Proyecto UTU - ANII - UDELAR

Tecnólogo Informático de Paysandú

Experiencia de capacitación docente en modalidad b-learning

Campus Virtual
Programa de Educación en Procesos Industriales
Consejo de Educación Técnico Profesional
Avenida Italia 6201 LATU Edificio “Los Cedros” 1er piso
Montevideo-Uruguay

Diseño de tapa: Gabriela Iaci
Diseño de interior y diagramación: Pablo Márquez
Corrección de Estilo: Lic. Andrés González
Compilación: Prof. Pablo Meyer
Coordinación: Prof. Gabriela Castro
ISBN: 978-9974-688-99-5

Consejo de Educación Técnico Profesional

Prof. Wilson Netto Marturet
Director General

Prof. Javier Landoni Seijas
Consejero

Mtro. Téc. César González Saldivia
Consejero

ANII

Dr. Rodolfo Silveira
Presidente

Ing. Luciana Balseiro
Ejecutiva

Facultad de Ingeniería

Dr. Ing. Héctor Cancela
Decano

Índice

Agradecimientos

Esta publicación recoge el material de cada uno de los cursos, jornadas de capacitación y formación técnico - docente que se dictaron en modalidad b_learning a través del Campus Virtual del CETP- UTU. Nuestro deseo es que este material sirva de base y guía para docentes y estudiantes de la carrera del Tecnólogo Informático; una pequeña contribución desde el convencimiento que la información debe estar accesible a todos y no depender de las circunstancias geográficas en que se encuentre el estudiante. Su existencia no sería posible sin la generosa participación de todas las personas que han compartido sus conocimientos y competencias.

Concretar un esfuerzo de esta naturaleza no habría sido posible sin la ayuda y colaboración de todos los docentes, compañeros de ruta virtuales y no virtuales, técnicos, integrantes de comisión de carrera del tecnólogo de Paysandú y de Montevideo que participaron del proyecto. A ellos se suma el apoyo desinteresado de la profesora Ana Iruleguy coordinadora del Tecnólogo de Paysandú, quien nos hizo sentir como en casa en cada instancia presencial, y de la Ingeniera Luciana Balseiro de ANII.

Un especial reconocimiento a la Fundación Ricaldoni quien administró nuestro proyecto.

Deseo expresar mi profundo agradecimiento al Ingeniero Luis Marco por alentarme y confiar la coordinación de este proyecto interinstitucional y a las autoridades de la Universidad del Trabajo del Uruguay que apoyaron esta iniciativa, en especial al profesor Wilson Netto y su visión integradora.

Nuestra gratitud al pro rector de enseñanza de la Universidad de la República, Dr. Luis Calegari, quien nos asistió en varias oportunidades y al Ingeniero Héctor Cancela quien, al inicio del proyecto, desde su cargo como director del InCo, y luego como decano de la Facultad de Ingeniería, tuvo un rol preponderante en todo el desarrollo del mismo.

A nuestras familias por su enorme paciencia.

Extiendo a todos ustedes mi profundo aprecio.

Prof. Gabriela Castro del Pino
Coordinadora del Campus Virtual

Presentación

Cuando una sociedad define modificar su lugar en el concierto mundial respecto a la distribución internacional del trabajo, un requerimiento imprescindible es el desarrollo de su población, donde el conocimiento ocupa un lugar relevante.

La actividad desarrollada, como su publicación, son una muestra más del camino que ha tomado la educación para abordar éstos desafíos.

La Universidad de la República y la Universidad del Trabajo del Uruguay no sólo desarrollan carreras conjuntas, sino que contribuyen a generar una nueva cultura interinstitucional modificando esa concepción y organización balcanizada de nuestra sociedad.

La modalidad en que se desarrolló la actividad también muestra como las TICs permiten generar nuevas oportunidades de socialización del conocimiento, construyendo ámbitos de intercambio entre personas y profesionales radicados en distintos puntos del país, la región, o el mundo.

Es de destacar la muestra de pasión y compromiso, componente sustantivo de la profesión docente, de todos quienes han participado en esta actividad.

Por su aporte personal y el de sus equipos expreso un agradecimiento muy especial al Ing. Héctor Cancela y a la Prof. Gabriela Castro.

Profesor Wilson Netto Marturet

Es una tarea muy grata el escribir unas breves palabras para presentar esta publicación que recoge el material generado en el transcurso del proyecto de capacitación docente en modalidad b-learning, orientado a promover el desarrollo de la sede Paysandú del Tecnólogo en Informática, carrera mixta UTU/UDELAR.

Este proyecto, que fue posible gracias a la iniciativa y visión de la UTU y el apoyo y financiación de la ANII, contó desde su puesta en marcha con el apoyo entusiasta del Instituto de Computación y la Facultad de Ingeniería, y el soporte de la Fundación Julio Ricaldoni, convirtiéndose en un verdadero caso de éxito de cooperación interinstitucional.

Las características del proyecto lo hacían sumamente atractivo por varios motivos. Desde un punto de vista práctico, la sede Paysandú del Tecnólogo en Informática apenas comenzaba su actividad cuando el proyecto fue formulado y puesto en marcha. Mediante su ejecución fue posible apoyar la formación de un conjunto importante de docentes de diversas disciplinas, que estaban participando o se incorporaron posteriormente en el equipo a cargo del dictado de esa carrera. Del punto de vista conceptual, un proyecto cuyo objetivo es formar a formadores, y particularmente utiliza la informática (en este caso concreto, un espacio virtual de aprendizaje) como herramienta para transmitir conocimientos en distintas áreas temáticas de la informática, tiene una formulación recursiva y un aspecto demostrativo del poder de las tecnologías de la información que podemos catalogar de irresistible.

Adicionalmente, el proyecto ha permitido generar un material que será seguramente de interés para docentes actuales y futuros de las diversas sedes de la carrera de Tecnólogo en Informática, por lo que nos congratulamos de que esta publicación permita su difusión entre todos aquellos que puedan aplicarlo.

Para cerrar, agradecemos al Prof. Wilson Netto, Director de UTU/CETP durante la realización del proyecto, que con su calidez, empuje y disposición a colaborar permitió generar este espacio de acción conjunta; a la Prof. Gabriela Castro, que coordinara las distintas acciones del proyecto, con gran eficiencia, y calidad humana; y a todos los docentes que participaron, sea preparando y orientando estos cursos, sea responsabilizándose de su propia formación mediante una participación activa en estas instancias de formación muy intensas.

Dr. Ing. Héctor Cancela

Decano de la Facultad de Ingeniería

El anhelar un País Productivo implica contar con una masa crítica de recursos humanos formados, en regiones donde hoy no se encuentran fácilmente.

Para alcanzar esa meta debemos optimizar una interacción entre las instituciones que nos posibilite trabajar como un verdadero Sistema.

El libro que hoy llega a sus manos recoge el trabajo de muchos compañeros, quienes han demostrado que innovando en estrategias educativas se puede fortalecer la desconcentración de la enseñanza terciaria. A todos ellos muchas gracias por su enorme aporte el cual, seguramente, despertará nuevas iniciativas.

Ing. Tecnol. Luis Marco

3. Programación avanzada

Ing. Daniel Calegari

Diseño
Diseño de la estructura de una colaboración

Programación Avanzada

Diseño
Diseño de la Estructura de una Colaboración

Contenido

Introducción

- La asignación de responsabilidades ha sido completada.
- La parte dinámica de la colaboración que se está diseñando ha sido determinada.
- Habiendo finalizado la construcción de los diagramas de comunicación es posible especificar la parte estructural de la colaboración.

Introducción (2)

- Esta especificación se realizará mediante los diagramas de clases de UML .
- Estos diagramas:
  - Ilustran la estructura de la solución
  - Están anotados con información de diseño, como,por ejemplo,operaciones y navegabilidades.
- Al artefacto resultante lo llamamos **Diagrama de Clases de Diseño** (DCD) y será incluido en el Modelo de Diseño.

Diagrama de Clases de Diseño

- Un Diagrama de Clases de Diseño especifica la estructura de una colaboración.
- Los elementos que contiene son representaciones gráficas de algunos elementos de diseño contenidos en el modelo.
- Los elementos a incluir son solamente aquellos que sean necesarios para solucionar el(los)caso(s)de uso realizado/s por la colaboración.

Diagrama de Clases de Diseño (2)

- Elementos de diseño a incluir:
  - Clases, asociaciones y atributos.
  - Navegabilidades de asociaciones.
  - Operaciones de clases y existencia de métodos.
  - Interfaces con sus operaciones .
  - Información acerca del tipo de los atributos y de los valores devueltos por las operaciones (incluyendo datatypes).
  - Generalizaciones entre clases o interfaces.
  - Dependencias entre elementos.

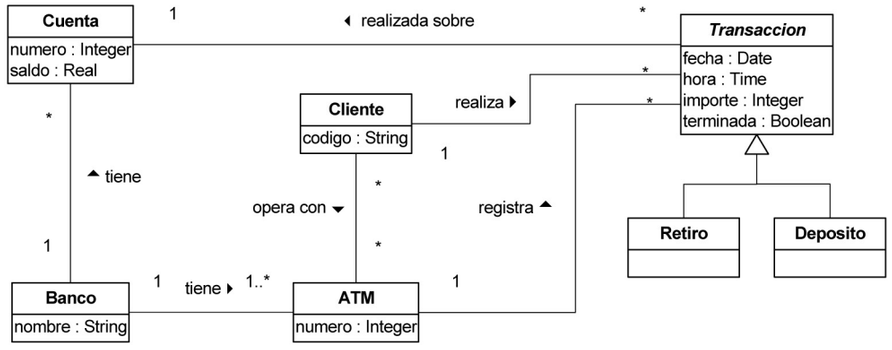
Construcción de un DCD

- Para la construcción de un DCD:
  1. Identificar todas las clases que participan de la solución de los casos de uso. Hacer esto analizando los diagramas de comunicación.
  2. Incluirlas en un el diagrama de clases .
  3. Replicar los atributos de los conceptos correspondientes en el Modelo de Dominio, agregando aquellos nuevos que sean necesarios.
  4. Agregar las operaciones correspondientes a cada clase analizando los diagramas de comunicación .

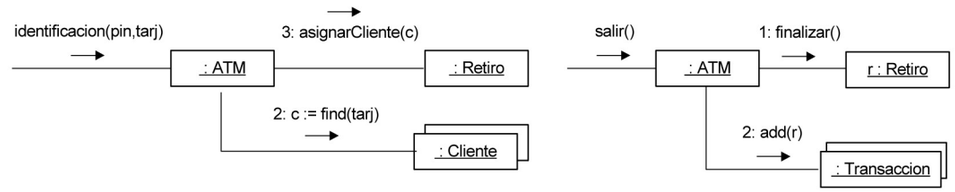
Construcción de un DCD (2)

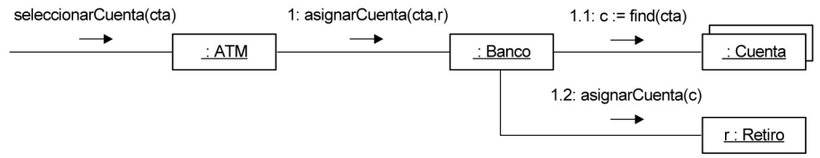
- Para la construcción de un DCD (cont.):
  5. Agregar la información de tipos a los atributos y operaciones.
  6. Agregar las asociaciones necesarias para permitir las visibilidades por atributo requeridas en los diagramas de comunicación.
  7. Agregar navegabilidades para indicar la dirección de cada visibilidad por atributo.
  8. Agregar dependencias para reflejar los demás tipos de visibilidades existentes.
  9. Agregar interfaces, fábricas y datatypes.

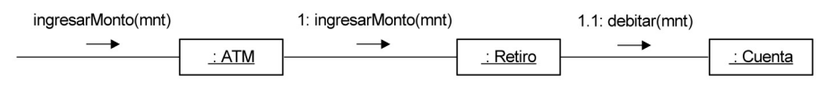
Construcción de un DCD
Información Previa (Dominio)



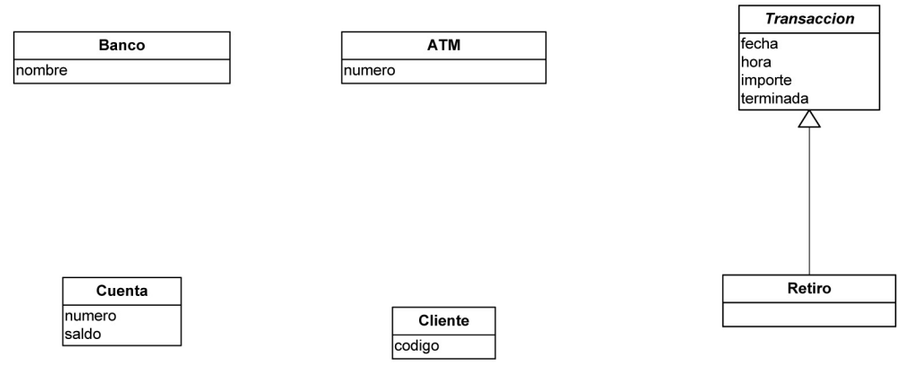
Construcción de un DCD
Información Previa (Interacciones)



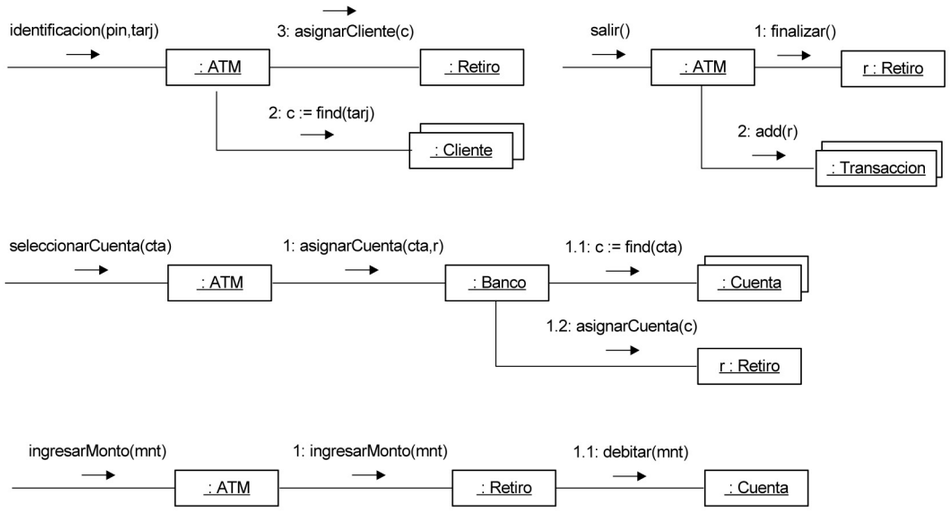




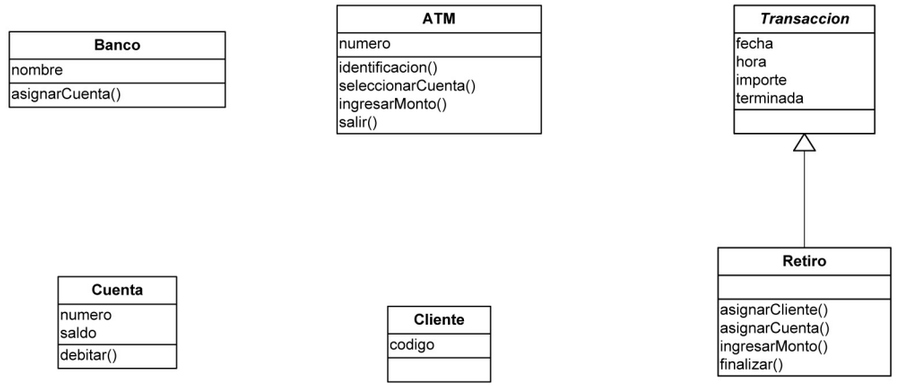
Identificar las Clases e Ilustrarlas
Pasos 1, 2 y 3



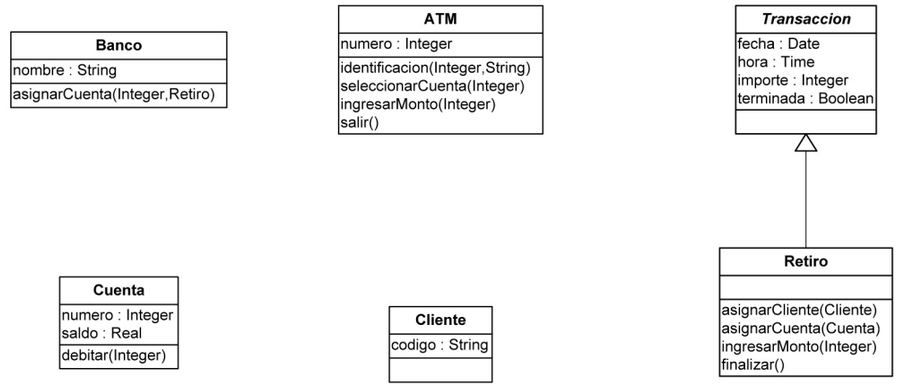
Construcción de un DCD
Información Previa (Interacciones)



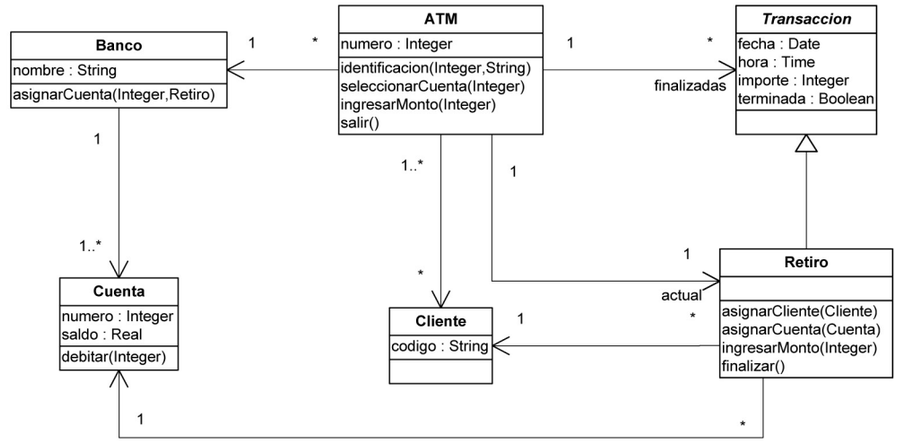
Agregar Operaciones y Métodos
Paso 4



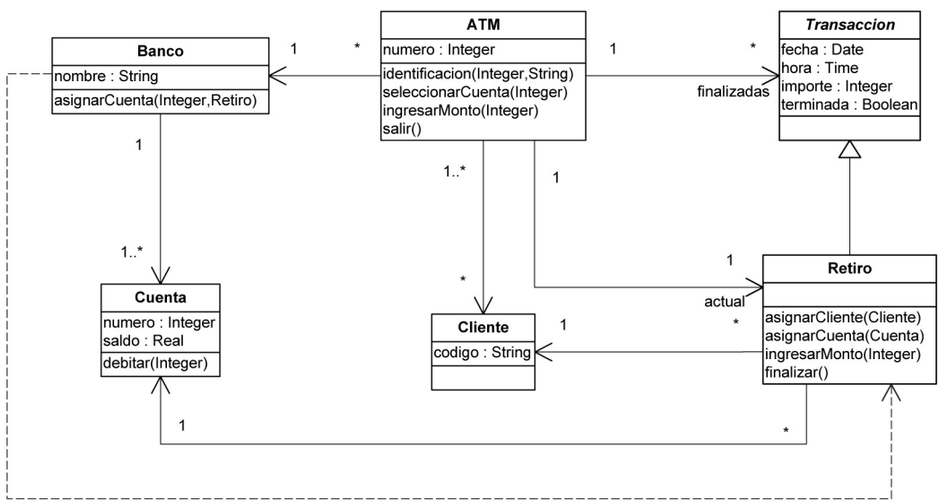
Agregar Información de Tipos
Paso 5



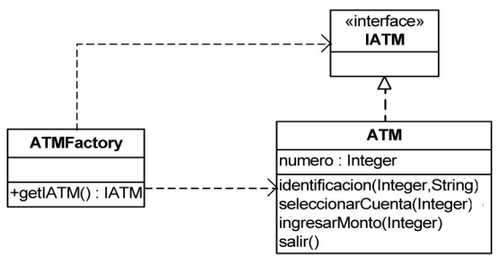
Agregar Asociaciones y Navegabilidad
Pasos 6 y 7



Agregar Dependencias
Paso 8



Agregar interfaces, fábricas y datatypes
Paso 9



Inclusión de Operaciones

- Operación *create*:
  - La operación *create* es utilizada para la creación de instancias.
  - Esta forma es propia de UML e independiente de todo lenguaje de programación.
  - Este mensaje se corresponde con los constructores de clases.
  - Los constructores están siempre presentes en las clases,por lo que es común omitirlos en los diagramas de clases de diseño.

