solicita la suma un servidor, para ello haga uso de lenguaje java y CORBA.
7. Investigue en que consiste la Arquitectura Orientada a Servicios.
8. Considere el funcionamiento de un elevador. Realice el modelado de dicho funcionamiento utilizando una herramienta de modelado UML, todos los diagramas de caso de uso, de secuencia, y especialmente de estado.
9. Investigue en que consiste Kerberos y descríbalo con un ejemplo.
10. Investigue y explique haciendo uso de un diagrama en que consiste el Acceso múltiple por división de código de banda ancha.
11. Busque y lea la especificación de minimun CORBA.
12. Investigue en que consisten los sistemas colaborativos e investigue que sistemas colaborativos en idioma español existen.

Lecturas Recomendadas

Libros

Buschmann, F., Meunier, R. “Pattern-Oriented Software Architecture, A system of Patterns, Volume I”, 1996 ,Willey.

NEMIROVSKI, G., “Setting Requirements for Learning Software”,1998, Ed-Media & Ed-Telecom’98, Freiburg, Alemana,

Artículos.

Medvidovic, N., “Modeling Software Architectures in the Unified Modeling Language”, ACM, Vol 11 January 2002.

.Páginas Web.

http://www.ide.hk-r.se/[~msv/ | ~bosch/] , “Evolution in Software Product Lines”, Department of Software Engineering andComputer Science, University of Karlskrona/Ronneby,

4. Pruebas de Software

El proceso de desarrollo de una aplicación software aún siendo ésta una aplicación monousuario, puede implicar varias líneas de código, al cual se debe haber llegado después de realizado un minucioso estudio de los requisitos funcionales y no funcionales, un exhaustivo análisis y diseño de los elementos de software que comprenden la aplicación.

Sin embargo a pesar del empeño puesto en este desarrollo, la premisa de que "todo sistema es factible de falla", siempre resulta cierto. Lo cual puede deberse a los diversos factores que siempre están presentes desde el inicio de cualquier aplicación de software como pueden ser: la premura del cliente por una entrega rápida, la falta de tiempo del cliente o usuarios para exponer de forma clara y precisa los requisitos funcionales y no funcionales, la falta inclusive de conocimiento de parte del cliente y de los usuarios del proceso y flujo de información que la aplicación debería proveer, algunas veces provocado por el desconocimiento parcial del proceso administrativo en que la aplicación se desempeñará. También atribuible no sólo a los usuarios y a los clientes, la falta de experiencia y/o conocimiento del o los desarrolladores en el desarrollo de aplicaciones similares o en el lenguaje a utilizar, y en las plataformas de software y hardware en las que la aplicación deberá ejecutarse, así como en las técnicas y metodologías más adecuadas a aplicar dependiendo del tipo de aplicación y entorno.

Todas estas posibles causas se ven enormemente incrementadas mientras más compleja resulta la aplicación. Es por todo esto que las Pruebas de Software se hacen indispensables antes de liberarlo a producción. Ya que un desarrollo defectuoso puede no sólo no agilizar el proceso de información, sino todo lo contrario entorpecerlo. Cuantos sistemas no han sido hechos y después de tener sólo un pequeño lapso de tiempo en producción, para después ser abandonados por los usuarios, incurriendo en pérdidas no sólo de tiempo sino de dinero a las empresas. Debido a que este software funcionaba muy bien en manos de su desarrollador, con algunos pocos registros de información, pero una vez puestos en una producción real se encontraron fallas por la falta de previsión en la realización y ejecución de un adecuado plan de pruebas.

En el paradigma orientado a objetos una aplicación de software está compuesta por un conjunto de elementos o componentes de software es por ello que bajo este paradigma las pruebas de software estarán encaminadas a la verificación de puntos tales como:

- Que el funcionamiento de los componentes del sistema sea libre de errores.
- Las interfaces entre las componentes deben proporcionar una adecuada comunicación entre ellas para un correcto ensamblaje entre las componentes, así como, una comunicación que siga las normas de buen funcionamiento con sistemas o subsistemas externos.

- En el caso de que en un futuro se requiriese la actualización o cambio de algún componente del sistema, dicho cambio no debe significar la introducción de un comportamiento no deseado o fallas en el sistema.
- Que el sistema en su conjunto cumpla con los requerimientos estipulados por el usuario desde el punto de vista de funcionalidad, rendimiento, confiabilidad y seguridad.

Por lo tanto para garantizar la calidad de un software es necesario contar con un proceso de pruebas dentro del ciclo de desarrollo de la aplicación, que por lo menos cubra las siguientes etapas:

- Planeación: Se debe crear un documento en el que se describan los requisitos, los casos de prueba, los tiempos y personas involucradas.
- Diseño: Los casos de prueba o pruebas deben ser diseñadas de tal forma que los resultado de los mismas garantice que los requisitos son fielmente cumplidos.
- Ejecución: Las pruebas o casos de prueba deben ser ejecutados en las fechas y duración estipuladas para garantizar que su costo sea mínimo y deben ser cuidadosamente documentados.
- Resultados: Los resultados de las pruebas o de los casos de uso ejecutados, deben ser utilizados para la creación de normas y documentos que indiquen claramente las acciones correctivas y preventivas que deban ser llevadas a cabo en caso de ser necesarias.

Dado que el diseño del programa es dar un panorama general de las pruebas de software los siguientes puntos serán expuestos con dicha intención.

4.1 Definiciones

4.1.1 Prueba, caso de prueba, defecto, falla, error, verificación, validación.

Prueba. La definición de prueba es que se trata de un medio o alguna forma con la cual se pretende averiguar se demostrará la verdad o falsedad de algo que se da por establecido. En el ambiente del software una prueba consiste en examinar la ejecución de un subsistema o un módulo o una interfaz o incluso una porción de un código para tener un registro de la observación de los resultados que se obtienen tras la ejecución y posteriormente comparar si concuerdan con lo especificado en el momento del diseño.

Caso de prueba. Un caso de prueba es un conjunto de normas, reglas y pasos que deben seguirse conforme a un orden establecido, para determinar si un requisito en particular sea éste primario o secundario, es cumplido a completa satisfacción del cliente de acuerdo a lo que se estableció en el análisis y diseño del mismo.

Defecto. Un defecto es definido como una imperfección en algo. En el campo de la ingeniería de software, es más bien la carencia de la característica intrínseca ya sea de un paso de una actividad o tarea o toda una tarea o actividad, cuando se da la ejecución de un código, dando como resultado un procesamiento incorrecto.

Fallo. Se define como la frustración o el no efecto tener el efecto deseado en la realización de algo. En ingeniería de software establece la incapacidad que demuestra alguna componente de un sistema para llevar a cabo las funciones requeridas dentro que den como resultado que el o los requisitos especificados cumplan con los parámetros establecidos. .

Error. Puede ser definido de varias maneras como una acción o conceptos equivocados, algo hecho fuera de las especificaciones establecidas, etc. Sin embargo cuando se le quiere dar a la definición de error un sentido de precisión se define como: La diferencia existente entre el valor real y el valor calculado teóricamente. Desde el punto de vista de ingeniería de software un error es el que se encuentra cuando se prueba el código y el resultado de dicha prueba, produce un resultado diferente al esperado.

Verificación. En ingeniería de software, se considera la acción de determinar si la elaboración de la aplicación de software es la correcta o no.

Validación. Constituye en el comprobar si todos y cada uno de los requisitos de software han sido satisfechos.