Inclusión de Operaciones (2)

- Operaciones de acceso:
  - Son utilizados para obtener el valor de un atributo (*get*) o para modificarlo (*set*).
  - Lo usual es declarar los atributos como privados y necesitar este tipo de operaciones.
  - Sin embargo,se las excluye de los diagramas.
  - Por defecto,se asume que un atributo tiene su *get* y *set* asociado.
  - Es posible indicar que para un atributo no se brindará la operación set correspondiente aplicándole la restricción {readOnly}.

Inclusión de Operaciones (3)

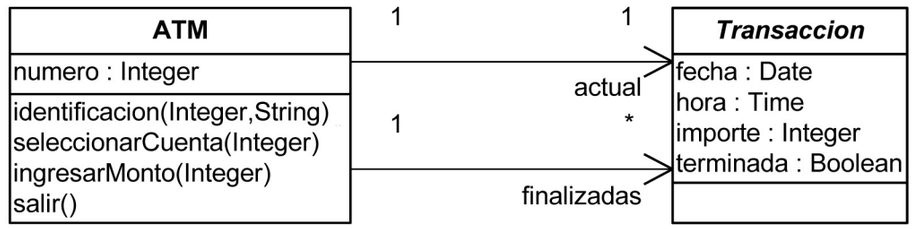
- Operaciones de acceso (cont.)
  - Ejemplo: la implementación de la clase Empleado contendrá las operaciones:
    - `calcularAportes()`
    - `asignarCliente()`
    - `getNombre()`
    - `setNombre()`
    - `getSueldo()`

Inclusión de Colecciones

- Las colecciones (tratadas como fuera indicado) usualmente disponen todas de las mismas operaciones.
- Por tal razón,no aportarían mayor información al diagrama y es común omitirlas.
- La necesidad de una colección se deriva de las multiplicidades.

Inclusión de Colecciones (2)

- Ejemplo



Un ATM tendrá asociado:
- Una sola transacción actual.
- Una colección de transacciones finalizadas.

Diseño
Guías para el abordaje del diseño

Programación Avanzada

Diseño

Guías para el Abordaje del Diseño

Contenido

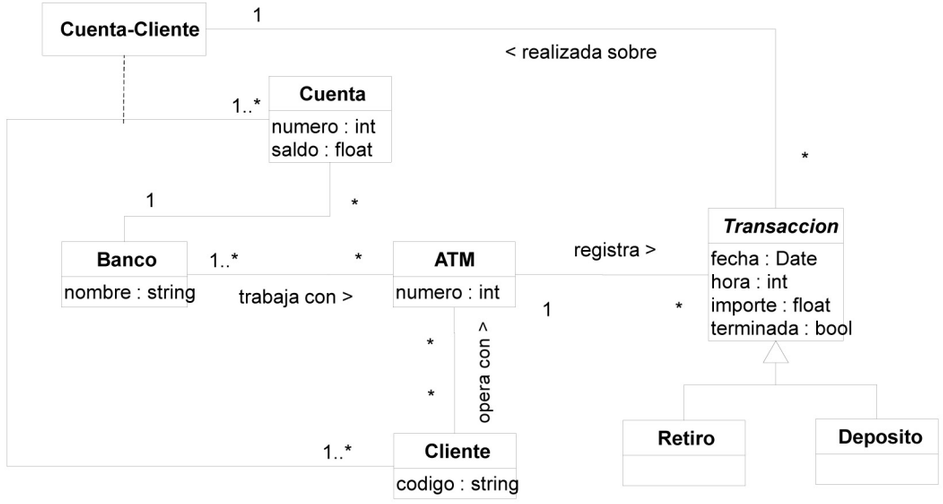
Introducción

- Se desea abordar la etapa de diseño con un enfoque sistemático.
- Se presentarán pautas para organizar de mejor forma la tarea.
- Se ejemplificarán las mismas por medio de un caso de estudio.

Caso de Estudio

- Gestión de cuentas en bancos a través de ATMs (cajeros automáticos):
  - A través de una red de ATMs, los clientes acceden a sus cuentas sobre las cuales realizan transacciones (depósitos y retiros).
  - Las cuentas pueden ser compartidas por más de un cliente.

Caso de Estudio
Modelo de Dominio



Diseño de la Estructura
Errores Comunes

- No incluir las dependencias existentes.
- Omitir la definición de los datatypes.
- No incluir interfaces, controladores ni fábricas.
- Sobrecargar el diagrama con operaciones omitibles (create, set, etc.).
- Incluir colecciones como clases innecesariamente.

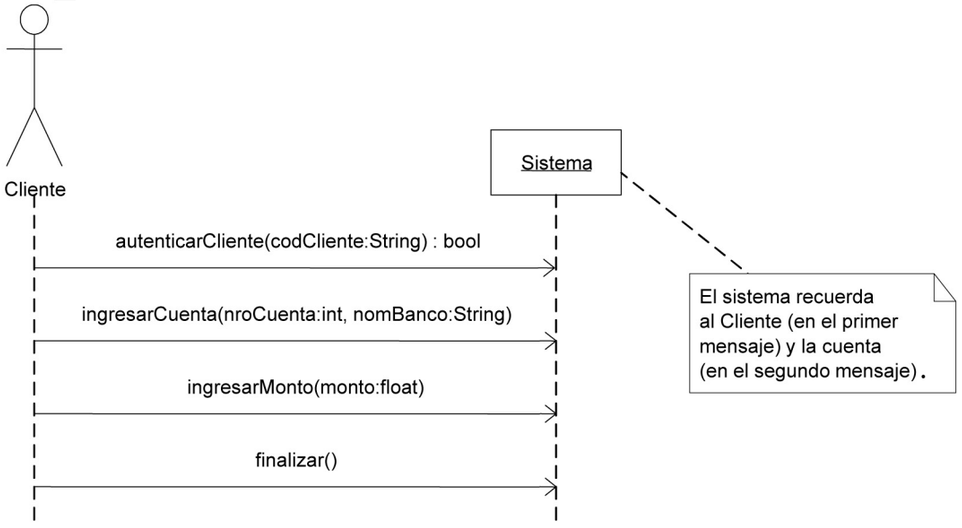
Caso de Estudio
Caso de Uso

Nombre	Retiro de Cuenta	Actores	Cliente
Sinopsis	El caso de uso comienza cuando el cliente inserta su tarjeta en el cajero e ingresa su clave de usuario. Tras validar al cliente, el sistema recibe el nombre del banco y el número de cuenta para iniciar la transacción de retiro correspondiente. El cliente ingresa el monto que desea retirar de la cuenta y el sistema realiza el débito. Finalmente, el cliente retira su tarjeta.		

Caso de Estudio
Descripción de Operaciones

- **`autenticarCliente (codCliente:String):bool`**
  - Valida la existencia del cliente.
- **`ingresarCuenta (nroCuenta:int, nomBanco:String)`**
  - Obtiene la cuenta **`nroCuenta`** del banco **`nomBanco`** sobre la cual se realizará la transacción.
- **`ingresarMonto (monto:float)`**
  - Realiza la transacción de débito por el monto indicado sobre la cuenta del cliente.
- **`finalizar ()`**
  - Finaliza la operativa del sistema.

Caso de Estudio
DSS con Memoria



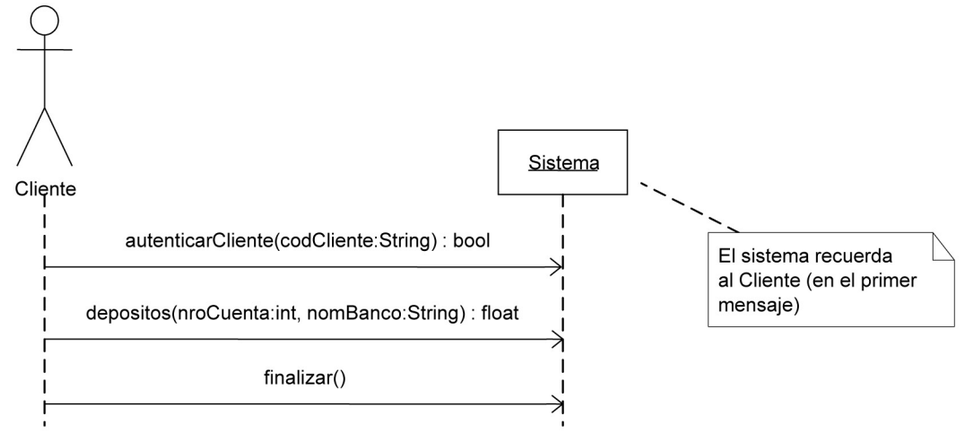
Caso de Estudio
Descripción de Operaciones (2)

- **`autenticarCliente (codCliente:String):bool`**
  - Misma operación que en el DSS anterior.
- **`depositos (nroCuenta:int, nomBanco:String):float`**
  - Devuelve la suma de los montos de todos los depósitos realizados en la cuenta **`nroCuenta`** del banco **`nomBanco`**.
- **`finalizar ()`**
  - Misma operación que en el DSS anterior.

Caso de Estudio
Caso de Uso (2)

Nombre	Consulta de Depósitos	Actores	Cliente
Sinopsis	El caso de uso comienza cuando el cliente inserta su tarjeta en el cajero e ingresa su clave de usuario. Tras ingresar los datos de validación (igual que en el caso de uso Retiro de Cuenta), el cliente indica el nombre del banco y el número de cuenta sobre la cual desea consultar el total de depósitos (históricos). Posteriormente, el sistema calcula el total de depósitos histórico y lo devuelve. Finalmente, el cliente retira su tarjeta.		

Caso de Estudio
DSS con Memoria (2)



Guías para el Abordaje del Diseño

- El abordaje de la etapa de diseño puede realizarse sistemáticamente.
- Por ejemplo, considerando estos pasos:
  1. Organizar Operaciones.
  2. Definir Ubicación de Instancias.
  3. Definir Colaboraciones.
  4. Diseñar Colaboraciones.

Guías para el Abordaje del Diseño
Organizar Operaciones

- Definir los Controladores a utilizar.
- Definir las Interfaces del Sistema que contendrán las operaciones del sistema.
- Organizar operaciones según:
  - Afinidad temática (según dominio).
  - Afinidad funcional (según objetivos).
  - Casos de Uso.
- Definir la Fábrica de controladores.

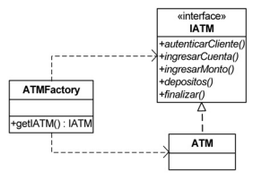
Guías para el Abordaje del Diseño
Organizar Operaciones (2)

- Al definir Controladores, considerar:
  - Operaciones repetidas en casos de uso.
  - Memoria del Sistema.
- Si un Controlador realiza una Interfaz del Sistema, asigna un método a <u>todas</u> las operaciones presentes en ella.

Guías para el Abordaje del Diseño
Ejemplo (Organizar Operaciones)

- ¿Algún concepto del dominio podría ser un Controlador?, ¿ATM?, ¿Banco?

Una posible opción:



Guías para el Abordaje del Diseño
Definir Ubicación de Instancias

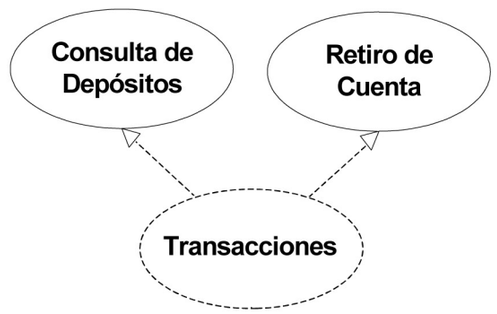
- Diferenciar:
  - Las colecciones que pueden ser alojadas en un controlador (ej: ATM, Banco).
  - Las que serán accedidas únicamente a través de otra clase (ej: Cuenta accesible a través de Banco).
  - En caso de ser necesario, alojar separadamente una colección que sea compartida entre varios controladores.

Guías para el Abordaje del Diseño
Definir Colaboraciones

- Una colaboración realiza uno o más casos de uso.
- Agrupar casos de uso con cierta afinidad:
  - Comúnmente afinidad temática,pero no hay una regla estricta.
- Definir una colaboración por cada grupo de casos de uso asignándole un nombre.
- Priorizar las colaboraciones según el impacto esperado sobre el diseño.

Guías para el Abordaje del Diseño
Ejemplo (Definir Colaboraciones)

- Definir una sola colaboración para ambos casos de uso es beneficioso ya que están relacionados:



Guías para el Abordaje del Diseño
Diseñar Colaboraciones

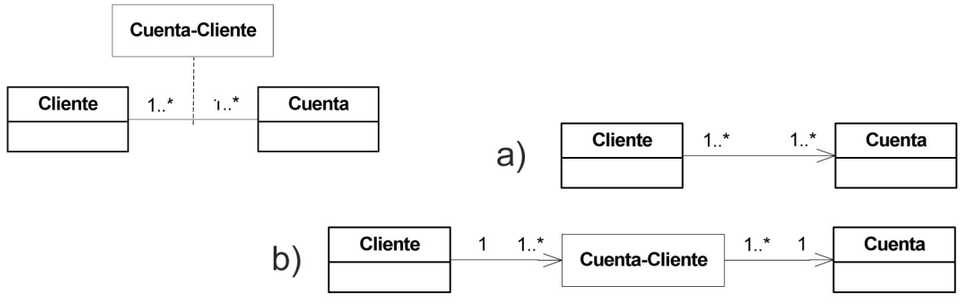
- Diseñar cada colaboración en orden de prioridad:
  - Realizar diagramas de comunicación para las operaciones del sistema involucradas.
  - Considerar:(a) criterios de asignación de responsabilidades, (b) decisiones tomadas en iteraciones anteriores (consistencia) y (c) nuevos problemas de diseño.
  - Realizar el diagrama de clases de diseño.

Guías para el Abordaje del Diseño
Ejemplo (Diseñar Colaboraciones)

- ¿Cómo se asignan responsabilidades?
  - ¿Quién crea las transacciones?
    - ¿ATM?, ¿Banco?, ¿Cuenta?
  - ¿Quién es el experto en calcular el total de depósitos realizados?
    - ¿Banco?, ¿Cuenta?, ¿Cliente?
  - ¿Qué visibilidades se necesitan?
    - ¿ATM → Transacción?, ¿Banco → Cuenta?, ¿Cliente → Transacción?, ¿ATM → Cuenta?

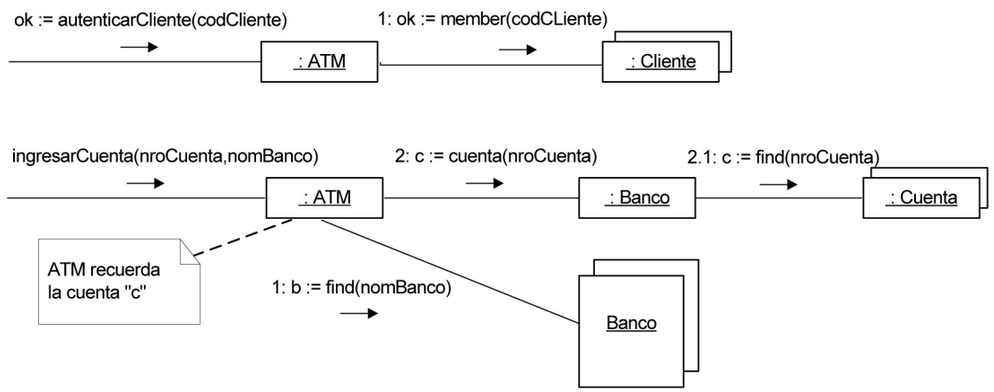
Guías para el Abordaje del Diseño
Ejemplo (Diseñar Colaboraciones) (2)

- ¿Qué sucede con los tipos asociativos?
  - Si poseen información relevante puede convenir mantenerlas:

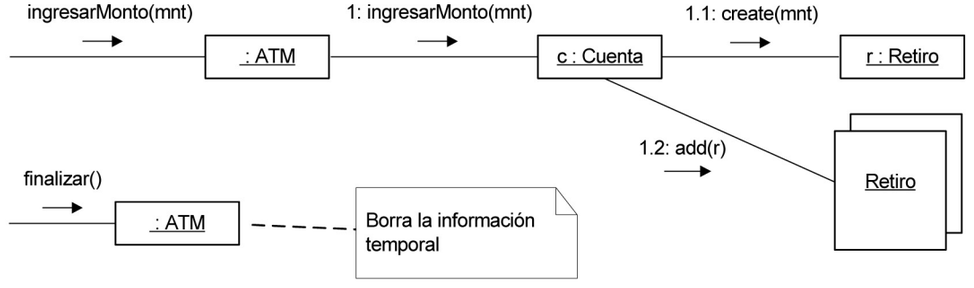


Guías para el Abordaje del Diseño
Ejemplo (Diseñar Colaboraciones) (3)

- Una posible solución:

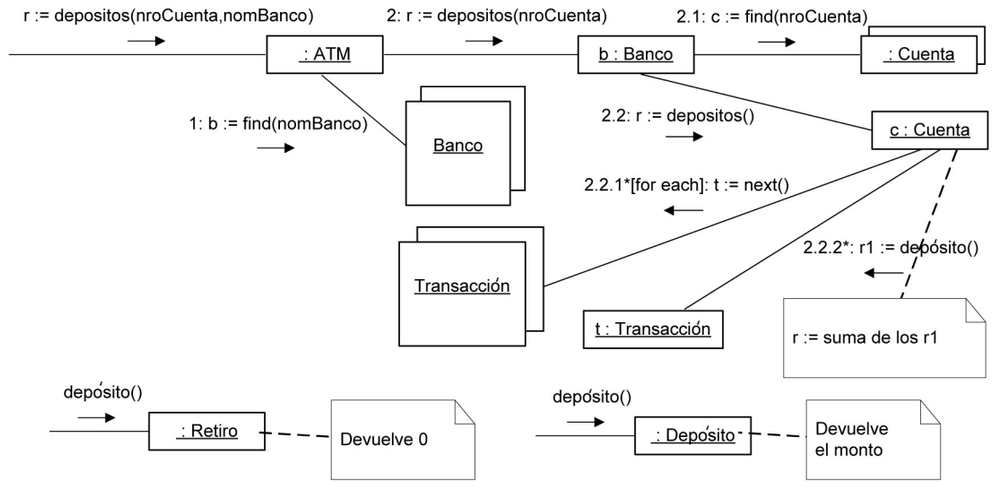


Guías para el Abordaje del Diseño
Ejemplo (Diseñar Colaboraciones) (4)



- Otras opciones:
  - Asociar la transacción con el cliente.
  - Delegar las transacciones al banco.

Guías para el Abordaje del Diseño
Ejemplo (Diseñar Colaboraciones) (5)



Implementación
Generación de código

Introducción

- Propósito: realizar la implementación de una parte del diseño (una colaboración).
- El resultado es código fuente en la forma de Elementos de Implementación.
- Una tarea de implementación se enfoca en obtener cierta funcionalidad (al implementar la realización de un caso de uso) que implica la implementación de diferentes elementos de diseño que contribuyan a dicha funcionalidad.

Modelo de Implementación

- El Modelo de Implementación representa la composición física de la implementación.
- Está expresado principalmente en términos de Elementos de Implementación.
- Éstos son típicamente elementos físicos como archivos, pero también directorios:
  - Archivos de código (fuentes, binarios, ejecutables).
  - Archivos de datos y configuración .
- Dichos elementos pueden organizarse en Subsistemas de Implementación.

Modelo de Implementación (2)

- Contenido:
  - **Introducción:** Breve descripción que sirve como introducción al modelo.
  - **Subsistemas de implementación:** Conjuntos de Elementos de Implementación, definen una jerarquía.
  - **Elementos de implementación:** Todos los archivos que conforman la implementación del sistema, contenidos en los subsistemas.

Modelo de Implementación (3)

- Contenido (cont.):
  - **Relaciones:** Las relaciones del modelo entre elementos de implementación, contenidas en los subsistemas.
  - **Diagramas:** Representación de los elementos del modelo (p.e. dependencias estáticas entre fuentes, dependencias de tiempo de ejecución entre ejecutables).

Implementación de una Colab.

- Para implementar una colaboración (que realice caso(s)de uso):
  - Implementar la estructura de la colaboración:
    - Implementar interfaces.
    - Implementar clases:
      - Implementar atributos .
      - Implementar operaciones .
    - Implementar relaciones:
      - Implementar generalizaciones.
      - Implementar realizaciones.
      - Implementar asociaciones.
  - Implementar las interacciones de la colaboración:
    - Implementar métodos.

Implementar la Estructura
Implementar Interfaces

- Las interfaces se implementan directamente a partir del DCD.
- Las operaciones se obtienen de la propia especificación de la interfaz.
- **Advertencia:** algunos lenguajes de programación no proveen una construcción para implementar directamente interfaces:
  - En esos casos,se suele implementar una clase abstracta, sin atributos,y con todas sus operaciones abstractas.

Implementar la Estructura
Implementar Clases

- La implementación de las clases se hace en forma directa a partir del DCD.
- Los lenguajes de programación orientados a objetos incluyen una construcción para este fin (la clase).
- Los atributos y operaciones se obtienen de la propia especificación de la clase:
  - Se incluyen los constructores y destructor.
  - También las operaciones de acceso o modificación de los atributos.

Implementar la Estructura
Implementar Clases (2)

- Ejemplo en C++

```
class Empleado {
   private:
      String nombre;       //atributo
   public:
      Empleado();          //constructor por defecto
      Empleado(String);    //constructor común
      ~Empleado();         //destructor
      String getNombre();          //op. de acceso
      void setNombre(String nom)   //op. de modif.
      virtual float getPago() = 0; //op. abstracta
};
```

Implementar la Estructura
Implementar Relaciones

- Las relaciones entre elementos de diseño empleadas son:
  - Generalizaciones.
  - Realizaciones.
  - Asociaciones.
  - Dependencias.

Implementar la Estructura
Relaciones – Generalizaciones

- Las generalizaciones se obtienen directamente del DCD.
- Los lenguajes de programación orientados a objetos proveen una construcción para esto:
  - En la declaración de la clase se especifica su ancestro (muchos lenguajes permiten sólo uno).
- Ejemplos:
  - Java: `class Jornalero extends Empleado`
  - C++: `class Jornaleo : public Empleado`
  - C#: `class Jornalero : Empleado`

Implementar la Estructura
Relaciones – Realizaciones

- Las realizaciones también se obtienen directamente del DCD.
- Los lenguajes de programación que no proveen interfaces tampoco proveen realizaciones:
  - En la declaración de la clase se especifica la(s) interfaz(ces) que realiza.
  - En C++ se utiliza una generalización.
- Ejemplos:
  - Java: `class ATM implements IRetiro`
  - C#: `class ATM : IRetiro`
  - C++: `class ATM : IRetiro`

Implementar la Estructura
Relaciones – Asociaciones

- Los lenguajes de programación generalmente no proveen una construcción específica para la implementación de asociaciones.
- Para que una clase A pueda estar asociada a una clase B se suele incluir un atributo en A:
  - Este atributo no pertenece al conjunto de atributos definidos en el diseño,por lo que se lo denomina “pseudoatributo”.
- A través del pseudoatributo una instancia de A puede mantener una referencia a otra de B y así implementar el link.

Implementar la Estructura
Relaciones – Asociaciones (2)

- Se define un pseudoatributo en A solamente si la asociación es navegable hacia B.
- El tipo de un pseudoatributo para A depende de la clase B, pero también de la multiplicidad en el extremo de la asociación del lado de B.
- Se distinguen dos casos dependiendo del máximo de dicha multiplicidad:
  - Si el máximo es 1: el pseudoatributo es de tipo B.
  - Si el máximo es mayor que 1 (típicamente *): el pseudoatributo es de tipo Coleccion(B).

Implementar la Estructura
Implementar Interfaces (2)

- Ejemplo en Java:

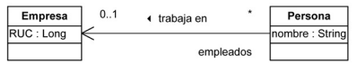
```
public interface IRetiro {
   public void identificacion(int,String);
   public void seleccionarCuenta(int);
   ...
}
```

- Ejemplo en C++:

```
class IRetiro {
   public:
      virtual void identificacion(int,String) = 0;
      virtual void seleccionarCuenta(int) = 0;
      ...
}
```

Implementar la Estructura
Relaciones – Asociaciones (3)

- Caso 1:

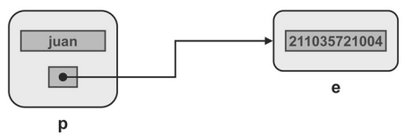


  - Ejemplo en C++

```
class Persona {
   private:
      String nombre;
      Empresa * empresa; // pseudoatributo
   public:
      ...
};
```

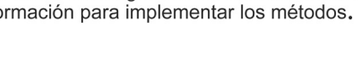
Implementar la Estructura
Relaciones – Asociaciones (4)

- Caso 1 (cont.):
  - Una persona puede tener una referencia a una empresa:



Implementar la Estructura
Relaciones – Asociaciones (5)

- Caso 2:

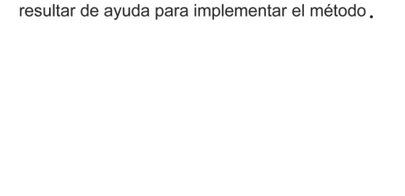


  - Ejemplo en Java

```
class Empresa {
   private long RUC;
   private Collection empleados; // pseudoatributo
   ...
};
```

Implementar la Estructura
Relaciones – Asociaciones (6)

- Caso 2 (cont.):
  - Una empresa tiene una colección de referencias a personas:



Implementar la Estructura
Relaciones – Asociaciones (7)

- Caso 2 (cont.):
  - La elección de la estructura de datos que implemente la colección se realiza en función de simplicidad, disponibilidad, requerimientos de eficiencia, etc.
  - En casos en que el extremo de asociación tenga aplicada la restricción {ordered} es necesario utilizar una colección lineal con operaciones de acceso a los elementos por posición.
  - Muchos ambientes de programación cuentan con bibliotecas de clases con diferentes tipos de colecciones predefinidas.

Implementar la Estructura
Relaciones – Dependencias

- Las dependencias se declaran en la definición de un elemento para tener visibilidad sobre otros
- Esto se hace cuando en el DCD existe una dependencia desde un elemento A hacia otro B.
  - Una asociación navegable, una generalización y una realización son también formas de dependencia.
- En C++ se utiliza `#include`.
- En Java se utiliza `import`.
- En C# se utiliza `using`.

Implementar las Interacciones
Implementar Métodos

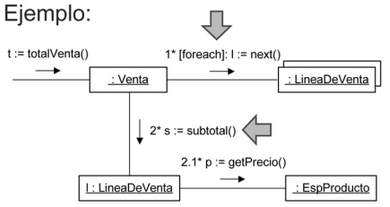
- La implementación de la estructura conduce a la definición de los elementos de diseño junto con sus relaciones.
- Las clases incluyen sus operaciones pero no los métodos asociados.
  - Esto significa que no existen invocaciones implementadas,por lo que aún no hay comportamiento
- A partir de los diagramas de interacción se extrae información para implementar los métodos.

Implementar las Interacciones
Implementar Métodos

- Un diagrama de comunicación no tiene como objetivo servir de pseudocódigo.
- Sin embargo,generalmente ilustra la lógica general de las operaciones.
- Al implementar el método asociado a la operación `op()` en una clase A:
  - Se busca el diag. de comunicación que incluya un mensaje `op()` llegando a una instancia de A.
  - La interacción anidada en ese mensaje debería resultar de ayuda para implementar el método.

Implementar las Interacciones
Implementar Métodos (2)

- Ejemplo:



Para implementar el método asociado a `totalVenta()` observamos la interacción anidada en el mensaje (en el primer nivel) en el diagrama de comunicación.

Implementar las Interacciones
Implementar Métodos (3)

- Ejemplo (cont.):

```
class Venta {
   public float totalVenta() {
      float total = 0;
      LineaDeVenta ldv;
      IteratorLineaDeVenta it = lineas.getIterator();

      while (it.hasCurrent()) {
         ldv = it.current();
         total = total + ldv.subtotal();
         it.next();
      }
      return total;
}
```

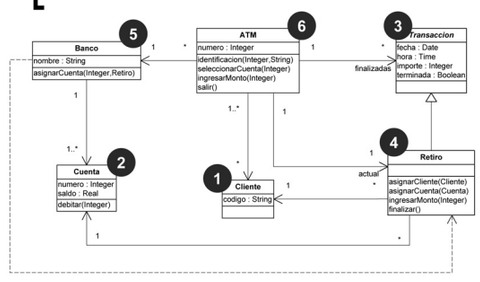
Sugerencias

- Antes de implementar una clase desde cero es recomendable considerar si el código existente puede ser reutilizado o adaptado.
- Comprender en qué lugar de la arquitectura encaja la implementación ayuda a:
  - Identificar oportunidades de reuso.
  - Asegurar que el código nuevo sea coherente con el del resto del sistema.

Sugerencias (2)

- Orden de implementación de las clases:
  - Las clases deben ser implementadas comenzando por las menos acopladas y finalizando por las más acopladas.
  - De esta forma,las clases van disponiendo de todos los elementos necesarios para su implementación.
  - Esto permite que,al terminar de implementar una clase,se pueda testear inmediatamente.

Sugerencias (3)



Implementación
Manejo de objetos

Introducción

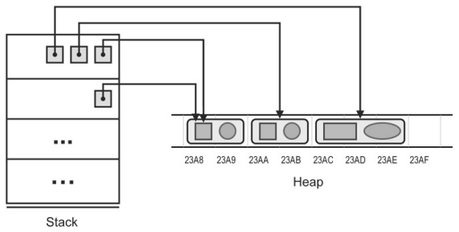
- Los objetos son manipulados a través de referencias.
- Dependiendo de cómo los lenguajes de programación las implementen aplican ciertas consideraciones tanto a la manipulación como a la destrucción de objetos.
- La identidad requiere que los objetos sean compartidos.
- Esto hace que las copias necesiten ser examinadas en detalle.

Referencias

- En tiempo de ejecución los objetos no son alojados en el stack sino en el heap.
- La forma de acceder a un objeto es mediante referencias.
- Una referencia es una variable (tipada) que es alojada en el stack (o en el heap si está dentro de un objeto) tal que:
  - No identifica a ningún objeto (*void*).
  - Identifica a un objeto particular de una determinada clase (*attached*).

Referencias (2)

- Ejemplo:



Referencias (3)

- En algunos lenguajes de programación las referencias se implementan explícitamente:
  - En C++ las referencias se implementan mediante punteros:

```
Empleado * e1;
// 'e1' es una referencia a un objeto
// de clase Empleado y no un objeto.

Empleado e2;
// 'e2' sí es un objeto (en el stack).
```

Referencias (4)

- Otros lenguajes manejan referencias en forma implícita:
  - En Java y C# no es posible definir objetos en el stack sino únicamente referencias:

```
Empleado e;
// 'e' es una referencia a un objeto
// de clase Empleado y no un objeto.
```

Referencias (5)

- Hacer que una referencia sea *void*:
  - En C++: `e = NULL`
  - En Java y C#: `e = null`
- Hacer que una referencia sea *attached* (en cualquiera de los tres lenguajes):
  - Crear un objeto y adjuntar la referencia a él
    `e = new Jornalero();`
  - Adjuntar la referencia a un objeto obtenido a través de otra referencia:
    `e1 = e2;`

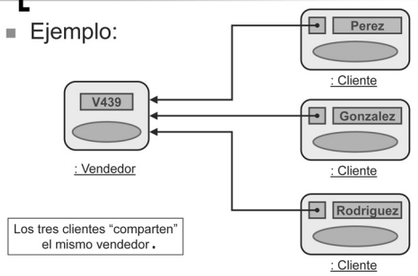
Esto es asignación de referencias y no de objetos.

Objetos Compartidos

- En sistemas orientados a objetos es usual que un objeto sea "conocido" por otros varios objetos.
- La identidad requiere que dichos objetos referencien al mismo objeto y no a copias de él.
- Eso implica que el objeto será "compartido" por otros.
- Esto se logra teniendo en cada objeto una referencia al objeto compartido.

Objetos Compartidos (2)

- Ejemplo:



Copia de Objetos

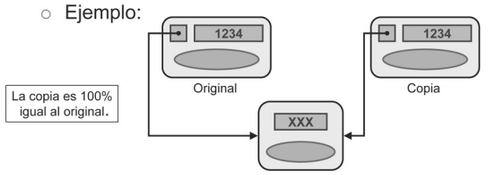
- La identidad y la necesidad de compartir objetos hace que, en general, no sea correcto copiar objetos.
- Recordar que una copia de un objeto es otro objeto que luego de la copia tiene propiedades iguales a las del original.
- A partir de la copia ambos elementos evolucionan independientemente.

Copia de Objetos (2)

- En determinadas situaciones es aceptable la copia de objetos.
- Distinguiremos tres casos:
  - Objetos que implementan data values.
  - Objetos que son instancias de clases del diseño.
  - Objetos que representan colecciones.
- A su vez, distinguimos dos enfoques de realizar copias de objetos:
  - Copia plana.
  - Copia en profundidad.

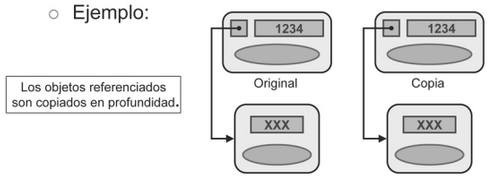
Copia de Objetos (3)

- Copia plana:
  - Su resultado es un objeto exactamente igual al original, incluyendo sus referencias.
  - Ejemplo:



Copia de Objetos (4)

- Copia en profundidad:
  - El objeto resultante es exactamente igual al original, salvo las referencias.
  - Ejemplo:



Copia de Objetos (5)

- Copia de Data Values:
  - Algunos Data Types deben ser implementados mediante clases, por lo cual sus instancias serán formalmente objetos.
  - Estos objetos se pueden copiar dado que
    - No tienen identidad (ya que son data values).
    - Se desea disponer de un ejemplar diferente en cada lugar donde se lo requiere.
  - La copia de data values se realiza **en profundidad**.

Copia de Objetos (6)

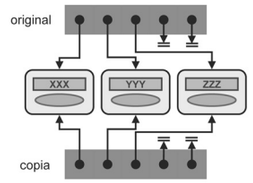
- Copia de Objetos:
  - Los objetos sí tienen identidad.
  - En caso de requerir a uno desde más de un lugar se debe compartirlo (no es aceptable copiarlo).
  - Como regla general NO se debe copiar objetos.
  - Existen casos controlados donde es posible realizar copias de objetos.

Copia de Objetos (7)

- Copia de Colecciones:
  - El caso de las colecciones es particular porque pueden involucrar:
    - Una estructura de datos (sin identidad).
    - Objetos (con identidad).
  - Las colecciones de data values se tratan como el caso de los data values (en profundidad).
  - En casos en que sea necesario otra colección igual a la original se debe copiar solamente la estructura de datos (plana).

Copia de Objetos (8)

- Copia de Colecciones (cont.)
  - La copia de colecciones de objetos se realiza en forma **plana**.
  - Ejemplo:



Destrucción de Objetos

- Los objetos alojados en el *heap* permanecen allí hasta que el programa termina (a diferencia de aquellos alojados en el *stack*).
- Cuando un objeto ya no es de utilidad se lo suele retirar del *heap* para liberar la memoria.
- Existen dos enfoques para ello:
  - Automático, mediante el llamado *Garbage Collector* (Java, C#).
  - Manual (C++).

Destrucción de Objetos (2)

- Garbage Collector:
  - Forma parte del ambiente de ejecución del lenguaje de programación.
  - Corre en paralelo con el programa:
    - Busca objetos en el *heap* tales que no exista ninguna referencia adjunta a ellos.
    - Cuando encuentra un objeto tal lo elimina y libera la memoria que éste ocupa.
  - Permite al programador solicitar memoria sin tener que preocuparse por "devolverla".

Destrucción de Objetos (3)

- Destrucción Manual:
  - Este enfoque es más complejo y delicado
  - Requiere que el programador explícitamente libere la memoria ocupada por un objeto
  - Problemas frecuentes:
    - Memoria inaccesible.
    - Referencias colgantes.