La verificación, al igual que la validación, están inmersas en el proceso de gestión de la calidad de software el está constituido de un extenso grupo de actividades, como lo son las revisiones técnicas formales, las auditorías de calidad y de cambios, la revisión de la documentación, el análisis de los algoritmos , de la factibilidad de la aplicación el análisis y prueba de las bases de datos, llevando a cabo el monitoreo del desempeño a través de pruebas a lo largo del desarrollo, así como pruebas de la usabilidad, para las cuales se llevan a cabo simulaciones, y por último las pruebas en la instalación.

4.1.2 Relación entre defecto-falla-error.

En el consenso general de los ingenieros de software error, falla y defecto son sinónimos. Sin embargo algunos autores los diferencian de tal manera que un error que no haya sido detectado en la etapa de pruebas del sistema , producirá que una porción de código sea defectuosa, dando a su vez como resultado que se produzca una falla durante la ejecución de alguna interfaz o proceso en la aplicación. Una falla puede contener uno o más defectos.

4.1.3 Pruebas estructurales, funcionales y aleatorias.

Para el diseño de casos de prueba existen 3 enfoques:

- estructural o de caja blanca
- funcional o de caja negra
- enfoque aleatorio

En el primer enfoque, como su nombre lo indica se trata de probar las estructuras de control del código de una aplicación pero desde un punto de vista de lo que hace y produce dicho código, no de lo que se supone debería producir para cumplir con los requisitos, con el objetivo de centrarse en los caminos de ejecución para encontrar aquellos que sean incorrectos.

En el segundo enfoque, también como su nombre lo indica se encamina hacia probar las funciones, tanto las entradas y salidas que producen, para detectar por ejemplo como se comporta la aplicación con ciertos tipo y rangos de datos de entrada, al mismo con su volumen. En este enfoque importa constatar si los requisitos son cumplidos por la aplicación o no.

El último enfoque hace uso de modelos estadísticos que básicamente representan probables entradas al sistema.

4.1.4 Documentación del diseño de las pruebas.

De acuerdo con el estándar IEEE 829, los documentos relacionados con el diseño de pruebas son los siguientes:

- Plan de Pruebas. Este documento sirve de base para conocer cada uno de los requisitos que serán probados y de manera general el enfoque que se da a cada prueba, así como las actividades, técnicas, herramientas, los tiempos y personas necesarias, y los riesgos asumidos. Este plan da pauta a las especificaciones de diseño.
- Especificaciones de diseño de pruebas. Para cada requisito estipulado en el plan de pruebas se hace un refinamiento de cada uno de los puntos, es decir que se detallan los elementos de software que serán incluidos en la prueba de un determinado requisito, que tiempo, personas, serán utilizadas en estas pruebas. Especificando a su vez los casos de prueba y también los procedimientos de prueba, que involucraran las pruebas a un determinado requisito.
- Especificaciones de casos de prueba. Cada requisito puede contar con varios casos de prueba. El documento de un caso de prueba especifica cada entrada del caso y sus relaciones con las demás entradas. Lo mismo para las salidas considerando sus tiempos de respuesta y las dependencias que puedan existir entre casos.
- Especificaciones de procedimiento de pruebas. El objetivo de este documento es el detallar los pasos necesarios para que un conjunto de casos puedan ser ejecutados, entre los que se cuentan aquellos que vigilan la secuencia de acciones para preparar la ejecución, la secuencia de acciones para empezar la ejecución, la secuencia de acciones necesarias durante la ejecución, la secuencia de acciones para la suspensión de la prueba en caso de ser necesario y de ahí la secuencia de acciones detener ordenadamente la ejecución, así como la secuencia de acciones necesarias para restaurar el entorno y dejarlo en la situación existente antes de las pruebas. Y por último la secuencia de acciones para tratar los acontecimientos anómalos, así como el considerar que instrumentos de medición se utilizarán para la realización de medidas, como por ejemplo, el tiempo de respuesta.
- Informes de ejecución de pruebas. En esto documentos se plasmarán los resultados de las pruebas y en caso de ser necesario las medidas que se tomarán para una reingeniería. Son de tres tipos de documentos:
    - Histórico de pruebas. Se plasman los hechos relevantes ocurridos durante la ejecución de las pruebas.
    - Informe de Incidentes. Se describen los incidentes que ocasionaron la interrupción de una prueba.
    - Informe Resumen de las pruebas. Se detallan los resultados de cada prueba y como cada uno de los elementos de software contribuyo a dicho resultado, las posibles causas y las posibles soluciones.

4.2 Proceso de pruebas.

4.2.1 Generar un plan de pruebas.

Un plan de pruebas debe contar con los siguientes puntos:

- Un identificador del plan. El cual debe ser un mnemónico para un fácil entendimiento de que se trata el plan y cual es su alcance.
- Versión. Todo plan al igual que cualquier proceso, está sujeto a revisiones y cambios por lo que es indispensable tener un registro de la versión actual.
- Fechas. La fecha en el que el plan fue creado así como la última fecha en el que fue actualizado.
- Alcance. En el se manifiesta que configuración de la aplicación (cual versión) va a ser probada.
- Elementos del software que intervendrán en la prueba.
- Estrategia. Se analiza cómo plantear las pruebas en el Ciclo de desarrollo de la aplicación. Quedan descritas las técnicas y herramientas a ser usadas en los casos de prueba, así como los casos de prueba a diseñar. Por lo regular una estrategia consta en la ejecución de pruebas a distintos niveles, así se inicia en el probar los módulos o componentes mayores, luego se comprueba la integración de los mismos y por último se realizan pruebas con el usuario para su aceptación.
- Expectativas y Criterios a seguir en el curso del plan. Se debe establecer un conjunto de criterios que permitan categorizar la configuración de las pruebas, como cuando y bajo que esquemas, las pruebas deben ser suspendidas, cuando reanudadas y cuando se dan por terminadas.
- Productos derivados, es decir, los documentos que se tendrán al realizarse las pruebas.
- Supuestos, ámbito, prioridades y complejidad de las pruebas.
- Requisitos funcionales. Se refieren a las tareas necesarias para preparar y ejecutar las pruebas.
- Recursos necesarios. Elementos tanto de hardware como de software y espacios u otros elementos físicos, incluidos los de seguridad, necesarios para la ejecución de las pruebas. Así como las personas responsables y encargadas de ejecutar y observar las pruebas, y cualquier otra dependencia.
- Programación de tiempos. También se deben establecer los hitos existentes durante el proceso de prueba y el grafo de dependencia en el tiempo de las tareas a realizar.

- Riesgos y plan de contingencia.

4.2.2 Diseñar pruebas específicas.

Una vez que se cuenta con el plan maestro de pruebas si el sistema es complejo se debe desarrollar un plan detallado para cada prueba, para lo cual se hace un refinamiento de lo establecido en el plan de pruebas. Estos planes, al igual que el maestro, se basan en la especificación funcional y en otros documentos de diseño de alto nivel. Para cada uno de ellos, se asigna a una prioridad, la cual se basa en la probabilidad de su impacto en el negocio. Estos planes suelen estar divididos conforme a la organización del grupo de pruebas, al orden de disponibilidad de varios componentes del sistema o el área de funcionalidad.