Destrucción de Objetos (4)

- Destrucción Manual (cont.)
  - **Memoria inaccesible**: esto ocurre cuando un objeto no tiene ninguna referencia adjunta a él.
  - Ejemplo:

```
{
   Empleado * e;
   e = new Jornalero();
}
```

La única referencia adjunta al jornalero recién creado se perdió cuando se llega a la llave de cierre. En consecuencia,el jornalero queda inaccesible.

Destrucción de Objetos (5)

- Destrucción Manual (cont.)
  - **Referencias colgantes**: esto ocurre cuando un objeto es compartido y se destruye a través de una de las referencias.
  - Ejemplo:

```
{
   Empleado * e1, *e2;
   e1 = new Jornalero();
   e2 = e1;
   delete e2;
}
```

Moraleja: no destruir objetos compartidos.

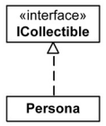
Destrucción de Objetos (6)

- Enfoques para la destrucción manual de objetos:
  - No destruir objetos: aplicable en sistemas donde no se cree una gran cantidad de objetos.
  - Utilizar contadores de referencias: cada objeto contiene un contador de referencias adjuntas a él.
  - Desarrollar una estrategia particular: en función de las particularidades del problema el programador "sabe" cuándo y cómo eliminar un objeto en forma segura.

Implementación Colecciones

Colecciones Genéricas
Genericidad de la Colección (2)

- Ejemplo:
  - La clase `Persona` debe realizar la interfaz de marca `ICollectible` para poder agregar personas a una colección genérica.



Recordar que una interfaz de marca no posee ninguna operación, por lo que no obliga a las clases que la implementan a presentar ningún servicio.

Introducción

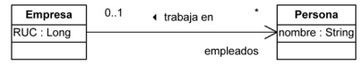
- La implementación de asociaciones usualmente requiere del uso de colecciones para permitir links con muchos objetos.
- El tipo de los elementos de las colecciones depende de la clase correspondiente al extremo de asociación navegable.
- Por tratarlas de manera uniforme éstas comparten una misma estructura que puede ser reutilizada para generarlas.

Introducción (2)

- Se distinguen dos tipos de colecciones,dependiendo de si los elementos contenidos poseen una clave que los identifique o no.
- La definición de las colecciones a utilizar en la implementación se estudiará incrementalmente.
- Se comenzará definiendo una **colección genérica** de elementos sin clave,la cual será aumentada para:
  - Permitir iteraciones sobre sus elementos.
  - Soportar el uso de claves.
  - Soportar diferentes tipos de búsquedas.
- Dichas definiciones serán luego utilizadas para implementar **colecciones concretas**.

Colecciones de Objetos

- Las colecciones de objetos son una herramienta fundamental para la implementación de muchas de las asociaciones presentes en un diseño.



El pseudoatributo **`empleados`** de la clase **`Empresa`** introduce la necesidad de una colección de personas.

Colecciones de Objetos (2)

- Las colecciones deben permitir:
  - Realizar iteraciones sobre sus elementos.
  - Realizar búsquedas de elementos por clave (en caso de que los elementos tengan una).
  - Realizar búsquedas diversas.
- Las colecciones concretas difieren en el tipo de elementos que contendrán pero coinciden en el tipo de servicios que brindan.

Colecciones de Objetos (3)

- Desarrollar cada colección en forma íntegra cada vez que se necesita resulta poco práctico.
- Es posible definir una única vez una infraestructura común que sirva de base para todas las colecciones específicas:
  - **Colecciones paramétricas** (templates): el tipo del elemento a almacenar es declarado como parámetro que será instanciado al generar la colección particular.
  - **Colecciones genéricas**: pueden almacenar directamente cualquier tipo de elemento.

Colecciones Genéricas

- Una colección genérica está definida de forma tal que pueda contener a cualquier tipo de elemento.
- Aspectos a considerar:
  - ¿Cómo lograr que un elemento de una clase cualquiera pueda ser almacenado en la colección genérica?
  - ¿Cómo se define la colección genérica?

Colecciones Genéricas
Genericidad de la Colección

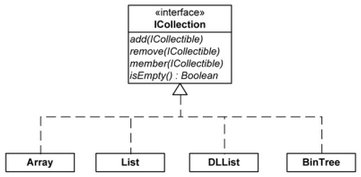
- ¿Cómo lograr que un elemento de una clase cualquiera pueda ser almacenado en la colección genérica?
- Se define la interfaz de marca `IColllectible`.
- Cuando se desea que los elementos de una cierta clase puedan ser almacenados en una colección genérica se solicita que dicha clase realice la interfaz `IColllectible`.
- De esta forma, la colección genérica contendrá elementos "coleccionables" (es decir, que implementarán la interfaz `IColllectible`).

Colecciones Genéricas
Encapsulamiento

- ¿Cómo se define una colección genérica?
- La noción de colección es independiente de su implementación.
- Se separa la especificación de la implementación:
  - Se define una interfaz `ICollection`.
  - Una cierta colección genérica será una implementación que realice esta interfaz.

Colecciones Genéricas
Realizaciones

- Podrán existir diferentes realizaciones (cada una con su estructura de datos particular) de la colección genérica:



Colecciones Concretas

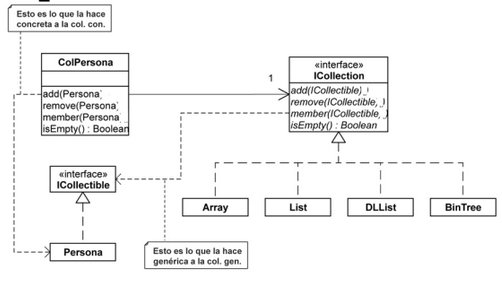
- ¿Cómo se define una colección concreta a partir de una colección genérica?
- Una colección concreta es aquella que:
  - Presta los mismos servicios que la genérica.
  - Puede definir nuevos servicios.
  - Fija el tipo de los objetos a coleccionar.
- El pseudoatributo **`empleados`** de la clase **`Empresa`** puede ser de tipo **`ColPersona`**
- Una instancia de clase **`ColPersona`:**
  - Encapsula a una colección genérica (que es quien contendrá efectivamente a las personas).
  - Puede definir operaciones que actúen sobre las personas.
  - Asegura que a dicha colección genérica le sea agregado elementos de clase **`Persona`** únicamente.

Colecciones Concretas (2)

- Una colección concreta se define como un "wrapper" de una colección genérica.
- Ejemplo:

Cada instancia de `ColPersona` utiliza una realización concreta de `IColllection` para almacenar (únicamente) instancias de `Persona`.

Enfoque Completo



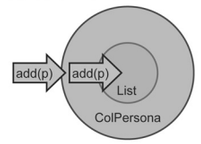
Enfoque Completo (2)

- Interacciones de una colección concreta:
  - Por ejemplo, lo diseñado de esta manera:

  - Podría, en la práctica, hacerse de esta otra:

Enfoque Completo (3)

- Interacciones de una colección concreta:
  - Visto de otra manera:

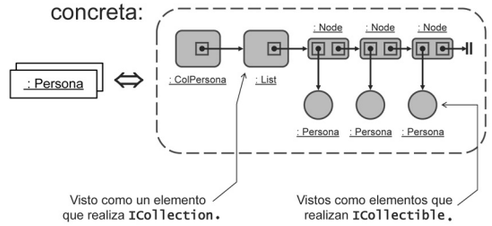


`ColPersona` "envuelve" al contenedor de objetos

Una empresa que tenga una instancia de `ColPersona` nunca utilizará el contenedor de elementos (en este caso `List`) directamente.

Enfoque Completo (4)

- Estructura de una colección concreta:

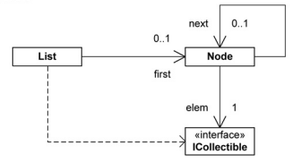


Realización de una Col. Genérica

- La interfaz `ICollection` declara qué servicios debe proveer una colección.
- Es posible realizar dicha interfaz de diferentes maneras mediante diferentes estructuras de datos.
- Realizaciones posibles:
  - Array o Vector.
  - Lista común o doblemente enlazada.
  - Arbol binario.
  - Etc.

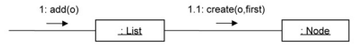
Realización de una Colección Genérica
Lista Común

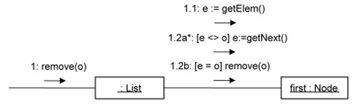
- El diseño de una lista común utilizando clases no difiere significativamente del diseño usual:



Realización de una Colección Genérica
Lista Común (2)

- Las operaciones no se resuelven en forma completa en la clase **`List`:**





Iteradores

- Es muy común necesitar realizar iteraciones sobre los elementos de una colección.
- La interfaz **`ICollection`** es aumentada con la siguiente operación:

`getIterator():IIterator`

- A su vez la colección concreta **`ColPersona`** es aumentada con la operación:

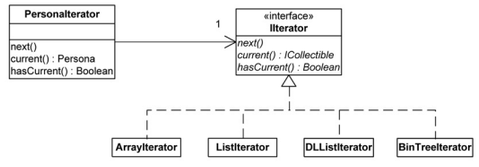
`getIterator():PersonaIterator`

Iteradores (2)

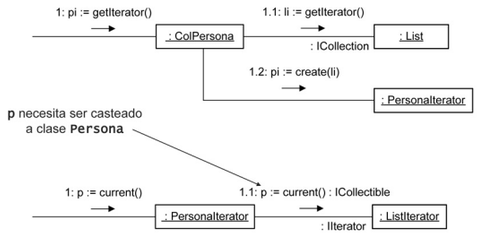
- Un iterador (sugerido en el patrón de diseño Iterator) es un objeto que permite recorrer uno a uno los elementos de una colección.
- Un iterador es un observador externo de la colección (no en el sentido del patrón Observer).
- Una colección puede tener diferentes iteradores realizando diferentes iteraciones simultáneamente.

Iteradores
Estructura

- Un iterador concreto, como **`PersonaIterator`**, encapsula a una realización de **`IIterator`**:



Iteradores
Interacciones



Iteradores
Uso de Iteradores

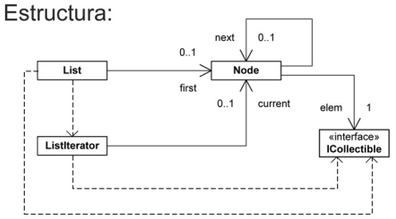
```
class Venta {
   private:
      ColLineasDeVenta lineas;
   public:
      float totalVenta();
};

float Venta::totalVenta() {
      float total = 0;
      LineaDeVentaIterator it = lineas.getIterator();

      while (it.hasCurrent()) {
         total = total + it.current()->subtotal();
         it.next();
      }
      return total;
}
```

Iteradores
Realización de Iteradores

- Como ejemplo de realización de iteradores se presenta el diseño de un iterador sobre listas comunes.
- La clase **`ListIterator:`**
  - Realiza la interfaz **`IIterator`.**
  - Es encapsulado por un iterador concreto (como **`PersonaIterator`**).

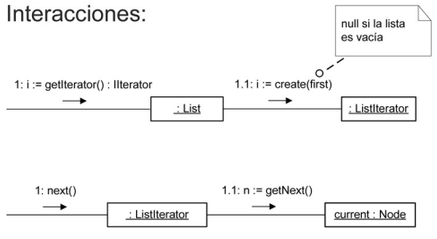
Iteradores
Realización de Iteradores (2)

- Estructura:



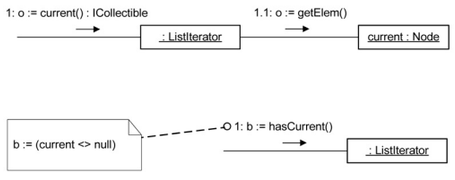
Iteradores
Realización de Iteradores (3)

- Interacciones:



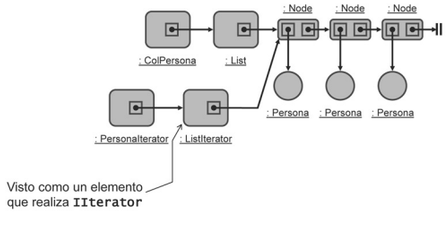
Iteradores
Realización de Iteradores (4)

- Interacciones (cont.):



Iteradores
Realización de Iteradores (5)

- Ejemplo:



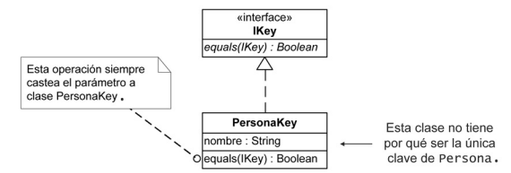
Diccionarios

- Un diccionario es un tipo particular de colección en el cual se almacenan objetos que pueden ser identificados por una clave.
- Se define la interfaz **`IDictionary`** y se utiliza en forma análoga a la interfaz **`ICollection:`**
  - Existirán diferentes realizaciones de **`IDictionary`.**
  - Las mismas contendrán elementos que realicen la interfaz **`ICollectible`.**
  - Un diccionario concreto como **`DictPersona`** encapsulará una realización de **`IDictionary`.**

Diccionarios
Uso de Claves

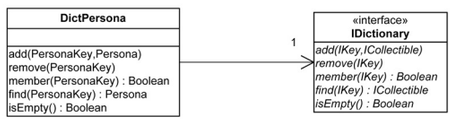
- Se está tratando la noción de diccionario genérico, por lo que la clave que identifica a los elementos debe ser también genérica.
- Se define la interfaz **`IKey:`**
  - Debe ser realizada por una clase que representa la clave de los elementos a incluir en el diccionario.
  - Contiene únicamente la operación **`equals(IKey):boolean`** utilizada para comparar claves concretas.

Diccionarios
Uso de Claves (2)



Los atributos de la clave concreta son una combinación de atributos de la clase.

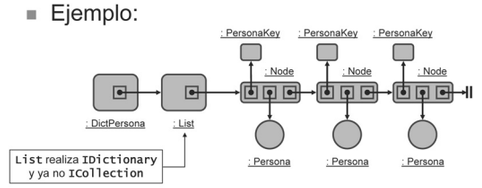
Diccionarios
Diccionarios Concretos



La clase **`DictPersona`** encapsula a una realización de **`IDictionary`** en forma análoga a como lo hace la clase **`ColPersona`** con la interfaz **`ICollection.`**

Diccionarios
Diccionarios Concretos (2)

- Ejemplo:



La relación entre una persona y su clave es particular a cada diccionario concreto. La clave que le corresponde a una persona es la referenciada por el nodo que referencia a la persona (determinada al momento del **`add()`**).

Diccionarios
Iteraciones en Diccionarios

- Un diccionario es una colección, por lo que tiene sentido necesitar iterar sobre sus elementos.
- Se incorpora a la interfaz **`IDictionary:`**
  - **`getElemIterator():IIterator`** que devuelve un iterador sobre los elementos contenidos en el diccionario.
  - **`getKeyIterator():IIterator`** que devuelve un iterador sobre las claves de los elementos contenidos en el diccionario.

Diccionarios
Iteraciones en Diccionarios (2)

- En diccionarios concretos (p.e. en el caso de **`DictPersona`**) las operaciones para realizar iteraciones son:
  - **`getPersonaIterator():PersonaIterator`** que devuelve un iterador sobre las personas del diccionario.
  - **`getKeyIterator():PersonaKeyIterator`** que devuelve un iterador sobre las claves (**`PersonaKey`**) de las personas contenidas en el diccionario.

Búsquedas

- Las búsquedas por clave no son el único tipo de búsqueda que se suele requerir.
- Existe otro tipo de búsquedas que no involucran necesariamente una clave:
  - Buscar todos los empleados menores de una cierta edad.
  - Buscar todos los empleados contratados antes de una fecha dada
- Este tipo de funcionalidad es análogo al que proporciona la operación **`select()`** de OCL.

Búsquedas (2)

- Dado que este tipo de búsquedas dependen de cada colección se implementan en las colecciones concretas.
- De esta forma, se define una operación por cada búsqueda necesaria:
  - Por ejemplo para buscar los empleados contratados antes de una fecha dada se incluye en **`ColPersona`**:

```
selectContratadosAntes(Fecha):ColPersona
```

Búsquedas (3)

```
ColPersona * ColPersona::selectContratadosAntes(Fecha f) {
   ColPersona * result = new ColPersona(new List());
   PersonaIterator it = getIterator();

   while(it.hasCurrent()) {
      if(it.current()->getFechaContratacion()<f)
         result->add(it.current());
      it.next();
   }
   return result;
}
```

Notar que todas las variantes de **`select()`** de **`ColPersona`** serán exactamente iguales entre sí a menos de esta porción del código.

Búsquedas (4)

- Las operaciones de búsqueda de una colección concreta son muy similares entre sí.
- Incluso no solamente las correspondientes a una misma colección concreta: las búsquedas en todas las colecciones son similares salvo:
  - La condición que determina la inclusión de un elemento en el resultado.
  - Los parámetros.
  - El tipo del iterador y la colección resultado.

Búsquedas (5)

- Sería posible incorporar a las interfaces **`ICollection`** e **`IDictionary`**, respectivamente,las operaciones:
  - **`select(ICondition) : ICollection`**
  - **`select(ICondition) : IDictionary`**
- La interfaz **`ICondition`** se define como:

en cada realización **`holds()`** indicará si un cierto objeto debe formar parte del resultado del **`select()`**.

Búsquedas (6)

- ¿Cómo manejar las diferencias mencionadas entre las diferentes implementaciones?
  - El tipo del iterador sería **`IIterator`**.
  - El tipo del resultado sería **`ICollection`** o **`IDictionary`** respectivamente.
  - A su vez,la condición encapsula:
    - El o los parámetros de la búsqueda (en sus atributos).
    - El algoritmo que determina si un elemento de la colección debe pertenecer,además,al resultado (en el método asociado a **`holds()`**).

Búsquedas (7)

- Una posible implementación de **`select()`** en una realización de **`ICollection`** sería:

```
ICollection * List::select(ICondition * cond) {
   ICollection * result = new List();
   IIterator * it = getIterator();

   while(it->hasCurrent()) {
      if(cond->holds(it->current()))
         result->add(it->current());
      it->next();
   }
   return result;
}
```

Búsquedas (8)

- De esta forma las clases que implementan **`ICondition`** son estrategias concretas que el **`select()`** utiliza para construir la colección resultado.
- En esta aplicación de Strategy se dan las siguientes correspondencias:
  - **`List`** → Context
  - **`ICondition`** → Estrategia
  - **`select()`** → solicitud()
  - **`holds()`** → algoritmo()

Búsquedas (9)

- Ejemplo de condición concreta:

```
class Persona : ICollectible {
private:
      String nombre;
      int edad;
public:
      Persona();
      String getNombre();
      int getEdad();
}
```

```
// CondEdad.h
class CondEdad : ICondition {
private:
      int valorEdad;
public:
      CondEdad(int i);
      bool holds(ICollectible *ic);
}
// CondEdad.cpp
CondEdad::CondEdad(int i) {
      valorEdad = i;
}
bool CondEdad::holds(ICollectible *ic) {
      Persona * p = (Persona *)ic;
      return (p->getEdad() == valorEdad);
}
```

4. Base de datos

A/S. Gabriella Savoia

Ejemplo 1

```
CREATE TABLE Empleados(
IdEmp       INT NOT NULL,
Apellido    VARCHAR(30) NOT NULL,
Nombre      VARCHAR(30) NOT NULL,
Direccion   VARCHAR(100) NOT NULL,
FecNac      DATETIME NOT NULL,
Salario     MONEY NOT NULL CONSTRAINT  check_salario CHECK (Salario >
0))
```

Ejemplo 2

```
ALTER TABLE  Empleados
ADD CONSTRAINT pk_empleado PRIMARY KEY (IdEmp)
```

Ejemplo 3

```
ALTER TABLE Empleados
DROP CONSTRAINT pk_empleado
```

Ejemplo 4

Dependiendo del DBMS, se pueden habilitar o deshabilitar (sin eliminarlas):

```
-- deshabilitar la restricción check_salario en la tabla.
  ALTER TABLE Empleados
  NOCHECK CONSTRAINT check_salario
-- habilitar la restricción check_salario en la tabla.
  ALTER TABLE Empleados
  CHECK CONSTRAINT check_salario
```

Ejemplo 5

"Todo libro se identifica por un ISBN. Todo libro es escrito por al menos UN autor."

```
CREATE TABLE Libros (
ISBN        INT NOT NULL PRIMARY KEY,
idAutor     INT NOT NULL,
Nombre      VARCHAR(100) NOT NULL,
Precio      MONEY  NOT NULL);

CREATE TABLE Autores (
idAutor      INT NOT NULL PRIMARY KEY,
Nombre       VARCHAR(100) NOT NULL);

ALTER TABLE Libros
ADD CONSTRAINT fk_autor FOREIGN KEY (idAutor )
REFERENCES Autores (idAutor )
ON DELETE CASCADE;
```

CREATE TABLE

Crea una nueva tabla.

Dependiendo del DBMS, exigirá crearla bajo un Esquema determinado.

Sintaxis:

```
CREATE TABLE     table_name
   ( { < column_definition >
     | < table_constraint > } [ ,...n ]
   )
< column_definition > ::= column_name data_type
[ DEFAULT constant_expression ]
[ < column_constraint > ] [ ...n ]
```

Definición de restricciones por COLUMNA

```
< column_constraint >::= [CONSTRAINT constraint_name]
{   [ NULL | NOT NULL ]
          | [{PRIMARY KEY | UNIQUE }]
| [[ FOREIGN KEY ]
REFERENCES ref_table [ ( ref_column ) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
]
| CHECK ( logical_expression )   }
```

Definición de restricciones a nivel de TABLA

```
< table_constraint > ::= [ CONSTRAINT constraint_name ]
{ [ { PRIMARY KEY | UNIQUE }
                    { ( column [ ASC | DESC ] [ ,...n ] ) } ]
| FOREIGN KEY  [ ( column [ ,...n ] ) ]
REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE| NO ACTION } ]   }
```

Ejemplo:

PostgreSQL:

```
CREATE TABLE "Empleados" (
    "CI" character(8) PRIMARY KEY ,
    "Nombre" character varying(100),
    "Direccion" character varying(200),
    "Fec_Nacimiento" date    )
WITH (  OIDS=FALSE);
```

MS-SQLServer:

```
CREATE TABLE Empleados (
    CI character(8) PRIMARY KEY ,
    Nombre  varchar(100),
    Direccion varchar(200),
    Fec_Nacimiento smalldatetime);
```

MySQL:

```
CREATE TABLE Empleados (
    CI character(8) PRIMARY KEY ,
    Nombre  varchar(100),
    Direccion varchar(200),
    Fec_Nacimiento date);
```

Ejemplos con constraints:

```
CREATE TABLE PuestosDeTrabajo(
Id_puesto   smallint        PRIMARY KEY,
Descripcion         varchar(50)     NOT NULL
    DEFAULT 'Nueva posición',
Nivel_Min           tinyint
    NOT NULL
    CHECK (min_lvl >= 10),
Nivel_Max           tinyint
    NOT NULL
    CHECK (max_lvl <= 250)
)
```

Estas restricciones son a nivel de COLUMNA

```
CREATE TABLE Empleados (
Id_emp  int  PRIMARY KEY,
Id_puesto   smallint   NOT NULL ,
id_seccion   smallint  NOT NULL

FOREIGN KEY (Id_puesto ) REFERENCES
PuestosDeTrabajo(Id_Puesto)
CHECK (id_seccion IN ('1389', '0736', '0877', '1622', '1756')
OR id_seccion LIKE '99[0-9][0-9]')
)
```

Contiene restricciones a nivel de COLUMNA y TABLA.

ALTER TABLE

Modifica la definición de una tabla. Permite:

- Alterar, agregar o eliminar: columnas o restricciones.
- Habilitar o deshabilitar constraints y triggers.

No puede aplicarse a tablas del sistema.

Sintaxis:

```
ALTER TABLE table
{ [ ALTER COLUMN column_name
      { new_data_type [ ( precision [ , scale ] )]  [ NULL | NOT NULL ] } ]
   | ADD    { [ < column_definition > ]
     |  column_name AS computed_column_expression  ej: cost AS price * qty} [ ,...n ]
     | [ WITH CHECK | WITH NOCHECK ]
           { < table_constraint > } [ ,...n ]

WITH CHECK :
```

Indica si la nueva constraint será chequeda o no sobre los datos ya existentes en la tabla (al momento de crear el constraint).

```
ALTER TABLE
   | DROP
           { [ CONSTRAINT ] constraint_name
           | COLUMN column } [ ,...n ]
     | { [ WITH CHECK | WITH NOCHECK ] CHECK |          NOCHECK }
CONSTRAINT
   | { ENABLE | DISABLE } TRIGGER
   }
```

Ejemplos

- Agregar una nueva columna:

```
CREATE TABLE doc_exa ( column_a INT);
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL ;
```

- Eliminar una columna:

```
CREATE TABLE doc_exb ( colA INT, colB  VARCHAR(20) NULL) ;
ALTER TABLE doc_exb DROP COLUMN colB;
```

- Agregar columna con restricción de integridad UNIQUE:

```
CREATE TABLE doc_exc ( column_a INT);
ALTER TABLE doc_exc ADD column_b VARCHAR(20) NULL
CONSTRAINT exb_unique UNIQUE;
```


Usos de ALTER

1 - Crear claves foraneas.

"La ciudad se identifica por un codigo, pero un codigo de ciudad puede repetirse en diferentes departamentos (entidad débil Ciudad)."

```
CREATE TABLE  DEPARTAMENTOS (
      IdDep int primary key,
      NomDep varchar(100) not null);
CREATE TABLE  CIUDADES  (
      IdDep  int,
      IdCiud int,
      NomCiud varchar(100) not null
      CONSTRAINT PK_CIUDADES PRIMARY KEY (IdDep,IdCiud)  );
ALTER TABLE CIUDADES  ADD  CONSTRAINT FK_DEPTOS
      FOREIGN KEY(IdDep)
      REFERENCES DEPARTAMENTOS (IdDep);
```

2 - Agregar una columna con valores por defecto.

WITH VALUES provee valores para las filas ya existentes en la tabla (sino cada fila quedaría con el valor NULL).

```
ALTER TABLE MyTable
ADD AddDate smalldatetime NULL
CONSTRAINT AddDateDflt
DEFAULT getdate() WITH VALUES ;
```

3 - Agregar constraints en tablas que ya lo violan.

Se agrega una constraint para una columna existente.

Si la columna actualmente posee un valor que viola la constraint, se puede usar WITH NOCHECK para evitar el chequeo contra las filas existentes y permitir agregar la restricción de todos modos.

```
CREATE TABLE  T1 ( column_a INT) ;
INSERT INTO      T1 VALUES (-1);
ALTER TABLE      T1 WITH NOCHECK
ADD CONSTRAINT exd_check CHECK (column_a > 1);
```

DROP TABLE

- Elimina:
  - La definición de la tabla:
  - Todos sus datos.
  - Objetos asociados: índices, triggers, constraints, especificaciones de permisos.
- No siempre pueden eliminarse: solo cuando no existen constraints de otras tablas hacia ella.
- Cualquier vista o stored procedure referenciado deben ser explícitamente eliminados antes con DROP VIEW o DROP PROCEDURE.
- No pueden eliminarse tablas del sistema.

Sintaxis:

```
DROP TABLE table_name
```

Comparativo: Restricciones estáticas vs. dinámicas

Componentes	Funcionalidad	Costo de Performance	Antes o Despues de la Transacción
Constraints	Media	Baja	Antes
Triggers /Stored Procedures	Alta	Alta	Despues

Consejos:

- Usar Constraints porque son ANSI-Compliant.
- CON CUIDADO: usar integridad referencial en cascada en lugar de Triggers (siempre que se pueda).
- Siempre generar SCRIPTS con los objetos de cada base, al mayor detalle posible (incluir indices, restricciones, etc.).

DML – SELECT

SELECT

- Sentencia única de consulta en bases de datos relacionales.
- Implementación de operaciones vistas en Algebra Relacional (selección, proyección, join, etc.).
- Permite obtener datos de varias tablas simultáneamente.
- Los resultados siempre serán conjuntos de tuplas: No necesariamente se devuelven en orden.
- La ejecución de esta sentencia NO MODIFICA dato alguno ni genera cambios en las base.
  Puede afectar el rendimiento general del DBMS si se hace descuidadamente.

Formato de la sentencia:

SELECT [ALL|DISTINCT] columnas deseadas
FROM tablas
[WHERE condición]
[GROUP BY lista-nombre-columna o lista-posición]
[HAVING condición de grupo]
[ORDER BY nombre-columna o posición]

Seleccionando todas las columnas

Ejemplo:
SELECT * FROM SECCIONES

IdSec	NomSec	IdSecSup
1	Directorio	1
2	Tecnologia	1
5	Marketing	4
4	Ventas	1
3	Finanzas	1

Seleccionando columnas específicas (proyección)

Ejemplo:
SELECT NomEmp, Direccion FROM EMPLEADOS

NomEmp	Direccion
Juan Perez	Sarando 619 apto. 101
Roberto Martínez	Tiburcio Gomez 1420
Sandra Perez	Of. 17 m. S/N
Mariana de Leon	Ruta 8 Km. 28 Paraje El Grillo
Andres Gomez	Cno. Del Andaluz Km. 8 ½

Seleccionando valores únicos:

Ejemplo:

```
SELECT cuenta
FROM Movimientos

CUENTA
10002
10002
10004
10002
10003
10004
```

```
SELECT DISTINCT cuenta
FROM Movimientos

CUENTA
10002
10004
10003
```

La cláusula WHERE:

Especifica un criterio de selección de registros a ver (selección).

SELECT lista_de_columnas
FROM nombre_de_tablas
WHERE condición

```
SELECT cliente, cuenta, producto
FROM   cuentas
WHERE producto = 1

CLIENTE  CUENTA     PRODUCTO

10002    100           1
10002    125           1
10004    789           1
```

```
SELECT cliente, cuenta, producto
FROM   cuentas
WHERE producto <> 1

CLIENTE  CUENTA     PRODUCTO

10015    110           6
10003    351           5
10004    454           2
```

Delimitadores

- En Strings o Fechas, suelen ser comillas dobles o apóstrofes.
- Se usan para delimitar los literales usados en el SELECT y evitar la confusión entre el nombre de una columna y su contenido:

Ejemplo:
FecNacimiento = '01/01/2001'
apellido = 'PEREZ'
CI >= "1000000-0"

```
SELECT cliente, nombre
FROM clientes
WHERE apellido = 'PEREZ'

CLIENTE        APELLIDO
10007          PEREZ
10008          PEREZ
10001          PEREZ
```

Operadores Relacionales

Significado	Simbolo	Ejemplo
Igual	=	Cuenta = 12003
No igual	<> , !=	Cuenta <> 12003
Mayor que	>	FecNac > '01/01/2001'
Menor que	<	FecNac < '01/01/2001'
Mayor o igual	>=	Saldo >= 12000
Menor o igual	<=	Saldo <= 12000
Pertenece a una lista	IN	Apellido IN ('PEREZ','MARTINEZ')
En un rango de valores	BETWEEN	Sueldo BETWEEN 12000 and 24000
Contiene un string	LIKE	Nombre LIKE '%JUAN%'

DML – Uso del NULL

```
SELECT persona, direccion
FROM personas
WHERE direccion IS NULL

PERSONA  DIRECCIÓN

10045
10063
10036
```

```
SELECT persona, direccion
FROM personas
WHERE direccion IS NOT NULL

PERSONA  DIRECCIÓN

10015    18 DE JULIO 2323
10034    18 DE JULIO 2325
10030    18 DE JULIO 2324
```

Dependiendo del programa de consulta puede mostrar NADA o la palabra *NULL*.

DML – Operadores Lógicos

Operador	Significado
AND	Devuelve TRUE (verdadero) cuando ambas condiciones son verdaderas.
OR	Devuelve TRUE (verdadero) cuando al menos UNA de las 2 condiciones es verdadera.
NOT	Devuelve la negación de la condición.

IMPORTANTE:
Los operadores poseen prioridad de asociación.
- El AND posee la más alta prioridad.
- Si necesitamos condiciones complejas con AND y OR debemos utilizar PARÉNTESIS.

1) Listar las personas que viven en "La Paloma" (en el departamento de Rocha).
SELECT persona, nombre
FROM personas
WHERE ciudad = "La Paloma"
AND departamento = "Rocha"

2) Listar las personas que viven en Rocha o Durazno.
SELECT persona, nombre
FROM personas
WHERE departamento = "Rocha"
OR departamento = "Durazno"

3) Ejemplo combinado de AND y OR.
¿Cuáles son los títulos de las películas del estudio "MGM" que fueron filmadas luego de 1970 o cuya duración es menor a 90 minutos?
Incorrecto :
SELECT NomPelicula
FROM Peliculas
WHERE anio > 1970 OR duracion < 90 AND NomEstudio = 'MGM'

Error: el AND tiene mayor precedencia, el compilador entiende
anio > 1970 OR (duracion < 90 AND NomEstudio = 'MGM')

Correcto:
SELECT NomPelicula
FROM Peliculas
WHERE (anio > 1970 OR duracion < 90) AND NomEstudio = 'MGM'

DML – Más búsquedas

Buscando en un rango de valores (BETWEEN).
Dos ejemplos equivalentes:

```
SELECT fecha,cuenta,importe
FROM movimientos
WHERE sucursal = 1  AND
      (importe >= 10000 AND
      importe <= 20000)
```

```
SELECT fecha,cuenta,importe
FROM movimientos
WHERE sucursal  = 1 AND
      importe BETWEEN 10000 AND 20000
```

Buscando en un conjunto de valores (IN).
Dos ejemplos equivalentes:

```
SELECT cliente,nombre
FROM clientes
WHERE cliente = 10052
      OR cliente = 10035
      OR cliente = 10028
      OR cliente = 10068
```

```
SELECT cliente,nombre
FROM clientes
WHERE cliente  IN  (10052,10035,10028,10068)
```

Uso del operador LIKE.

Búsquedas por caracteres o patrones.

Mascara	Significado
%	Equivale a cero o más caracteres cualesquiera.
_ (guión inferior)	Representa a UN caracter cualquiera.
[v1-v2]	Intervalo de valores posibles, una ocurrencia.
[^v1-v2]	Excluye el intervalo de valores, una ocurrencia

Búsquedas en Strings (char, varchar, char varying, etc.)
Ejemplo: Nombres que finalizan en Pérez.

```
SELECT cliente, nombre
FROM clientes
WHERE
nombre LIKE "%Perez"
```

CLIENTE	NOMBRE
10002	Juan Perez
10013	Pedro Perez
10016	Alberto Perez
10012	Francisco Perez

Búsquedas en Strings (char, varchar, char varying, etc.)
Nombres que terminan en Pere y el último carácter es cualquiera:

```
SELECT cliente,nombre
FROM clientes
WHERE nombre LIKE "%Pere_"
```

CLIENTE	NOMBRE
10002	Juan Perez
10013	Pedro Perez
10016	Alberto Perez
10012	Francisco Perez
10022	Luiz Peres

Otros ejemplos:

```
SELECT cliente,nombre
FROM clientes
WHERE Nombre
LIKE "[a-zA-Z]%Pere_"
```

CLIENTE	NOMBRE
10002	Juan Perez
10013	Pedro Perez
10016	Alberto Perez
10012	Francisco Perez
10044	Frank Peret
10022	Luiz Peres

```
SELECT cliente,nombre
FROM clientes
WHERE Nombre
LIKE "^[A-J]%Pere_"
```

CLIENTE	NOMBRE
10013	Pedro Perez
10022	Luiz Peres

La cláusula ORDER BY

SELECT no devuelve los registros en algún orden preestablecido.

- ORDER BY indica en qué orden quiero que muestre el resultado.
- Pueden ser varias columnas, en ese caso se respeta el orden de izquierda a derecha.
- ASC o DESC indican Ascendente o Descendente, ASC es el default.

Sintaxis:

```
SELECT    campos
FROM      tablas
[WHERE    condición]
...
ORDER BY nombre-columnas o posiciones [ASC | DESC]
```

Ej.:

```
SELECT     cliente, nombre
FROM       clientes
WHERE      nombre LIKE "%Perez"
ORDER BY   nombre ASC
```

CLIENTE	NOMBRE
10016	Alberto Perez
10034	Francisco Perez
10012	Francisco Perez
10002	Juan Perez
10022	Luiz Peres
10013	Pedro Perez

Operadores Aritméticos

Permiten formar expresiones complejas.

Utilidad:

- Devolver valores calculados (no incluidos en campos).
- Expresar condiciones (en WHERE o HAVING).
- Nuevos campos en Vistas.

Operadores:

- + suma.
- - resta.
- * multiplicación.
- / división.
- % módulo (resto).

Ejemplo 1

"Necesitaria ver la cotizacion de las monedas y cuánto sería si subieran todas un 5%."

```
SELECT    moneda, cotización, cotización * 1.05
FROM      cotizaciones
WHERE     moneda <> moneda_val
ORDER BY  cotizacion DESC

MONEDA    COTIZACION      (expression)
1         15,70           16.485
4         2,70            2.835
3         2,20            2.31
5         0,89            0.9345
```

Ejemplo 2

"Quiero todos los articulos cuyo precio de compra sea menor al 80% del precio de venta"

```
Select *
From ARTICULOS
Where  precio_compra < (precio_venta * 0.8).
```


Etiquetas

Los campos calculados devueltos en SELECT no poseen nombre: se les puede inventar un nombre "on-the-fly".

```
select moneda, cotización,          'nueva_cotizacion' = cotización * 1.05
from cotizaciones
where moneda <> moneda_val
ORDER BY nueva_cotizacion DESC
```

También pueden utilizarse para presentar otro nombre para el campo:

```
select "Codigo Articulo" = IdArt, "Nombre Articulo" = NomArt
from ARTICULOS
where ….
```

Joins

- Permite recuperar información de varias tablas vinculadas lógicamente entre si.
- Implementa la operación Join del Algebra Relacional.

Ej: "Quiero saber todos los datos de los Clientes más sus Nº de cuenta."
Tengo las tablas:
CLIENTES (nro_cliente, nom_cliente, direccion).
CUENTAS (nro_cliente,nro_cuenta, cod_moneda).