4.2.3 Tomar configuración del software a probar.

No se requiere en que cada vez que se modifica la aplicación de software se debería forzosamente llevar a cabo una prueba a todos y cada uno de los módulos, sobre todo cuando se está en los inicios del desarrollo, ya que esto sólo llevaría a retrasos e incremento de costos innecesarios. Más bien esto dependerá del ciclo de desarrollo del sistema, ya que por ejemplo anteriormente si un sistema tenía un ciclo de desarrollo lineal, las pruebas eran llevadas a cabo una vez que se tenían ya programados los módulos del sistema, en cambio por ejemplo en un ciclo incremental, dado que existe entregas parciales, previo a esto se deben llevar acabo pruebas.

Por otra parte para un desarrollo llevado a cabo con el paradigma orientado a objetos y con un ciclo de vida espiral, las pruebas son ejecutadas aunque no de una manera formal en cada refinamiento inicial de los requisitos ya que continuamente se busca la aceptación del usuario a través del proceso de desarrollo.

Sin embargo existen algunos factores que deben ser tomados en cuenta, sin importar el ciclo de desarrollo de sistema y estos son: el tiempo y el costo de llevar a cabo una prueba versus los riesgos de no hacerlas en una determinada configuración. Ya que los riesgos pueden llegar a ser altos si se sigue adelanto con una nueva configuración del software (cambios hechos en el código, es decir una nueva versión) sin haber antes realizado las pruebas pertinentes, ya que se podrían arrastrar errores y defectos, que al ser detectados posteriormente pueden incurrir en altos costos de reingeniería.

4.2.4 Configurar las pruebas.

El diseño de pruebas especificas dependerá del paradigma en el que se diseñen, así como de las técnicas que sean adecuadas, por ejemplo si es un desarrollo con arquetipo orientado a objetos y en base a casos de uso, los escenarios de éstos, así como el requisito funcional plateado en los mismos, se traducen en uno o varios casos de pruebas detallados con el fin de crear documentos que muestran los pasos en detalle a seguir, los requisitos de datos para realizar la prueba, los resultados esperados y los métodos para registrar los resultados.

Hay que tener en cuenta sin embargo que las pruebas al igual que el proceso de desarrollo de software son un proceso dinámico que puede variar de acuerdo a las circunstancias. Es decir el plan de pruebas puede estar sujeto a cambios debido principalmente al resultado que se vaya obteniendo de las pruebas iniciales, ya que si alguna de ellas resulto que paso por alto algún criterio durante su planeación, y el cual se llego advertir durante la ejecución de la misma, la prueba puede requerir un cambio por lo que es necesario al configurar estas pruebas, tener en cuenta los riesgos y las alternativas a seguir en caso de que se de alguna variación, puesto que las pruebas pueden tener dependencias con más de una prueba.

4.2.5 Evaluar resultados.

De la evaluación de los resultados de cada prueba hecha al sistema dependerá si la planificación de su ciclo de vida debe o no sufrir modificaciones, ya que los tiempos programados para la terminación de cada etapa de desarrollo de las componentes pueden verse afectado, siendo necesaria una reprogramación de los mismos.

Aunque estas contingencias deben estar previstas dentro del programa de tiempos de un sistema, cuando se establecen las holguras a cada uno de dichos tiempos. Únicamente el problema se volvería crítico si dichas holguras no son suficientes y la depuración y análisis de errores implica más tiempo del probable considerado.

4.2.5.1 Depuración.

Una vez obtenidos los resultados de las pruebas es conveniente hacer una depuración de los mismos, clasificándolos en cuanto a la afectación o no afectación en el proceso de desarrollo, y a su vez hacer una subclasificación de acuerdo a su importancia en cuanta a la afectación.

4.2.5.2 Análisis de errores.

Aquellas pruebas cuyos resultados no fueron satisfactorios, deberán analizarse para encontrar las causas o errores que produjeron fallas durante la realización de las pruebas, para programar su corrección.

4.3 Técnicas de diseño de casos de pruebas.

El empleo de adecuadas técnicas de diseño de pruebas dará como resultado un software de calidad. Las técnicas clásicas empleadas se determinan por el enfoque empleado en el diseño de los casos de prueba, así tenemos técnicas basadas en un enfoque estructural como son las técnicas de caja blanca, o las técnicas basadas en un enfoque funcional como las técnicas de caja negra.

Pruebas de Caja Blanca o Estructurales.

El porque es importante dar seguimiento a los caminos de ejecución de un código en lugar de enfocarse exclusivamente a comprobar si dicho código cumple o no con los requisitos de usuario, se debe principalmente a que tanto los errores lógicos como las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino, dicho de otro modo que aquellos caminos que consideramos tienen muy poco probabilidad de ejecutarse, al hacerle pueden producir errores de considerable magnitud.

Por lo tanto las técnicas empleadas en la prueba de caja blanca son:

- Prueba del camino básico.
- Prueba de Bucles.

Prueba del camino básico.

Esta prueba tiene como meta el obtener una medida de la complejidad lógica de un diseño procedimental lo que se conoce como complejidad de Mc Cabe, para utilizarla como una guía en la definición de un grupo de caminos de ejecución, que garanticen que durante la prueba las sentencias en dichos caminos se ejecutan por lo menos una vez.
En un código se pueden encontrar distintas sentencias como por ejemplo: una o más sentencias de asignación, una o más sentencias de condición (“if”, “ if-else”, “switch” o “case”) y/o una o más

sentencias de ciclo (“while”, “for”, “for each”, “do-while” o “repeat-until”). Las cuales pueden generar varios caminos de ejecución, y ya que la prueba implica que toda sentencia sea ejecutada por lo menos una vez, la gran cantidad de caminos que se pueden tener podría sugerir que el probarlos a todos y cada uno de ellos llevaría un tiempo excesivo, sin embargo esto puede reducirse si se encuentra y prueban sólo los caminos independientes.

Por lo que este código es representado como un grafo en el que cada nodo representa una o más sentencias, un único nodo puede corresponder a una secuencia de pasos del código que termina con una decisión y las aristas representan el flujo de control.

La figura 87 presenta un grafo para un conjunto de instrucciones de asignación, en la figura 88 se muestran los grafos correspondientes a las sentencias de condición, y por último en la figura 89 se esquematizan los grafos a instrucciones de ciclo.

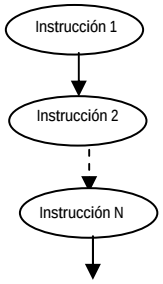
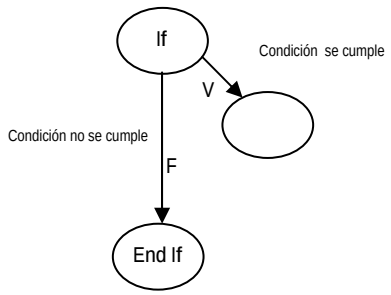
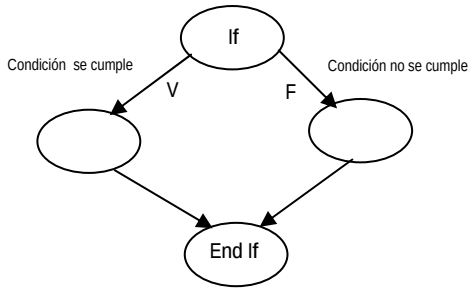


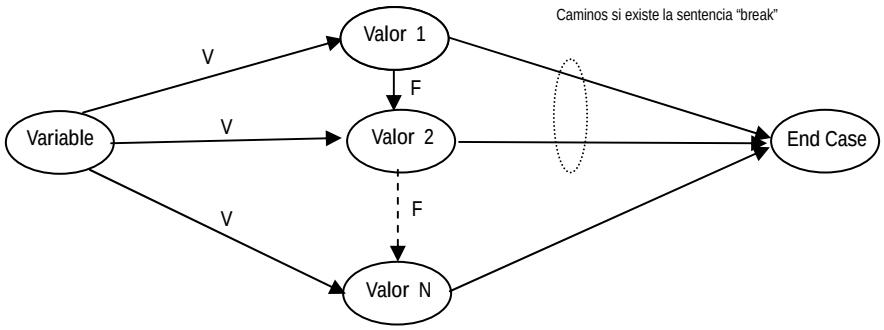
Figura 87. Grafo de una secuencia de instrucciones de asignación o sin caminos de bifurcación.