Consulta con Join :

```
SELECT Clientes.*, Cuentas.nro_cuenta
FROM   Clientes , Cuentas
    WHERE Clientes.nro_cliente = Cuentas.nro_cliente
```

Listamos todos los datos de Clientes + Nº de Cuenta

Incluimos 2 tablas en el FROM

Obligatorio : condición de Join

Joins: ¿Qué son?

- Es la implementación del Producto Cartesiano (T1 x T2) + Selección.
- Si no se especifica una condición, el conjunto resultante no posee sentido práctico.

Clientes

Nro_cliente	Nom_cliente	Dirección
1000	Ana	Dir1
1100	Pedro	Dir2
1200	Maria	Dir3
1300	Roberto	Dir4

Cuentas

Nro_cliente	Nro_cuenta	Cod_moneda
1000	3521	1
1200	3687	0

```
Select *
From Clientes, Cuentas
```

Solo tienen sentido los que coinciden en **nro_cliente**

Nro_cliente	Nom_cliente	Dirección	Nro_cliente	Nro_cuenta	Cod_moneda
1000	Ana	Dir1	1000	3521	1
1000	Ana	Dir1	1200	3687	0
1100	Pedro	Dir2	1000	3521	1
1100	Pedro	Dir2	1200	3687	0
1200	Maria	Dir3	1000	3521	1
1200	Maria	Dir3	1200	3687	0
1300	Roberto	Dir4	1000	3521	1
1300	Roberto	Dir4	1200	3687	0

Aplicando la condición de Join:

```
Select *
From Clientes, Cuentas
Where Clientes.nro_cliente = Cuentas.nro_cliente
```

Nro_cliente	Nom_cliente	Dirección	Nro_cliente	Nro_cuenta	Cod_moneda
1000	Ana	Dir1	1000	3521	1
1200	Maria	Dir3	1200	3687	0

El campo Nro_cliente aparece dos veces: uno por cada tabla donde aparece.
Solución:

1. Exponer en el SELECT solo los campos que queremos ver.
2. Utilizar ALIAS.

Joins: Alias

- Son un modo de "renombrar" las tablas para mayor comodidad.
- Permite hacer más legible joins de varias tablas.

Ejemplo: "Listado de todos los Clientes con su Nº de cuenta y moneda."

```
SELECT CLI.nom_cliente, CU.nro_cuenta, M.nom_moneda
FROM   Clientes CLI, Cuentas CU, Monedas M
WHERE
       CLI.nro_cliente = CU.nro_cliente
AND  CU.cod_moneda = M.cod_moneda
```

Sintaxis ANSI del Join

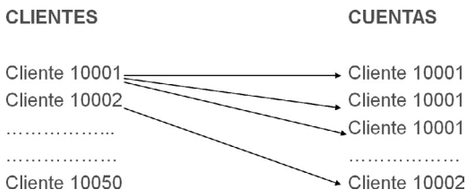
```
SELECT    CU.*,  CLI.nro_cliente, CLI.nom_cliente
FROM                    Cuentas
AS CU  JOIN Clientes AS CLI
ON CU.nro_cliente =  CU.nro_cliente
WHERE CLI.nom_cliente like '%PEREZ%' ;
```

Equivalente más compacto:

```
SELECT    CU.*,  CLI.nro_cliente, CLI.nom_cliente
FROM                    Clientes CLI  , Cuentas CU
      WHERE CU.nro_cliente =  CU.nro_cliente
            and CLI.nom_cliente like '%PEREZ%';
```

Creando un JOIN:

Usualmente se desea recuperar información de más de una tabla. Por ejemplo:



- Creación de JOINs.

1. Creación del Producto Cartesiano.
2. Refinamiento aplicando restricciones y eliminando filas sin significado relevante incluyendo una claúsula WHERE válida.

- Tipos:

3. Equi-Join, Natural-Join and Join Multi-Tabla.
4. Outer-Joins.

- Información adicional en las cláusulas de la sentencia Select:

SELECT Indicar qué columnas se quiere seleccionar de cada una de las tablas.
FROM Especificar las tablas de las que se esta seleccionando informacion en la **SELECT**.
WHERE Indicar las columnas de las tablas seleccionadas que se igualarán para establecer el **join**.

- Consideraciones:

Clave Primaria (Primary Key)

Se define como el conjunto de uno o más campos de un registro que conforman su clave, determinando la unicidad de cada fila en la tabla.

Clave Externa (Foreign Key)

Asocia los campos de una tabla con un conjunto idéntico de campos, definidos como Clave Primaria en otra tabla. Esta asociación permite el chequeo de integridad referencial y actualizaciones automáticas.

- Algunas particularidades:

Primary Key – Foreign Key

Es muy común realizar joins entre tablas que se encuentran en una Relación de uno a muchos.

Las columnas que se igualarán para establecer el join no tienen porque tener el mismo nombre.

NOTA: Recordemos que el valor null significa sin valor o desconocido. A traves de él no se puede hacer un join.

El orden en el que se escriben las condiciones del join no afecta el significado del mismo.

Equi-Join:

Theta-Join basado en condición de igualdad:

R(A,B)

A	B
3	4
5	7

S(C,D)

C	D
2	7
6	8

$R \times S$

R.A	R.B	S.C	S.D
3	4	2	7
3	4	6	8
5	7	2	7
5	7	6	8

$R \bowtie_{R.B=S.D} S$

Resultado

R.A	R.B	S.C	S.D
5	7	2	7

Equi-Join: Join en el cual la condición de selección está basada en la **igualdad de valores entre columnas**, las que pueden aparecer como información redundante en el resultado.

Obs: los nombres de las columnas en las diferentes tablas no necesariamente debe ser el mismo.

```
SELECT * FROM Clientes, Cuentas
WHERE Clientes.cliente = Cuentas.cliente
```

¿Recupera?

Resultado con duplicado de información:

```
cliente nombre       cliente     cuenta moneda     saldo

10007 CUENTA 10007   10007       100071      1   51112.31
10007 CUENTA 10007   10007       100072      1  -31484.56
10009 CUENTA 10009   10009       100092      1   -5468.72
10010 CUENTA 10010   100010      100101      1 -425920.75
10010 CUENTA 10010   100010      100102      1       0.00
10011 CUENTA 10011   100011      100112      1       0.00
```

SELECT **clientes.cliente**, clientes.nombre,
cuenta, moneda, saldo
FROM **clientes, cuentas**
WHERE **clientes.cliente = cuentas.cliente;**
¿Problema?

Si el cliente No tiene cuenta, no figura en el resultado.

```
cliente  nombre                cuenta moneda           saldo

  10007 CUENTA 10007           100071      1   51112.31000000
  10007 CUENTA 10007           100072      1  -31484.56000000
  10009 CUENTA 10009           100092      1   -5468.72000000
  10010 CUENTA 10010           100101      1 -425920.75000000
  10010 CUENTA 10010           100102      1       0.00000000
  10011 CUENTA 10011           100112      1       0.00000000
```

Un error común:
SELECT **cliente**, clientes.nombre, cuenta,
moneda, saldo
FROM **clientes, cuentas**
WHERE **clientes.cliente = cuentas.cliente;**
Columna ambigua, existe en ambas tablas.

```
324: Ambiguous column (cliente).
```

Natural Join

Emp (name, dept)

Name	Dept
Jack	Physics
Tom	ICS

Contact (name, addr)

Name	Addr
Jack	Irvine
Tom	LA
Mary	Riverside

Emp ⋈ **Contact**: Todos los nombre, departamentos y direcciones

Emp × Contact

Emp.name	Emp.Dept	Contact.name	Contact.addr
Jack	Physics	Jack	Irvine
Jack	Physics	Tom	LA
Jack	Physics	Mary	Riverside
Tom	ICS	Jack	Irvine
Tom	ICS	Tom	LA
Tom	ICS	Mary	Riverside

Resultado

Name	Dept	Addr
Jack	Physics	Irvine
Tom	ICS	LA

Natural Join es un Equi-Join en el cual una de las columnas duplicadas es eliminada de la tabla resultante, usualmente utilizadas en la condición de Join.

SELECT monedas.*, fecha, cotizacion
FROM monedas, cotizaciones
WHERE monedas.moneda = cotizaciones.moneda;

Resultado: se evita la información redundante.

```
moneda nombre                moneda_val fecha       cotizac
     1 PESO URUGUAYO                  1 01/03/2002     1.00
     2 DOLAR AMERICANO                1 01/03/2002    15.60
     3 PESO ARGENTINO                 1 01/03/2002     8.00
     4 REAL                           1 01/03/2002     6.70
     5 EURO                           1 01/03/2002    14.00
     1 PESO URUGUAYO                  1 04/03/2002     1.00
     2 DOLAR AMERICANO                1 04/03/2002    15.75
     3 PESO ARGENTINO                 1 04/03/2002     8.00
     4 REAL                           1 04/03/2002     6.70
     5 EURO                           1 04/03/2002    14.00
```

Join con muchas tablas:

```
SELECT clientes.nombre,
       productos.nombre,
       monedas.nombre
FROM   clientes, cuentas, productos, monedas
WHERE cuentas.cliente    = clientes.cliente    AND
      cuentas.producto = productos.producto AND
      cuentas.moneda  = monedas.moneda
```

Resultado:

```
nombre                nombre                nombre

CLIENTE 10010         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10101         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10397         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10080         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10131         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10404         CAJA DE AHORRO        DOLAR AMERICANO
CLIENTE 10453         CAJA DE AHORRO        PESO URUGUAYO
```

Los ALIAS:

```
SELECT   CE.nombre cliente,
         P.nombre producto,
         M.nombre moneda
FROM     clientes CE, cuentas CU,
         productos P, monedas M
WHERE    CU.cliente    = CE.cliente
  AND    CU.producto = P.producto
  AND    CU.moneda   = M.moneda
```

Resultado:

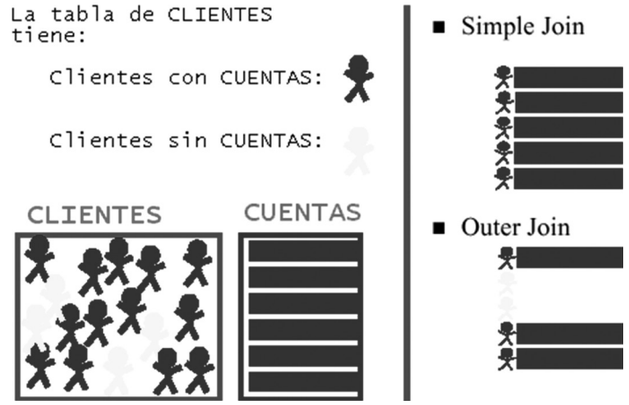
```
cliente               producto              moneda

CLIENTE 10010         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10101         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10397         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10080         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10131         CAJA DE AHORRO        PESO URUGUAYO
CLIENTE 10404         CAJA DE AHORRO        DOLAR AMERICANO
CLIENTE 10453         CAJA DE AHORRO        PESO URUGUAYO
```

Sintaxis ANSI del Join:

SELECT **M.***, fecha, cotización
FROM monedas **AS M JOIN** cotizaciones **AS C**
ON M.moneda = **C.**moneda
WHERE fecha < "01/05/2002" ;

```
moneda nombre              moneda_val fecha          cotizacion
     1 PESO URUGUAYO               1 01/03/2002      1.00000000
     2 DOLAR AMERICANO             1 01/03/2002     15.60000000
     3 PESO ARGENTINO              1 01/03/2002      8.00000000
     4 REAL                        1 01/03/2002      6.70000000
     5 EURO                        1 01/03/2002     14.00000000
     1 PESO URUGUAYO               1 04/03/2002      1.00000000
     2 DOLAR AMERICANO             1 04/03/2002     15.75000000
     3 PESO ARGENTINO              1 04/03/2002      8.00000000
     4 REAL                        1 04/03/2002      6.70000000
     5 EURO                        1 04/03/2002     14.00000000
```

El OUTER Join:



El Join común (INNER Join) trae solamente los registros de ambas tablas que cumplan con las condiciones del JOIN. Por ejemplo, cuando recuperamos los clientes con sus cuentas NO trae los clientes sin cuentas.

Es por ello que existe el OUTER Join que trae todos los registros de la tabla principal. Si no existen registros de la otra que cumplan la condición de Join pone sus campos en NULO (Null) y en caso contrario los trae.

Existen tres tipos: left, right, o full, según cual se considere la tabla "dominante".

Left Outer Join

R

Name	Dept
Jack	Physics
Tom	ICS

Name	Addr
Jack	Irvine
Mike	LA
Mary	Riverside

S

$R \times S$

Left outer join
$R \bowtie S$

R.name	R.Dept	S.name	S.addr
Jack	Physics	Jack	Irvine
Jack	Physics	Mike	LA
Jack	Physics	Mary	Riverside
Tom	ICS	Jack	Irvine
Tom	ICS	Mike	LA
Tom	ICS	Mary	Riverside

Name	Dept	Addr
Jack	Physics	Irvine
Tom	ICS	NULL

Rellena con nulos para tuplas sin correspondiente a la derecha

Ejemplo de Left OUTER Join:
SELECT C.cliente, C.nombre,U.cuenta
FROM clientes C LEFTOUTER JOIN
cuentas U ON (c.cliente = u.cliente)

Resultado Outer Join Simple:

```
cliente nombre           cuenta
   10005 CLIENTE 10005
   10006 CLIENTE 10006
   10007 CLIENTE 10007    100071
   10007 CLIENTE 10007    100072
   10008 CLIENTE 10008
   10009 CLIENTE 10009    100092
   10010 CLIENTE 10010    100101
   10010 CLIENTE 10010    100102
   10011 CLIENTE 10011    100112
   10011 CLIENTE 10011    100111
   10012 CLIENTE 10012    100121
   10012 CLIENTE 10012    100122
   10013 CLIENTE 10013
   10014 CLIENTE 10014
```

Right Outer Join

R

Name	Dept
Jack	Physics
Tom	ICS

Name	Addr
Jack	Irvine
Mike	LA
Mary	Riverside

S

$R \times S$

Right outer join
$R \bowtie S$

R.name	R.Dept	S.name	S.addr
Jack	Physics	Jack	Irvine
Jack	Physics	Mike	LA
Jack	Physics	Mary	Riverside
Tom	ICS	Jack	Irvine
Tom	ICS	Mike	LA
Tom	ICS	Mary	Riverside

Name	Dept	Addr
Jack	Physics	Irvine
Mike	NULL	LA
Mary	NULL	Riverside

Rellena con nulos para tuplas sin correspondiente a la izquierda

Full Outer Join

R

Name	Dept
Jack	Physics
Tom	ICS

Name	Addr
Jack	Irvine
Mike	LA
Mary	Riverside

S

R × S

Full outer join
R ⟗ S

R.name	R.Dept	S.name	S.addr
Jack	Physics	Jack	Irvine
Jack	Physics	Mike	LA
Jack	Physics	Mary	Riverside
Tom	ICS	Jack	Irvine
Tom	ICS	Mike	LA
Tom	ICS	Mary	Riverside

Name	Dept	Addr
Jack	Physics	Irvine
Tom	ICS	NULL
Mike	NULL	LA
Mary	NULL	Riverside

Rellena con blancos para las tuplas sin correspondiente tanto a la derecha como a la izquierda

Nested Simple Join:

```
SELECT C.cliente, C.nombre, U.cuenta, P.nombre
FROM    clientes C LEFT OUTER JOIN
(cuentas U JOIN productos P ON U.producto = P.producto)
            ON (C.cliente = U.cliente)
```

1. Realiza un join simple entre las tablas cuentas y productos.
2. Luego realiza un outer join para combinar la información con la tabla clientes.

Resultado Nested Simple Join:

```
cliente nombre                  cuenta nombre

   10001 CLIENTE 10001
   10002 CLIENTE 10002
   10003 CLIENTE 10003
   10004 CLIENTE 10004
   10005 CLIENTE 10005
   10006 CLIENTE 10006
   10007 CLIENTE 10007          100071 CAJA DE AHORRO
   10007 CLIENTE 10007          100072 CUENTA CORRIENTE
   10008 CLIENTE 10008
   10009 CLIENTE 10009          100092 CUENTA CORRIENTE
```

Self Join

Utilidad: comparación de valores en una columna con otros valores en la misma columna.

```
SELECT   X.cod_orden, X.peso, X.fecha_envio,
         Y.cod_orden, Y.peso, Y.fecha_envio
FROM     Ordenes X, Ordenes Y
WHERE    X.peso >= 5*Y.peso AND
         X.fecha_envio IS NOT NULL AND
         Y.fecha_envio IS NOT NULL
```


Resultado: Este SELECT encuentra pares de órdenes cuyo peso difiere en, por lo menos, un factor de 5 y cuyas fechas de envío no son nulas.

cod_orden	peso	fecha_envio	cod_orden	peso	fecha_envio
1004	95.8	05/03/1991	1011	10.4	07/03/1991
1004	95.8	05/03/1991	1020	14.0	07/16/1991
1007	125.9	06/03/1991	1015	20.6	07/30/1991
1007	125.9	06/03/1991	1020	14.0	07/16/1991
1007	125.9	06/03/1991	1022	15.0	07/16/1991

Funciones de Agregación

Toman valores que dependen de las columnas y retornan información respecto a las columnas (no las columnas propiamente):

- COUNT (*)
- COUNT (DISTINCT nombre_columna)
- SUM (columna/expresión)
- AVG (columna/expresión)
- MAX (columna/expresión)
- MIN (columna/expresión)

COUNT

Devuelve la cantidad de tuplas que cumplen la condición de WHERE o HAVING.

Ejemplo 1: ¿Cuántos movimientos se han hecho en el banco?

```
SELECT COUNT (*)  FROM Movimientos;
```

Ejemplo 2: ¿Cuántos alumnos hay Inscriptos a una materia?

```
SELECT COUNT (DISTINCT cod_alumno)
FROM    Inscripciones;
```

SUM

Suma los contenidos de un campo numerico.

Ej: ¿Cuánto dinero se ha depositado en la cuenta 100101?

```
SELECT SUM(importe)
FROM    Movimientos
WHERE importe>0 AND cuenta=100101;
```

(sum)
887786.31000000

Se les puede aplicar Etitquetas:

```
SELECT SUM(importe) AS 'Total Depositos'
FROM    Movimientos
WHERE importe>0 AND cuenta=100101;
```

Total Depositos
887786.31000000

AVG

Devuelve el promedio de los contenidos de un campo numerico.

AVG = SUM() / COUNT()

¿Cuál es el monto promedio depositado en la cuenta 100101?

```
SELECT AVG(importe)  as Promedio
FROM    Movimientos
WHERE importe>0 AND cuenta=100101;
```

Promedio
569847.320

MAX, MIN

Devuelven el mayor o menor valor del conjunto seleccionado.

Ej. 1: ¿Cuál fue el depósito más alto en la cuenta 100101?

```
SELECT MAX(importe)
FROM   Movimientos
WHERE cuenta = 100101;
```

(max)
661306.92000000

Ej. 2: ¿Cuál fue el depósito más pequeño en la cuenta 100101?

```
SELECT MIN(importe)
FROM   Movimientos
WHERE cuenta=100101 AND importe>0;
```

(min)
226479.39000000

Las funciones de Agregación se pueden aplicar simultáneamente.
En este caso se aplican al MISMO conjunto de tuplas.

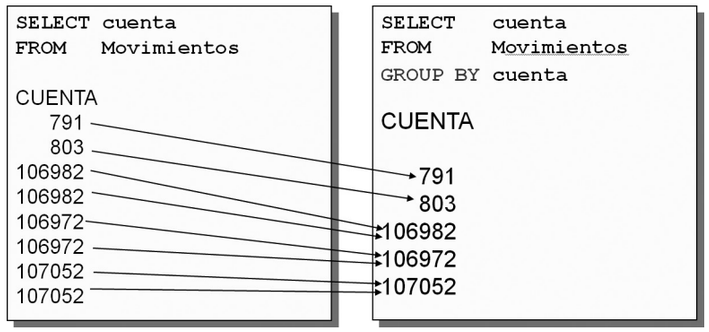
```
SELECT MAX(importe), MIN(importe), AVG(importe)
FROM   Movimientos
WHERE cuenta = 100101 and importe > 0;
```

En este ejemplo devuelve una tupla con tres campos.

Agrupamientos

GROUP BY

- Permite agrupar los registros por un campo (o más de uno).
- Produce un solo registro por cada grupo de registros.



Su utilidad es combinarlo con las Funciones Agregadas:
Ejemplo 1: "Quiero saber cantidad de movimientos y el importe por Cuenta."

```
SELECT cuenta, count(*), sum (importe)
FROM   Movimientos
GROUP BY cuenta
```

CUENTA	(count(*))	(sum)
791	1	167505.18
803	1	139000.00
106982	2	0.00
106972	2	0.00
107052	2	0.00

Ejemplo 2: "Quiero saber cantidad de personas por Departamento y luego por Ciudad."

```
SELECT departamento, ciudad, COUNT(*)
FROM   clientes
GROUP BY departamento, ciudad.
```

departamento	ciudad	(count(*))
MONTEVIDEO	MONTEVIDEO	598
CANELONES	CANELONES	22
CANELONES	ATLANTIDA	13
MALDONADO	MALDONADO	37
MALDONADO	P. DEL ESTE	25

Nota: No necesariamente se ordenan por ese criterio.

Ordenando el GROUP BY:

Ejemplo 1: Ordenado por departamento y luego ciudad.

```
SELECT      departamento, ciudad, COUNT(*)
FROM        Clientes
GROUP BY departamento, ciudad
ORDER BY departamento, ciudad
```

Ejemplo 2: Ordenado por Cantidad (descendente) y luego por ciudad y departamento.

```
SELECT      departamento, ciudad, COUNT(*)
FROM        Clientes
GROUP BY departamento, ciudad
ORDER BY 3 DESC, 2, 1
```

IMPORTANTE:

Todas las columnas en la lista del SELECT que no estén en funciones de agregación, deben figurar en los campos de GROUP BY.

Esto es porque el GROUP BY solo puede retornar una fila por grupo y para esas filas se aplica la función de agregación.

```
SELECT col1, col2,col3,...., colN, funcA(), funcB()
FROM   .....
WHERE ....
GROUP BY col1, col2,col3,...., colN
```

HAVING

- La claúsula HAVING usualmente complementa a GROUP BY aplicando condiciones a los grupos (especificados por el GROUP BY) luego de que éstos están formados.
- Ventajas: se pueden incluir funciones de agregación como condición de búsqueda, facilidad que no está permitida en:

WHERE.

Ejemplo:
"Quiero saber el total de dinero (por cuenta) de las cuentas > 10.000, pero solo de los que tengan un total positivo."

```
SELECT    cuenta,  SUM(importe) AS Total
FROM      Movimientos
WHERE     cuenta > 10000
GROUP BY cuenta
HAVING   SUM(importe) > 0;
```

HAVING filtra grupos así como el WHERE filtra registros.

cuenta	total
100071	51112.31000000
100102	100.0000000
100111	226857.2900000

Ejemplo de agrupamientos:

```
SELECT      cuenta,
            MAX(importe) Maximo,
            MIN (importe) Minimo,
            AVG(importe) Promedio
FROM        Movimientos
WHERE       cuenta > 10000
GROUP BY cuenta
HAVING    COUNT(*) > 2  AND
            SUM(importe) > 0
```

Funciones Escalares

- Funciones de String.
- Funciones Aritméticas.
- Funciones de Fecha.
- Funciones del Sistema.

Se pueden componer, siempre que se respeten los dominios de Entrada y Salida.

Funciones de String

- LEN
- STR
- SUBSTRING
- LOWER
- UPPER
- LTRIM
- CHARINDEX
- PATINDEX
- SPACE
- CHAR
- REPLICATE
- REVERSE
- STUFF
- DIFFERENCE
- RIGHT

LEN (campo/valor)
Devuelve el largo del string pasado como argumento.
len('HOLA') Resultado : 4
len('') Resultado : 0

STR (valor_numerico[, largo[, pos_decimales]])
SELECT str(-165.8768, 7, 2)
Resultado: '-165.88'

SUBSTRING (campo/valor, posicion inicial, largo).
Devuelve un fragmento del String (parametro 1).
Los caracteres comienzan en la posición 1.
SELECT substring("ROBERTO MARTINEZ DELGADO",8,7).
Resultado: ' MARTIN'.

LOWER (<char_expr>) Devuelve el mismo string pasado a minúsculas.

UPPER (<char_expr>) Devuelve el mismo string pasado a mayúsculas.

```
SELECT upper('Bob Smith1234*&^'), lower('Bob Smith1234*&^')
----------------                  ----------------
BOB SMITH1234*&^        bob smith1234*&^
```

LTRIM (<char_expr>) Remueve espacios en blanco a la izq.

```
SELECT   ltrim('     valor ')
--------------
valor
```

CHARINDEX retorna la posición de comienzo de una determinada cadena en una expresión, donde expresión usualmente es el nombre de una columna.

CHARINDEX (<'char_expr'>, <expression>)

SELECT charindex ('de', 'Un pequeño texto de muestra')

Resultado: 18

Funciones Aritméticas

Función	Parámetros	Semántica
• ABS	(N)	Devuelve el valor absoluto de N
• SIGN	(N)	Devuelve -1 si N<0 , 1 si N>0 o 0
• CEILING	(N)	Entero inmediato siguiente a N
• FLOOR	(N)	Entero inmediato anterior a N
• EXP	(N)	EXP(N) = e^N
• LOG	(N)	LOG(N) = $Log_e(N)$
• POWER	(x,y)	POWER(x,y) = x^y
• ROUND	(N, d)	Redondea N a d dígitos
• SQRT	(N)	Raíz cuadrada de N
• Trigonométricas		

Funciones de Fecha

DATEPART (<date_part>, d)

Devuelve un componente de la fecha d: year, month, day, hour, minute, second
select datepart(day, getdate()) ➔ 14
select datepart(year, '25/07/2009') ➔ 2009

DATENAME (<date part>, d)

Devuelve el nombre de una parte de la fecha d:
select datename(month,'25/07/2009') ➔ 'July'
select datename(weekday,'25/07/2009') ➔ 'Saturday'

DATEADD (<date part>, <number>, <date>)

Suma o resta intervalos a una fecha (dias, meses, años, etc.):
select dateadd(day, 10, '25/07/2009') ➔ '4/8/2009'
select dateadd(month, 2, '25/07/2009') ➔ '25/9/2009'
select dateadd(month, -9, '25/07/2009') ➔ '25/10/2008'

DATEDIFF(<date part>, <date1>, <date2>)

Calcula diferencia entre 2 fechas (en dias , meses, años, etc):
select datediff(day, '25/07/2009', '25/08/2009') ➔ 31
select datediff(month, '25/07/2009', '25/08/2009') ➔ 1

Funciones del Sistema

Permiten obtener información del entorno.
Devuelven información del sistema, usuario, BD y objetos de la BD.
Suelen depender del DBMS.

- getdate(), CURRENT_DATE(), CURRENT_TIME()
  Devuelven fecha/hora actual.
- host_name()
  Nombre del equipo desde donde se conectó.
- db_name(), DATABASE()
  Nombre de la base en que estamos posicionados.
- user_name(), CURRENT_USER()
  Devuelve el nombre del usuario del DBMS actualmente conectado.
- @@VERSION, VERSION()
  Devuelven la version del DBMS

Subqueries

- Son sentencias SELECT anidadas dentro de otra sentencia SELECT.
- Devuelven información a la principal y deben figurar siempre entre paréntesis.
- Permiten implementar la operación DIFERENCIA del Algebra Relacional.

Ejemplo: "Necesito listar personas que viven en la misma ciudad que 'CLIENTE 10010'"

```
SELECT persona, nombre, ciudad, departamento
FROM Personas
WHERE ciudad = (SELECT ciudad
                FROM Personas
                WHERE nombre = 'CLIENTE 10010')
```

Las subqueries son evaluadas primero y su(s) valor(es) son sustituido(s) en la consulta principal.

Una subquery puede retornar:

- Ningun valor.
  Consecuencias: Dicha subquery es equivalente a un valor Nulo.
  La query general no retona ningún valor.
- Un valor.
  Consecuencia: La subquery es equivalente a un número o valor carácter.
- Un conjunto de valores.
  Consecuencia: La subquery retorna o una fila o una columna.

Restricciones:

- Solo si el subquery devuelve UN valor puede preguntarse por =.
- Si el subquery devuelve UN campo se puede preguntar por IN.
- En caso contrario se debe preguntar por EXISTS.

Ejemplo 1: Listar las personas que viven en la misma ciudad que ' JUAN PEREZ'.

```
SELECT  persona, nombre, ciudad, departamento
FROM  Personas
WHERE  ciudad = (select  ciudad
                 from  personas
                 where  nombre = 'JUAN PEREZ' )
```

Podemos usar = porque sabemos que solo se va a devolver **1 tupla**.

Ejemplo 2: Listar personas que viven en la misma ciudad y departamento que 'JUAN PEREZ'.

```
SELECT persona, nombre, ciudad, departamento
FROM Personas P
WHERE EXISTS (select   *
                from   personas P2
                where  P2.nombre = 'JUAN PEREZ'
                   and  P.ciudad = P2.ciudad
                   and  P.departamento = P2.departamento)
```

Usamos EXISTS porque no podemos preguntar si <Ciudad,Departamento> IN .....

Tipo de Subqueries

- Correlacionadas
- No-Correlacionadas

Correlacionadas (o inner SELECT): el valor producido por ella depende de un valor producido por el SELECT externo. En cualquier otro caso son **No-Correlacionadas.**

Subqueries Correlacionados

Listar los Empleados cuyo sueldo está por debajo del promedio de su Sección:

```
SELECT nro_emp, nom_emp, seccion, sueldo
FROM    Empleados E1
WHERE  sueldo < (SELECT AVG (sueldo)
                  FROM   Empleados E2
                  WHERE E2.seccion =  E1.seccion)
ORDER BY 1, 2, 3
```

Deben referirse a la MISMA sección

La subquery es ejecutada por cada fila considerada por el SELECT externo.

Negación

Permiten implementar DIFERENCIA de tablas.

Ej: Qué clientes no tienen cuentas.

```
SELECT      *
FROM        Clientes
WHERE       cod_cliente NOT IN
            (SELECT cod_cliente  FROM  Cuentas)
```

Sentencia equivalente (correlacionada):

```
SELECT      *
FROM        Clientes
WHERE       NOT EXISTS
            (SELECT *
             FROM    Cuentas
             WHERE  Cuentas.cod_cliente = Clientes.cod_cliente)
```

Se utiliza NOT EXISTS cuando se desea evaluar un NOT IN con más de una columna.

Ej: "Movimientos para los cuales no se ha ingresado una cotización aun (están en tabla MOVIMIENTOS pero NO en COTIZACIONES)."

```
SELECT  M.id_mov, M.fecha, M.cuenta, C.moneda
FROM    movimientos M,  cuentas C
WHERE  M.cuenta = C.cuenta
   AND NOT EXISTS
            ( SELECT *
              FROM  Cotizaciones COT
              WHERE  COT.moneda = C.moneda
              AND  COT.fecha = M.fecha )
ORDER BY 1, 2, 3
```

Uso en HAVING

Se pueden utilizar en la cláusula HAVING para mayor expresividad.

"Clientes con un saldo en plazo fijo igual al más alto de todos los plazos fijos."

```
SELECT  cliente, sum(saldo)
FROM    Cuentas
WHERE  producto = 3 and moneda = 1
GROUP BY cliente
HAVING  sum (saldo) = (SELECT max (saldo)
                       FROM   cuentas
                       WHERE  producto = 3 AND
                              moneda = 1)
```

UNION

- Combina múltiples consultas en una sola.
- Facilita ordenamiento no posible con una consulta simple.
- UNION : Operador que une el resultado de dos o más Consultas en una Consulta Simple.

Dos tipos:

UNION: Excluye los resultados repetidos de las consultas unidas.

UNION ALL: Incluye TODAS las tuplas de las consultas unidas (aún con repetición).

Sintaxis:

```
SELECT lista de columnas
FROM    tablas
[WHERE condicion]
UNION  [ALL]
SELECT lista de columnas
FROM tablas
[WHERE condicion]
[Order By lista de columnas]
```

Condiciones y Requisitos:

- La cantidad de columnas en cada sentencia SELECT debe ser la misma.
- El tipo de datos de cada columna entre los dos SELECT's debe coincidir. No se exige que sea la misma columna, ni siquiera que posea el mismo nombre.
- Si deseo ordenar la salida, debo ubicar la sentencia ORDER BY al final de la consulta. Referencio las columnas por sus posiciones.

Ejemplo:
```
SELECT   unique cuentas.cuenta, cuentas.producto
FROM     movimientos, cuentas
WHERE    movimientos.cuenta = cuentas.cuenta
         and sucursal = 4
         and importe > 3000000
UNION
SELECT   unique cuentas.cuenta, cuentas.producto
FROM     cuentas, cuentas_intereses
WHERE    cuentas.cuenta =cuentas_intereses.cuenta
         and cuentas_intereses.interes = 0;
```

Resultado con **UNION**

Cuenta	Producto
95	3
100	3
102	3
188	3
247	3
256	3
266	3
338	3
570	3
578	3
599	3
610	3
947	3
1422	3
1423	3
23213	3
101801	3
102002	3
105612	3
105622	3

Resultado con **UNION ALL**

Cuenta	Producto
95	3
100	3
102	3
188	3
247	3
256	3
266	3
338	3
570	3
578	3
578	3
599	3
610	3
947	3
1422	3
1422	3
1423	3
23213	3
101801	3
102002	3
105612	3
105622	3

Ejemplos Válidos:

1)
```
SELECT      cod_cliente, nro_cuenta, cod_moneda, 'Cliente 102' as Grupo
FROM        Cuentas
WHERE       cod_cliente = 102
UNION
SELECT      cod_cliente, nro_cuenta, cod_moneda , 'Euros' as Grupo
FROM        Cuentas
WHERE       cod_moneda = 3 and cod_cliente <> 102
ORDER BY  3,1,2
```

2)
```
SELECT      E.id_persona, P.nombre
FROM        Personas P, Empleados E
WHERE       P.id_persona = E.id_persona
UNION
SELECT      C.cod_cliente, P.nombre
FROM        Personas P, Clientes C
WHERE       P.id_persona = C.id_persona
```


Ejemplos Incorrectos:

1)
```
SELECT      cod_cliente, fec_apertura
FROM        Cuentas
WHERE       cod_cliente = 102
UNION
SELECT      cod_cliente, nro_cuenta
FROM        Cuentas
WHERE       cod_moneda = 3 and cod_cliente <> 102
```

2)
```
SELECT      E.id_persona, P.nombre, E.fec_ingreso
FROM        Personas P, Empleados E
WHERE       P.id_persona = E.id_persona
UNION
SELECT      C.cod_cliente, P.nombre, ???????
FROM        Personas P, Clientes C
WHERE       P.id_persona = C.id_persona
```

Modificación de Datos (INSERT, UPDATE, DELETE)

INSERT

Única forma de agregar nuevos registros a una tabla.

Se pueden insertar valores NULL explícitamente, siempre que la definición de la columna lo permita.

Dos variantes:

1. INSERT con valores explícitos (una tupla a la vez).
2. INSERT con valores tomados de otra tabla.

En ambos tipos se pueden especificar los campos EXPLICITAMENTE o NO:

INSERT INTO T ……

INSERT INTO T (campo1, campo2, ……) ……..

Se recomienda poner siempre los campos a insertar EXPLÍCITAMENTE.

INSERT Tipo 1: Valores dados explícitamente.

Sintaxis:

INSERT INTO <tabla> [(columnas)] VALUES (v1, v2,… , vn).

Ej: (2 modos equivalentes de escribirlo):

1. Tabla : EMPLEADOS(nro_emp, nombre, direccion).
   INSERT INTO EMPLEADOS VALUES (1,'Juan','Colonia 6499');
2. INSERT INTO EMPLEADOS(nro_emp, nombre, direccion)
   VALUES (1,'Juan','Colonia 6499');

INSERT Tipo 2: Trayendo datos de otra tabla

Si una sola tupla fallara no se insertá NINGUNA: es una sola sentencia.

Sintaxis :

INSERT INTO <tabla> [(columnas)] SELECT <campos> FROM …..

Ej:
```
INSERT INTO MovCta545  (fecha, sucursal, importe)
      SELECT  fecha, sucursal, importe
          FROM movimientos
          WHERE cuenta = 545;
```

UPDATE

Actualiza datos de una Tabla.

Sintaxis:

```
UPDATE <tabla>
SET <campo1> = <valor1>,  <campo2> = <valor2>, ….. ,
      <campoN> = <valorN>
[ WHERE <condicion> ]
```

IMPORTANTE:

1. Si no se especifica WHERE el cambio se aplica a TODA LA TABLA.
2. Si el WHERE no es por un campo clave se corre el riesgo de actualizar tuplas que no se deseaba.
3. El UPDATE puede fallar si:
   a) Se intentan asignar valores NULL a columnas que no lo permiten.
   b) Se violan CHECKS.
   c) Se viola una PRIMARY KEY.

Ejemplo 1:

```
UPDATE EMPLEADOS
SET  nombre = 'Juan Martínez',  direccion = 'Colonia 6401'
WHERE nro_emp = 1;
```

Modifica nombre y dirección del empleado con codigo = 1

Ejemplo 2:

```
UPDATE EMPLEADOS
SET direccion= 'Mercedes 3423'
WHERE nombre like  '%Perez%'
```

Atención: modifica TODOS los Perez, pueden modificarse más de los que deseábamos.
Sugerencia: hacer antes un SELECT COUNT(*) …. .

Ejemplo 3:

```
UPDATE EMPLEADOS
SET direccion = ''
```

Deja en blanco TODAS las direcciones de la tabla EMPLEADOS.

Variante: UPDATE con subquery relacionado.
Ejemplo:
Recalcular el saldo de todas las cuentas.

```
UPDATE  Cuentas
      SET  saldo = (SELECT SUM (importe)
                    FROM   Movimientos
                    WHERE Cuentas.cuenta =  Movimientos.cuenta
                    )
```


DELETE

Elimina filas de una Tabla.