Instrucción “if”



Instrucción “if- else”



Instrucción “Switch” o “case””

Figura 88. Grafos de instrucciones de condición.

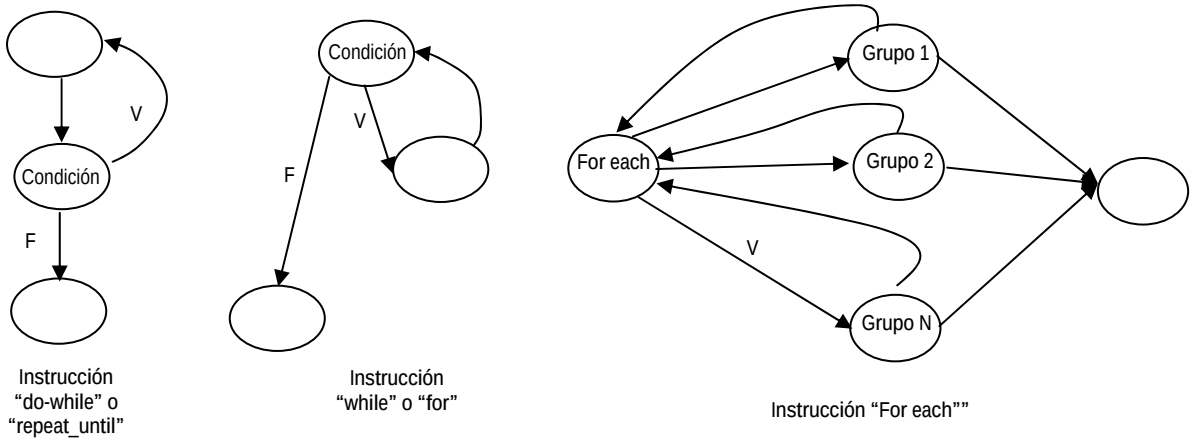


Figura 89. Grafos de instrucciones de ciclo.

Hay que considerar que si en el código se utilizan condiciones compuestas, en la generación del grafo se tienen que descomponer estas condiciones compuestas en condiciones sencillas. Un ejemplo de esta descomposición puede observar en la figura 90.

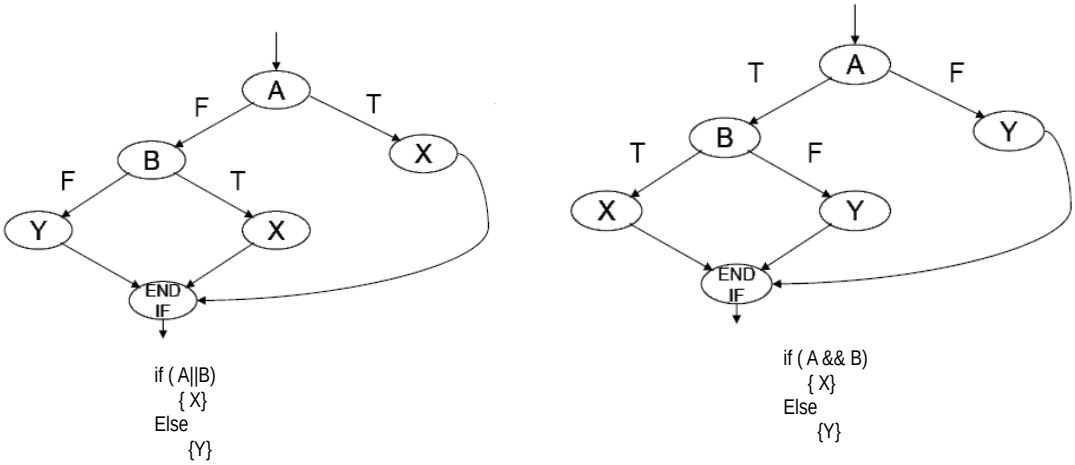


Figura 90. Grafos, resultado de condiciones compuestas.

Ahora bien para tener una idea de la complejidad ciclomática, que no es otra cosa más que el número de caminos independientes a probar, se hacen las siguientes consideraciones:

Sea V(G) la complejidad ciclomática,

Se tiene que esta complejidad puede ser calculada de las siguientes maneras:

V(G) = # de regiones, áreas cerradas limitadas por aristas.

V(G)= A – N + 2 , donde A es igual al número de aristas y N es igual al número de nodos

V(G)= Np +1, donde Np es igual al número de nodos predicado que son aquellos nodos con más de una arista de salida.

Una pregunta interesante sería si ¿este tipo de prueba puede ser aplicado a un paradigma orientado a objetos?, la respuesta es si, ya que la programación orientada a objetos involucra el mismo tipo de sentencias que el paradigma estructurado, pues en los métodos de los objetos encontraremos sentencias de condición, de asignación y de ciclo también. Por otra parte si se recuerda los casos de uso son una descripción de una secuencia de pasos ejecutados entre un actor y el sistema, y los casos de uso pueden tener distintos escenarios, como la ruta normal de eventos o la ruta por excepciones o las rutas por decisiones. De un caso de uso se pueden obtener uno o más diagramas de actividades, y éstos a su vez pueden ser representados como grafos.

Se recomienda que cuando el número de caminos es mayor de 10 y no existen sentencias tipo "switch", rediseñar el código, ya que la probabilidad defectos crece conforme el número de caminos independientes.

Prueba de bucles.

Dado que en código de un sistema es casi imposible no encontrar ciclos, las pruebas de bucles son importantes, ya que intentan demostrar la validez de los mismos. De hecho se tienen los siguientes tipos de bucles, como se ilustran en la figura 91:

- Simples
- Anidados
- Concatenados
- No estructurados

Simples

A este tipo de bucles se les puede aplicar cualquiera de las siguientes pruebas:

- No recorrer el bucle ni una sola vez
- Recorrer el bucle solo una vez
- Recorrer el bucle dos veces
- Dado N el número total de veces que el bucle puede ser recorrido, recorrerlo m veces donde m<N
- Recorrer N-1, N y N+1 el bucle.

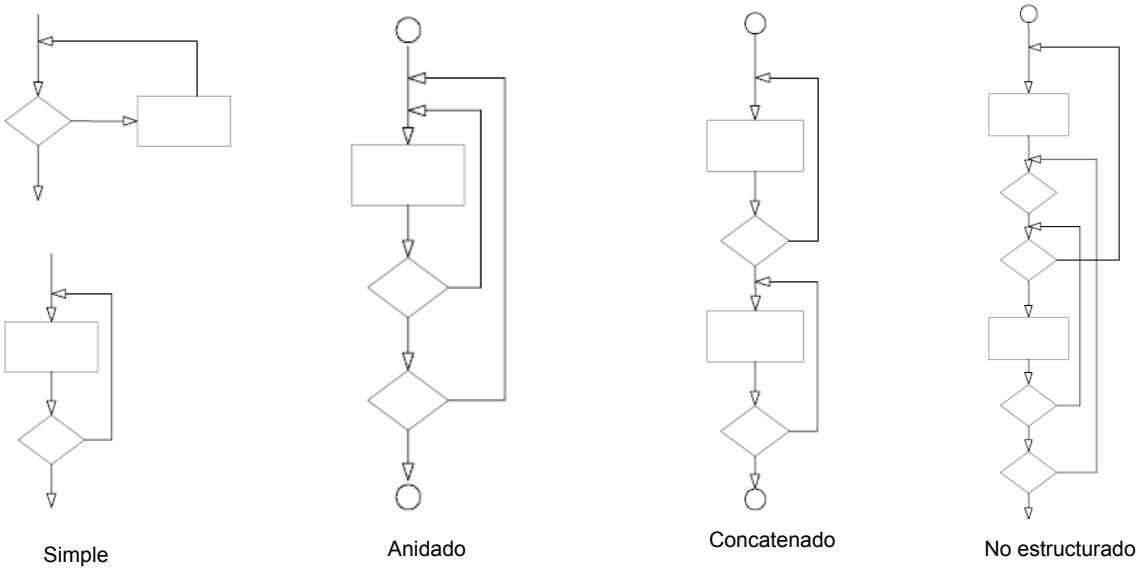


Figura 91. Tipos de ciclos

Anidado

Las pruebas para los bucles anidados no pueden ser iguales a los del tipo simple ya que crecerían de una manera geométrica, es por eso que se recomienda seguir los siguientes pasos:

- Iniciar con el bucle interno, poniendo a los demás bucles sus los valores mínimos.
- Ejecutar al bucle interno las pruebas de bucles simples, y mantener los valores mínimos de iteración en los bucles más externos.
- Continuar en el siguiente bucle externo, y mantener los bucles más externos a sus valores mínimos.
- Seguir hasta que se hayan probado todos los bucles.

Concatenado

Separar los bucles simples y aplicar las pruebas para tipo de bucle simple.

No estructurado

No permitir este tipo de bucles, transformarlo a un bucle estructurado.

Pruebas de Caja Negra o Funcionales.

El propósito principal de estas pruebas, es el enfocarse en las interfaces, sin considerar la estructura interna de las componentes de software ni su comportamiento, sino que se enfoca sobre su funcionabilidad, sus entradas y sus salidas; ya que lo que se trata de probar es que se cumpla con los requisitos funcionales.

Las Técnicas empleadas en las pruebas de caja negra son:

- Partición Equivalente
- Análisis de Valores Límite
- Grafos de Causa-Efecto
- Pruebas de Comparación

Partición Equivalente

Esta prueba consiste en agrupar los datos de entrada, en tipos de datos con un comportamiento equivalentes entre si, en cuanto a que el sistema los trata de manera similar. El objetivo es que si la aplicación falla al tratar un dato de un grupo, existe una alta probabilidad que fallará con los demás. Estos grupos o particiones se conocen con el nombre de ***clases de equivalencia***, y representan a un conjunto de datos que determinan estados válidos o inválidos para las condiciones de entrada del sistema; de tal manera que una clase válida, generara un valor esperado, mientras que una clase inválida generara un valor no esperado.

Dado que estas clases son obtenidas a partir de las condiciones de entrada, las cuales vienen representadas por sentencias en la especificación como:

- Un valor específico. En este caso se obtendría una clase de equivalencia válida que contempla al valor en cuestión y dos clases de equivalencia inválidas, una por debajo del valor y otra por encima.
- Un rango de valores. En este caso el número de clases de equivalencia válidas obtenido sería una clase que contemple los valores del rango, y dos clases de equivalencia inválidas: una por encima del rango y otra por debajo.
- Una condición lógica. En este caso se tendrá una clase de equivalencia válida que cumpla con la condición y una clase equivalencia inválida, la que no cumple con la condición
- Un rango de valores relacionados. Ya que los elementos de este grupo son tratados de forma diferente, se tendrá una clase de equivalencia válida por cada elemento y una para un elemento fuera de este rango.

Se deberán generar tantos casos de prueba como clases de equivalencia existan, tanto válidas como inválidas.

Análisis de Valores Límite

Dado que la experiencia ha demostrado que existe una alta probabilidad de encontrar fallas al aplicar valores límites en los campos de entrada y en las de salida. El análisis de valores límite es un complemento de la prueba de partición equivalente, ya que los casos de prueba se diseñan para ser ejecutados en los extremos de las clases de equivalencia.

De tal manera que si se tiene una condición de entrada con varios valores, los casos de prueba se diseñan para el valor máximo y el valor mínimo de ese rango de valores, y también casos de prueba para arriba del valor máximo y para abajo del valor mínimo. Los primeros serán casos de prueba para clases de equivalencia válidas y los dos últimos para clases de equivalencia inválidas. Lo mismo se hace para las condiciones de salida.

Por otra parte si se tienen estructuras de datos internas en las cuales se establecen valores límites se deben diseñar casos de prueba para probar dichos límites, los cuales serán clases de equivalencia válidas para los valores límite y clases de equivalencia inválidas para los valores por encima del máximo y por debajo del mínimo.

También se deben diseñar casos de prueba cuando existen valores específicos tanto para la condición de entrada, como para la condición de salida, siendo un caso de prueba para cada clase de equivalencia válida para cada valor específico y un caso de prueba para una clase de equivalencia inválida para el valor superior del valor específico y otro para el valor inferior del valor específico.

Grafos de Causa-Efecto

Los grafos de Causa-efecto son utilizados especialmente cuando se tienen especificaciones de salidas que tienen un alto grado de dependencia con respecto a combinaciones de entradas de tipo “OR” y de tipo “AND”, con la idea de detectar ambigüedades o no completitud en las especificaciones.

El concepto en el que se basan estos grafos es el generar salidas admisibles bajo las siguientes reglas empíricas:

Si existe en una entrada binaria un “OR”, entonces se consideraran todas las opciones con sólo una señal en “verdadero”.

Si por otra parte la entrada binaria es un AND”, entonces se consideraran todas las opciones con sólo una señal en falso.

4.4 Enfoque práctico recomendado para el diseño de casos.

Para el diseño de casos de prueba estructurales se recomienda llevar a la práctica los siguientes puntos:

Al aplicar pruebas mediante el camino básico:

- Obtener el grafo, a partir del código del módulo si se usa un enfoque estructurado y si se usa un enfoque orientado a objetos de los diagramas de actividad obtenidos a su vez de un caso de uso.
- Calcular la complejidad ciclomática del grafo.
- Definido el conjunto básico de caminos independientes, diseñar los casos de prueba que permitan la ejecución de cada uno de los caminos independientes.
- Ejecutar cada caso de prueba y comparar los resultados con los esperados.
- Documentar los resultados y las recomendaciones pertinentes.

Para el diseño de casos de prueba funcionales se recomienda llevar a la práctica los siguientes puntos:

- Cuando una especificación de diseño cuenta diversas combinaciones de condiciones de entrada, se debe comenzar diseñando pruebas con grafos de causa-efecto, con la finalidad de que faciliten la comprensión de dichas combinaciones.
- Cada causa que genere nuevos casos, debe considerarse como una clase de equivalencia y por lo tanto se debe hacer un cuidadoso análisis de valores-límites para agregar casos de prueba.
- Se deben reconocer las clases válidas y no válidas de equivalencia tanto para la entrada y como para la salida, y se deberá entonces agregar los casos no incluidos anteriormente.
- Cuando se tengan valores especiales se deberá hacer uso de la técnica de conjetura de errores para añadir nuevos casos.
- Llevar a cabo los casos generados de caja negra y aplicar las herramientas correspondientes para efectuar un análisis de cobertura.
- Examinar la lógica del código para en su caso añadir los casos de prueba de caja blanca que cumplan con el criterio de cobertura elegido.

4.5 Estrategias de aplicación de las pruebas.

4.5.1 De unidad.

Esta estrategia de prueba se utiliza para probar software convencional, específicamente la prueba se realiza sobre las componentes o módulos de una aplicación y asegurar que dicha componente funciona adecuadamente en cuanto al procesamiento de la lógica y estructuras de datos internas a la componente. Esta prueba puede llevarse a cabo sobre varias componentes de forma paralela.
La implementación de esta estrategia en el paradigma orientado a objetos, podría pensarse que la prueba de unidad debe hacerse sobre una sola clase, sin embargo, esto implica varios posibles riesgos, en primer lugar la clase puede ser una superclase, o incluso peor ser una clase abstracta o tener métodos abstractos, lo que significa que las clases que deriven de esta superclase implementarán dichos métodos, cada una de la manera más conveniente para ella. Por otra parte las instancias de clase en un aplicación orientada a objetos no trabajan solas, ya que la idea principal tras las orientación a objetos, es la cooperación entre ellos para en conjunto llevar a cabo la realización de una tarea en específico; es por esta razón por la cual debe realizarse un prueba de unidad no sólo sobre una clase en específico sino sobre todas y cada una que directamente estén involucradas sobre una tarea en particular, así por ejemplo, supóngase una aplicación que hace uso de un archivo de texto para su edición y cuya programación es en tres capas y considérese la tarea de abrir un archivo ya anteriormente creado, esta sola tarea involucra a varias clases, las que se encuentran en la capa de vista, las de control y las de acceso a datos, aunque se considerase una sola clase de interfaz como podría ser por ejemplo una ventana de diálogo en la cual se introdujese el nombre y ruta del archivo n cuestión, ésta estaría ligada a una clase de control que se encargaría de las verificaciones correspondientes (sobre el nombre y ruta). Si se ejecuta la prueba de unidad exclusivamente para la clase de interfaz, a pesar de ésta resultase exitosa, no garantiza que la tarea se realizase exitosamente.

La prueba de unidad en un contexto orientado a objetos por la tanto debería procurar cumplir con las mismas consideraciones que en un contexto no orientado a objetos y que son la verificación de que si existe una interfaz de usuario la información introducida o mostrada en ella, emane adecuadamente, es decir que cumpla con los requisitos funcionales, como por ejemplo: velocidad de respuesta, exactitud, confiabilidad, seguridad, etc., También que las estructuras de datos mantengan su integridad a lo largo de la ejecución de la prueba. Verificar que todos los caminos de ejecución se han ejecutado por lo menos una vez., así como los hilos de ejecución de excepciones y por último lo más importante que se han probado todas las condiciones límite. Una condición límite es aquella en la

cual las variables cuentan con los valores límites de un rango y las sentencias que se ejecutan sobre estas variables con estos valores deben garantizar que no producen errores, un error muy común es cuando en un arreglo de datos numéricos, se hace referencia a un elemento cuyo índice sobrepasa el rango especificado.

Dado que la prueba de unidad involucra el procedimiento interno de una unidad, es importante que su diseño se realice a la par del diseño de la unidad o por lo menos antes de la codificación. Dado que la funcionalidad de una tarea puede involucrar varias unidades, es conveniente por lo tanto crear un software que permita inicializar la unidad y en caso de ser necesario admitir datos de entrada para esta unidad, así como el imprimir los resultados que la unidad arroje.

Para la realización de pruebas de unidad se utilizan las técnicas de prueba de caja blanca y de caja negra, tratadas en un inciso anterior.

4.5.2 De integración.

Como se había mencionado anteriormente en un contexto orientado a objetos la prueba de una sola clase no garantiza el funcionamiento de una tarea ya que los objetos contribuyen entre sí para la realización de una tarea, no sólo basta con la prueba de unidad sobre todas las clases inmiscuidas, sino que también es necesario garantizar que la cooperación entre ellas sea efectiva, es decir que ejecuten la tarea con los resultados esperados. Para esto es entonces necesario ejecutar pruebas de integración.

Una prueba de integración entonces, será aquella que una vez aprobada garantice que los módulos o clase o unidades involucradas en la realización de una tarea la ejecuten de una manera que los requisitos funcionales se den por satisfechos.

La prueba de integración debe ser extensiva a todas las tareas pertenecientes a la aplicación, hasta concluir que toda ella es aceptable.

El enfoque más adecuado de una prueba de integración por lo tanto es el incremental, el cual a su vez se divide en dos aspectos:

- Ascendente. En este enfoque las unidades son integradas de abajo hacia arriba, es decir, se comienza con los módulos de nivel básico o atómico, como podría ser una clase interfaz, y sus correspondiente clase de control y de acceso a datos, por ejemplo, si la aplicación se desarrollo en un modelo orientado a objetos con una arquitectura en tres capas. Y así se sigue hacia arriba con todas las demás clases involucradas en un caso de uso y posteriormente con todos los casos de uso de un paquete, para llegar finalmente a la integración de todos los paquetes.

  En este enfoque se utilizan las ***pruebas de regresión*** y las ***pruebas de humo***.

  En la *prueba de regresión* se considera que cada vez que una unidad es integrada el software anteriormente integrado cambia en su totalidad, ya que se tienen nuevas entradas y salidas y caminos de flujo y procesamiento de datos, que pueden orillar a errores en funciones que antes de la nueva integración trabajaban correctamente. Es por ello que se deben ejecutar nuevamente las pruebas que ya se habían aplicado con anterioridad para asegurarse que la integración de la nueva unidad no produce errores. Claro está que esto podría llegar no sólo a ser tedioso sino consumir mucho tiempo el estar repitiendo todas y cada una de las pruebas cada vez que una nueva unidad es integrada, por lo cual no se aplican en la realidad todas las pruebas, sólo aquellas que se concentran en los procedimientos, datos y funciones que están relacionadas con la nueva unidad.

  Por su parte la *prueba de humo* consiste en probar de manera continua todo el software, mientras se construye, es decir que se integran de manera temporal y mientras se tenga ya desarrollada ciertas funcionalidades que la aplicación debe brindar, las unidades, para probar los posibles errores que hasta el momento se pueden tener. Estas pruebas son usadas cuando el tiempo del ciclo de vida del proyecto es crítico.

- Descendente. Otro de los enfoques de las pruebas de integración es el descendente. Este enfoque a su vez puede aplicarse de dos formas distintas, una de ellas es primero a lo ancho y la otra es primero a lo fondo.

  Si se considera que se tiene en cuenta una vista de desarrollo estructural en una aplicación de software, entonces dicho software visto en un nivel macro estará compuesto de módulos, en lugar de paquetes de componentes como fuese visto si se hubiese seguido un desarrollo orientado a objetos. Si se toma en cuenta que dichos módulos o paquetes pueden ser representados esquemáticamente como si se tratase de un organigrama, se vería en dicho esquema que los módulos pueden tener diferentes jerarquías, en las cuales unos módulos estarán en un mismo nivel mientras que otros dependerán de otros, es decir, estarán en nivele inferiores, así que una integración descendente puede ser considerada para llevarse a cabo en módulos de un mismo nivel de jerarquía, lo que significa que se hará de una manera

primero a lo ancho, o en su defecto la integración descendente puede hacerse con módulos de diferentes niveles de una manera primero a lo profundo.

En el enfoque descendente también se pueden efectuar también las pruebas de regresión y de humo. Sin embargo el enfoque descendente es más difícil de implementar ya que se requerirá el tener el conjunto de módulos a integrar o en su defecto el crear módulos ficticios, los cuales al no contar con toda la programación real, pueden carecer de algunos elementos no fácilmente detectables o previstos como posibles fuentes de errores.

4.5.3 Del sistema.

Las pruebas del sistema son un conjunto de pruebas que garantizan la integridad, funcionabilidad y confiabilidad del sistema en un entorno muy similar al real donde será utilizado. Estas pruebas incluyen tanto aspectos de software, como de hardware como humanos, ya que se considera también tanto a los usuarios como a los operarios del sistema. Podrían ser clasificadas en los siguientes rubros:

Pruebas de usabilidad. En estas pruebas se solicita a los usuarios del sistema que lo utilicen en las diferentes tareas para las cuales fue diseñado, se observa el tiempo que al usuario le toma comprender y ejecutar de manera efectiva las interacciones correspondientes a una tarea en específico, cual es su respuesta emocional en dicha ínter actuación con el sistema, si fue o no favorable; cuantos y cuales errores cometió, si su recuperación le fue sencilla de entender o no. Si pasado algún tiempo de ejecutar un grupo de interacciones y vueltas a ejecutar le es fácil recordar lo que tenía que hacer o no.

Pruebas de tolerancia a fallos. Se procura que la o las personas encargadas de realizar estas pruebas, efectúen el mayor número de errores posibles, para verificar la recuperación del sistema ante dichas faltas. También se introducen un conjunto de fallas a nivel hardware, como falla de energía eléctrica, falla de conexión con los servidores, dispositivos de entrada y salida no listos, etc.,

Pruebas de seguridad. Para garantizar la confiabilidad y la integridad en un sistema, se introducen elementos de ataque al mismo para probar que los mecanismos de protección implementados no sólo sean los adecuados sino que también funcionen correctamente.

Pruebas de resistencia y desempeño. No es difícil encontrar casos en los cuales un sistema a cumplido de manera aparente los requisitos funcionales y ha logrado la aceptación del cliente, para

luego de la manera más vergonzosa resultar un sistema tan lento que el usuario prefiere desecharlo y regresar a su anterior manera de procesar la información. Esto se debe principalmente a que las pruebas fueron realizadas con una carga de información relativamente muy pequeña a la carga real de trabajo. Es por eso muy importante el probar el sistema con cargas grandes, no sólo en los volúmenes de información, sino también el uso exhaustivo de los recursos de cómputo.

4.5.4 De aceptación.

Las pruebas de aceptación, como su nombre lo indica buscan la aceptación del cliente y sus usuarios para que el producto de software que ha sido elaborado pase a su etapa de producción. Por lo cual son realizadas en presencia de usuarios y cliente y cuando la aplicación ya ha pasado por las demás pruebas y se tiene un producto completo e integrado. En este tipo de prueba no sólo se prueban los requisitos funcionales y no funcionales, sino también el manual de instalación y uso del sistema. Por lo que se pide al usuario se base en estos documentos a través de la prueba, con el fin de constatar si están bien redactados, si sus instrucciones son fáciles de seguir, si los pasos que se indican son los debidos y son congruentes entre sí y entre la aplicación, si dichos pasos dan los resultados que se indican o si dichos resultados no se indican o no están bien indicados etc.,

Las pruebas de aceptación son conocidas como prueba alfa y prueba beta.

Prueba alfa. Se realiza en el lugar de desarrollo en un entorno controlado, el software es probado por un usuario en presencia de uno o más desarrolladores. Los resultados de esta prueba se documentan.

Prueba beta. Se realiza en lugar o lugares donde la aplicación será utilizada. Esta prueba no se hace en presencia de los desarrolladores es efectuada únicamente por los usuarios. Son ellos los que documentarán los resultados y utilizarán cargas leves, pero datos reales, es decir, no se cargará toda la información real de un estado en producción, sólo la necesaria para llevar a cabo la prueba. A parte de la documentación de las pruebas de ser necesario el usuario está en plena libertad de transmitir de forma verbal o escrita, de no ser posible la primera, todas las observaciones o sugerencias que crea conveniente al grupo de desarrolladores, los cuales deberán llevar a cabo las modificaciones pertinentes y presentar una nueva versión del producto de software para que sea nuevamente probada por todos y cada uno de los diferentes usuarios.

Preguntas

1. ¿Cuáles son las etapas del proceso de pruebas?
2. ¿cuál es la diferencia entre error, falla y defecto?
3. ¿Qué otro nombre reciben las pruebas funcionales?
4. ¿Cuáles son los puntos de un plan de pruebas?
5. ¿En que consisten las pruebas de bucles?
6. Explique en que consiste una clase de equivalencia.
7. ¿En que consisten las pruebas de usabilidad.
8. ¿Qué es una prueba alfa?

Ejercicios

1. Haga el código correspondiente al diagrama de secuencia mostrado en la figura 68 inciso a y aplique pruebas de camino básico al mismo.
2. Calcule la complejidad ciclomática del ejercicio 1.
3. Dibuje un ejemplo de grafo de causa y efecto.
4. Basándose en el ejercicio uno del capítulo 4. Explique en que consistiría una prueba de unidad.
5. Investigue en que consiste el perfil de pruebas de UML.

Lecturas Recomendadas

Libros.

Binder, R.V. “Testing Object-Oriented Systems”, 1999. Addison Wesley.

Artículos:

Grimán Anna. C, Pérez María, Mendoza Luis. E “Estrategia de Pruebas para Software OO que garantiza Requerimientos No Funcionales”, Universidad de Bolivar, Caracas Venezuela.

Gutiérrez, J.J., Escalona M.J., Mejías M., “Generation of test cases from functional requirements”. 2006 ,A survey. 4º Workshop on System Testing and Validation. Potsdam. Germany.

5. Implantación y mantenimiento.

5.1 Implantación e Integración de casos de uso y componentes de software.

Obviamente el implantar o llevar a ejecución los casos de uso y realizar su integración, requiere el haber ejecutado todos los pasos previos necesarios para convertir una abstracción en una aplicación de software.
Los requisitos funcionales del usuario fueron primero plasmados en casos de uso esenciales, de los cuales se obtuvieron diagramas de actividades para poder diseñar los casos de prueba, también de estos casos de usos esenciales se obtuvieron los diagramas de secuencia de todos sus escenarios y usando como base estos diagramas de secuencia se obtuvieron los diagramas de clases, el diseño de las interfaces de usuario, el diseño de las bases de datos, después de varios refinamientos se procede al diseño de los caso de uso reales y sus interfaces, de allí a los diagramas de secuencia del sistema reales y a los diagramas de clases y la estructuración de las bases de datos, a la codificación de las clases, así como de la codificación de sus interfaces, y a la ejecución de los casos de prueba primero de caja blanca, posteriormente de caja negra en las cuales a integración de los casos de uso es llevada a cabo, nuevamente se realizan varios refinamientos como resultado de las pruebas, hasta que se puede realizar el empaquetamiento para obtener las componentes de software finales.

Una vez llevadas a cabo las pruebas de aceptación y de sistema, el producto de software es implantado en su entorno de producción.

5.2 Mantenimiento del software.

El mantenimiento en general se define como el grupo de actividades necesarias para que “algo”, siga funcionando correctamente, incuso se habla de mantenimiento preventivo, como aquel grupo de actividades ejecutadas sobre “algo” para evitar que este falle. Sin embargo desde el punto de vista del software a pesar del hecho de que también es un producto, sus fallos no son provocados por desgaste, como en el caso de otros productos, aunque definitivamente el tiempo si es un factor que puede influir pero de manera distinta, ya que con el tiempo se pueden hallar errores que aún cuando se hayan llevado a cabo los casos de pruebas, al no ser las pruebas exhaustivas siempre existe, aunque pequeña, una probabilidad de encontrar errores al estar en producción un sistema. Además con el tiempo es muy factible que se presenten nuevas necesidades a cubrir, al cambiar las reglas de

negocio.

Ningún sistema por mucho cuidado que se haya puesto en su construcción, queda nunca exento de requerir un mantenimiento. Esto se debe principalmente a que los sistemas computacionales son sistemas abiertos y dinámicos, y que los seres humanos que intervinieron en ellos son factibles siempre, no sólo de cometer errores, sino también de sufrir olvidos o ser incapaces de concebir como ciertas necesidades de procesamiento de información, debieron haberse transformado en requisitos funcionales, de lo cual se dieron cuenta hasta haber tenido experiencia con el uso del sistema.

Es tal la importancia del mantenimiento de software que ha sido estandarizado por la organización IEEE, en su norma IEEE 1219, en la cual se describe al mantenimiento como la actividad realizada para llevar a cabo las modificaciones después de haber implementado un producto de software, con el objeto de corregir fallas y defectos o mejorar su rendimiento o adaptarlo a un cambio originado por cambios en la organización", y también se hace referencia a el en la norma de calidad de software ISO 9126.

Antes de las nuevas metodologías orientadas a objetos para el análisis y diseño de software como la del Proceso Unificado por ejemplo, el costo del mantenimiento era bastante elevado, la causa principal de esto es que el software se elaboraba sin que el usuario diese su opinión sobre resultados parciales, inclusive se implementaba sin que el usuario hubiese llevado a cabo pruebas importantes, sólo algunas con supervisión del desarrollador.

Pero el aplicar estas metodologías no implica que no existan todavía razones poderosas por las cuales los sistemas requieran mantenimiento, una de ellas es la migración a otras plataformas tanto de hardware como de sistemas operativos. Es por ello que las últimas metodologías de desarrollo tienden a generar componentes de software que soporten dichos cambios. Sin embargo en la actualidad existen muchos sistemas que requieren de mantenimiento.

El mantenimiento es clasificado en cuatro tipos distintos dependientes de la causa que origina la modificación o alteración del software original y son:

- Preventivo.-Este tipo de mantenimiento actualmente forma parte del desarrollo del sistema ya que al ejecutar los casos de prueba se pueden encontrar errores que deberán ser corregidos antes de pasar a las pruebas de aceptación del sistema.
- Correctivo. Este se realiza cuando en el transcurso del uso del producto de software se encuentran fallas que anteriormente no se habían notado las cuales pueden ser: salidas

incorrectas, caídas del sistema al evaluar ciertos datos, ralentización del sistema entre otras.

- Adaptativo.- Este mantenimiento se requiere cuando las circunstancias del entorno cambian como por ejemplo cuando hubo el cambio de milenio, muchos sistemas que consideraban años únicamente de dos cifras tuvieron fallas importantes; ya que por ejemplo si en su lógica existía una comparación tal como si el año en una fecha final debe ser mayor que el año en la fecha inicial, entonces la fecha 1ero. de Enero del 2000 resultaba menor a la del 31 de Diciembre de 1999, porque únicamente se consideraban las dos última cifras de lo años. Otra causa de este mantenimiento es la migración a nuevas plataformas y/o sistemas operativos.
- Perfectivo.- Este tipo de mantenimiento ocurre cuando se desean adicionar nuevas funcionalidades al sistema, por ejemplo la ejecución de nuevos reportes o cuando se desea mejorar funcionalidades existentes, por ejemplo realizar búsquedas más eficientes, mejorar interfaces de usuario etc.,

Para realizar el mantenimiento se deben llevar a cabo los siguientes pasos:

- ✓ Analizar las peticiones de los cambios que se desean realizar al software para lo cual, es indispensable tener por lo menos una idea global de los componentes que lo conforman, incluido su propósito y principales funciones.
- ✓ Estudiar el manual de usuario para una mayor comprensión de los cambios solicitados desde el punto de vista de los usuarios, en caso de dudas o discrepancias, retornar con el usuario para aclarecer dichas dudas.
- ✓ Estudiar la documentación técnica del sistema para tener un entendimiento profundo y poder generar alternativas de solución para cada una de las peticiones de cambio solicitadas.
- ✓ Exponer y evaluar junto con el usuario las actividades, así como el costo y tiempo involucrado para cada alternativa generada, con la finalidad de elegir la más viable.
- ✓ Una vez elegidas las actividades involucradas en el cambio hacer su planeación, involucrando todas las etapas correspondientes desde el diseño hasta las pruebas e implementación de los cambios.
- ✓ Ejecutar el diseño y codificación y pruebas para los cambios solicitados.

Para la realización de las modificaciones que debe sufrir el software se requiere llevar a cabo una reingeniería del mismo; ya sea para crear nuevas componentes de software, para reconstruir un módulo existente e incluso para reconstruir el sistema en su totalidad.
La reingeniería involucra los siguientes pasos:

- Análisis del antiguo código. Primero se realiza de una manera estática, es decir, sin ejecutarlo,

sólo se identifican errores de sintaxis y el seguimiento de estándares y métricas, como la complejidad estructural, jerárquica y del flujo, y la de McCabe entre otras, y también se analizan los flujos de datos. Posteriormente se analiza el código de manera dinámica ejecutando el código para observar su comportamiento y resultados.

- Reestructuración del código. Se reestructura la aplicación para hacerla más fácil de entender y cambiar o para hacerla menos susceptible de incluir errores en cambios posteriores. Se lleva a cabo en tres niveles: A nivel de Análisis donde los modelos son transformados en otros de más fácil entendimiento. A nivel de diseño donde los diseños son cambiados por otros nuevos y a nivel de implementación, donde los diseños son enfocados a los datos y los procesos y obtener las clases y componentes correspondientes.
- Aplicación de técnicas de ingeniería inversa o ingeniería directa para la obtención del nuevo código. En la ingeniería inversa el código es analizado para llevarlo a sus representaciones de diseño, es decir a su nivel abstracto. Por otra parte a ingeniería directa es la tradicional hacer el código a partir del diseño.

Preguntas

1. ¿Qué significa mantenimiento en Ingeniería de Software?
2. ¿En que consiste la norma IEEE 1219?
3. ¿Cuál es la diferencia entre la norma ISO 9126 y la IEEE 1219?
4. ¿Cómo se clasifica el mantenimiento de software?
5. ¿En qué consiste el mantenimiento adaptativo?
6. ¿En que consiste la reingeniería?

Lecturas Recomendadas

- Pigosky T.M. "*Practical Software Maintenance*. New York", 1999, John Wiley & Sons.
- April & Abran . "*Software Maintenance Management"*. 2008, New York: John Wiley & Sons

.

Índice

Tabla de contenidos