- Al igual que UPDATE se debe especificar con WHERE cuales tuplas se desean borrar.
- Puede fallar si se violaran restricciones de integridad.

Caso típico: se borran tuplas de tabla referenciada por FOREIGN KEYS desde otras.
Si ese fuera el caso no se borra NINGUNA tupla.
Sintaxis:

```
DELETE FROM <tabla>  WHERE <condicion>
```

Ejemplos:

1) DELETE FROM Clientes WHERE cliente = 10001;
2) DELETE FROM Empleados;
   Borra el contenido de TODA la tabla Empleados.
3) Quiero eliminar las personas del departamento de MALDONADO.

```
DELETE FROM Personas
WHERE departamento LIKE "M%O";
```

Atención: No se usó la Clave primaria y obtengo:

```
635 row(s) deleted.
```

Hemos eliminado también los clientes de "MONTEVIDEO".

Modelos de Datos y DBMS

Modelos de Datos:

¿QUÉ SON?
Lenguajes usados para especificar BDs.
Un Modelo de Datos permite expresar:

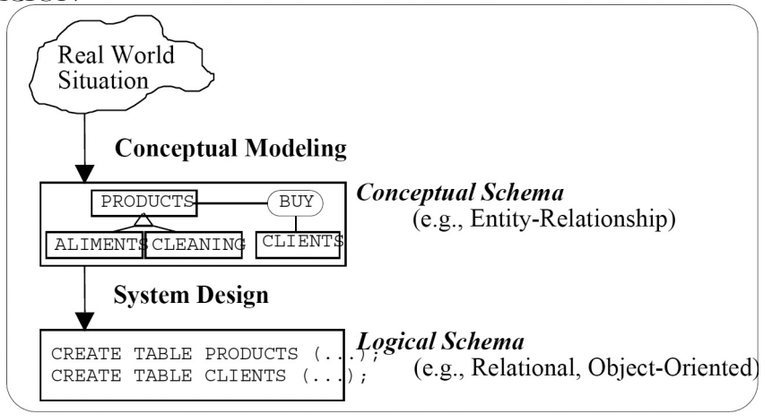
- Estructuras
  Objetos de los problemas:
  Por ejemplo: CURSOS(nro_curso, nombre, horas).
- Restricciones
  Reglas que deben cumplir los datos.
  Por ejemplo: ($\forall c \in CURSOS$) (c.horas < 120)
- Operaciones.
  Insertar, borrar y consultar la BD.
  Por ejemplo: Insert into CURSOS (303,"BD",90).

CLASIFICACIÓN
Según el nivel de abstracción:

- Conceptuales
  Representan la realidad independientemente de cualquier implementación de BD.
  Usado en etapa de Análisis.

- Lógicos
  Implementados en DBMSs.
  Usado en etapas de Diseño e Implementación.
- Físicos
  Implementación de estructuras de datos.
  Ej.: Arboles B, Hash.

APLICACIÓN

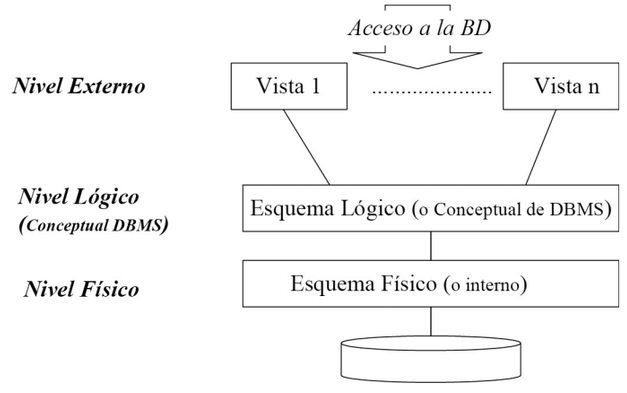


Esquema de una BD:
- Tipos de datos existentes.
  Por ejemplo:
    CURSOS(nro_curso, nombre, horas).
    ESTUDIANTES(CI, nombre, fecha_nacimiento).
    TOMA_CURSO(nro_curso, CI).
- Muy estables.

Instancia de una BD
- Datos almacenados.
- Muy volátiles.

Arquitectura lógica de DBMS
Propiedades importantes de DBMSs:
- Control global único de la BD.
- Separación entre esquema y aplicaciones.
  Esquema: visión global de los datos de la realidad.
  Aplicaciones: programas sobre la BD.
- Soporte a diferentes visiones de los datos.
  Usuarios/aplicaciones ven subconjuntos de la BD.
- Independencia de datos.
  Esquema lógico independiente de implementación.

ARQUITECTURA EN TRES NIVELES



Independencia de datos
- Independencia Lógica.
  - Independencia entre especificaciones de nivel Lógico y Externo.
  - Cambiar partes de esquema lógico sin afectar a los esquemas externos o a las aplicaciones.
- Independencia Física.
  - Independencia entre especif. de nivel Lógico y Físico.
  - Cambiar implementaciones sin afectar esq. Lógico.

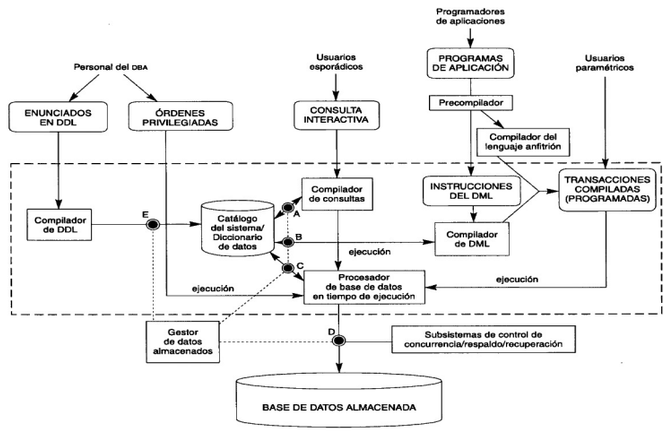
Lenguajes e Interfaces en ambientes BD
- Provistos por DBMS:
  - Definición de esquema:
    - VDL (o SSDL) - View Definition Language.
    - SDL - Storage Definition Language.
    - DDL - Data Definition Language.
      - Suele englobar estos tres lenguajes.
  - Manipulación de la BD:
    - DML - Data Manipulation Language.
      - Modificaciones en instancias.
    - QL - Query Language.
      - Subconjunto del DML, sólo para consultas.

- Tipos de QL:
  - Declarativos.
    - Se especifica qué propiedad cumplen los datos.
    - No se especifica cómo se recuperan de la BD.
    - Suelen recuperar conjuntos de items (registros).
    - Es el DBMS que define el plan de ejecución.
  - Procedurales.
    - Se especifica un algoritmo que accede a estructuras del esquema lógico y recupera los datos ítem por ítem (registro a registro).

- Lenguajes de programación:
  - Lenguajes host (anfitrión):
    - Lenguajes de uso general (C, COBOL, etc.) en el cual se *embeben* sentencias de DML.
    - Se tiene un pre-procesador que traduce el programa con *DML embebido* en un programa puro.
    - PROBLEMAS: *impedance-mismatch*.
  - Lenguajes 4GL:
    - Lenguajes procedurales orientados a acceso a BDs.
    - Conexión privilegiada con DMLs, reduce el *impedance-mismatch*.

- Interfaces especializadas:
  - Interfaces gráficas de consulta.
    - Se visualizan las estructuras en forma gráfica.
    - Resultados como gráficas (torta, lineas, etc.).
  - Interfaces de Lenguaje Natural.
    - Se procesan frases y se traducen al QL.
  - Interfaces para Administración.
    - Ambientes especializados.

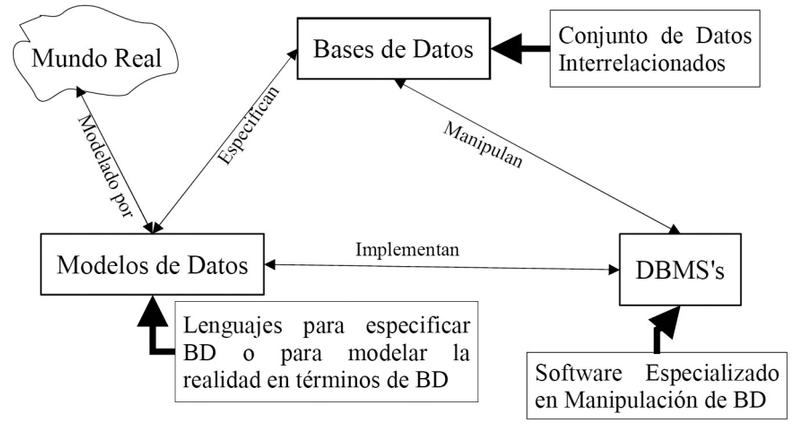
ESTRUCTURA DE DBMS



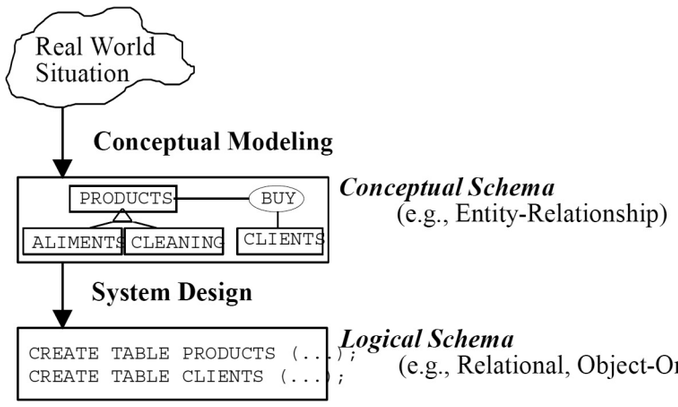
Diferentes tipos de DBMS

- Según el Modelo de Datos:
  - Relacional.
  - Orientado a Objetos.
  - Otros: Redes, Jerárquico, Deductivo, ...
- Según el porte:
  - Desktop (escritorio) / mono-usuario.
  - Servidor / multi-usuario.
- Según distribución de la BD:
  - Centralizado.
  - Distribuido.

RESUMEN DE LOS ELEMENTOS DE BASES DE DATOS



MODELOS DE DATOS



Calidad de Esquemas Conceptuales

Para asegurar la calidad de los esquemas conceptuales se define un conjunto de propiedades que se deben chequear durante y al final de su desarrollo:

- Completitud. } Maximizar
- Correctitud. } Maximizar
- Minimalidad.
- Expresividad. } Balancear
- Explicitud. } Balancear

- Completitud

Un esquema es completo cuando representa todas las características relevantes del problema.

Chequeo:

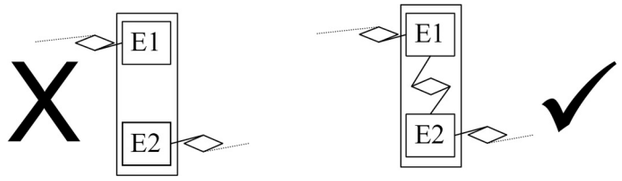
- Controlar que todos los conceptos del problema estén representados en alguna parte del esquema.
- Controlar que todos los requerimientos sean realizables con el esquema.
- Leer el resultado y compararlo con la descripción original.

- Correctitud

  Hay dos tipos:

  - Sintáctica: Habla de la forma en que se especifica el esquema con respecto al lenguaje usado para hacer esa especificación.
  - Semántica: Habla de la forma en que la especificación representa el problema.

Correctitud Sintáctica:
Un esquema es *correcto sintácticamente* cuando las distintas partes de éste están construidas correctamente con respecto al lenguaje utilizado.
Ej.: Las agregaciones se construyen sobre una relación, no sobre dos entidades cualesquiera u otra cosa.



Chequear:
- Existencia de cardinalidades en cada relación.
- Existencia de atributos determinantes en cada entidad. Si no existen, entonces verificar que sea entidad débil con respecto a otra.
- Existencia de una y sólo una relación y todas las entidades que intervienen en la misma dentro de cada agregación.

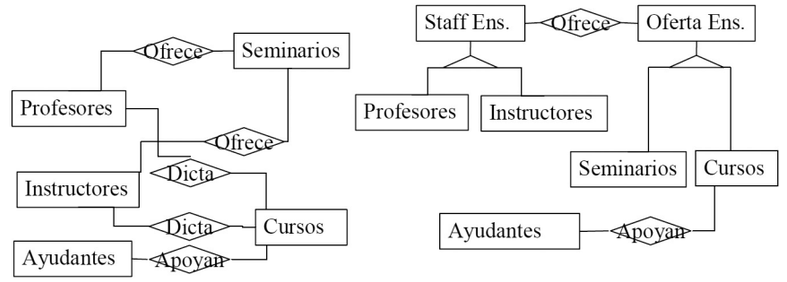
Correctitud Semántica:
Un esquema es *correcto semánticamente* si cada elemento del problema se representa utilizando estructuras adecuadas.

- Chequear o Responder para cada concepto del problema (de la realidad):
  - ¿Atributo o Entidad o Relación?
  - ¿Una sola categoría de entidades o más de una?
  - ¿Una Relación es binaria o múltiple?
  - ¿Cuál es el mecanismo de determinación del conjunto de entidades?
  - Las cardinalidades y totalidades, ¿tienen sentido?
- En general: la representación, ¿tiene sentido con respecto a la realidad?

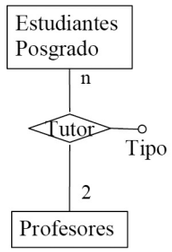
Minimalidad:
Un esquema es *minimal* si cualquier elemento de la realidad aparece sólo una vez en el esquema.

- Chequear:
  - Donde está representado en el esquema cada elemento de la realidad.
  - A qué elemento de la realidad corresponde cada elemento del esquema.
  - Controlar atributos calculados.

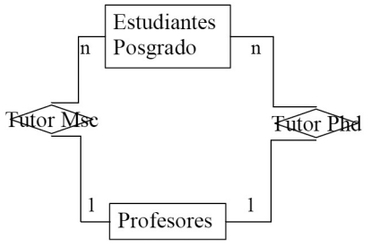
Expresividad:
Un esquema es *expresivo* si representa la realidad en una forma natural que puede ser fácilmente comprensible usando sólo la semántica del modelo.



Explicitud:
Un esquema es *explícito* si no utiliza más formalismos que el diagrama E-R.



Si un mismo estudiante tiene 2 tutores, entonces el tipo de uno es PHD y el otro es MSC.



Resumen:
Hay cinco propiedades fundamentales a controlar:

- Completitud
- Correctitud
- Minimalidad
- Expresividad
- Explicitud

Para las tres primeras propiedades se definieron criterios elementales de Chequeo.
Todas las propiedades se deben balancear, buscando un buen diseño:

- Hay que buscar esquemas completos y correctos, minimales, expresivos y explícitos.

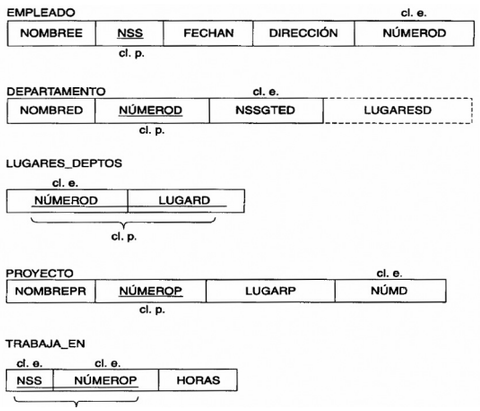
Diseño de Base de Datos Relacional

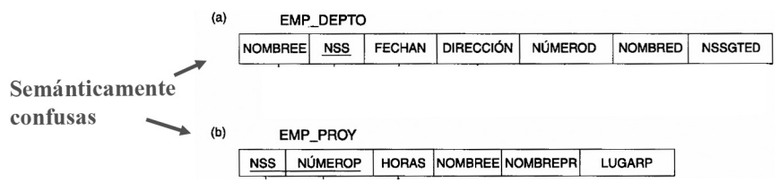
Pautas informales para el diseño.

Cuatro medidas informales de la calidad:

– Semántica de los atributos.
– Reducción de los valores redundantes en las tuplas.
– Reducción de los valores nulos en las tuplas.
– No generación de tuplas erróneas.

• Semántica de los atributos:
Ejemplos:





Pauta 1:

Diseñe un esquema de relación de modo que sea fácil explicar su significado. No combine atributos de varios tipos de entidades y tipos de vínculos en una sola relación.

• Reducción de los valores redundantes:
Información redundante en las tuplas.

EMP_DEPTO

NOMBREE	NSS	FECHAN	DIRECCIÓN	NÚMEROD	NOMBRED	NSSGTED
Silva, José B.	123456789	09-ENE-55	Fresnos 731, Higueras, MX	5	Investigación	333445555
Vizcarra, Federico T.	333445555	08-DIC-45	Valle 638, Higueras, MX	5	Investigación	333445555
Zapata, Alicia J.	999887777	19-JUL-58	Castillo 3321, Sucre, MX	4	Administración	987654321
Valdés, Jazmín S.	987654321	20-JUN-31	Bravo 291, Belén, MX	4	Administración	987654321
Nieto, Ramón K.	666884444	15-SEP-52	Espiga 957, Heras, MX	5	Investigación	333445555
Esparza, Josefa A.	453453453	31-JUL-62	Rosas 5631, Higueras, MX	5	Investigación	333445555
Jabbar, Ahmed V.	987987987	29-MAR-59	Dalias 980, Higueras, MX	4	Administración	987654321
Botello, Jaime E.	888665555	10-NOV-27	Sorgo 450, Higueras, MX	1	Dirección	888665555

EMPLEADO

NOMBREE	NSS	FECHAN	DIRECCIÓN	NÚMEROD
Silva, José B.	123456789	09-ENE-55	Fresnos 731, Higueras, MX	5
Vizcarra, Federico T.	333445555	08-DIC-45	Valle 638, Higueras, MX	5
Zapata, Alicia J.	999887777	19-JUL-58	Castillo 3321, Sucre, MX	4
Valdés, Jazmín S.	987654321	20-JUN-31	Bravo 291, Belén, MX	4
Nieto, Ramón K.	666884444	15-SEP-52	Espiga 957, Heras, MX	5
Esparza, Josefa A.	453453453	31-JUL-62	Rosas 5631, Higueras, MX	5
Jabbar, Ahmed V.	987987987	29-MAR-59	Dalias 980, Higueras, MX	4
Botello, Jaime E.	888665555	10-NOV-27	Sorgo 450, Higueras, MX	1

DEPARTAMENTO

NOMBRED	NÚMEROD	NSSGTED
Investigación	5	333445555
Administración	4	987654321
Dirección	1	888665555

Anomalías de actualización:

– Anomalías de inserción.
– Anomalías de eliminación.
– Anomalías de modificación.

Pauta 2

Diseñe los esquemas de las relaciones de modo que no haya anomalías de inserción, eliminación o modificación en las relaciones. Si hay anomalías señálelas con claridad a fin de que los programas que actualicen la BD operen correctamente.

• Valores nulos en las tuplas
Posibles problemas:

– Desperdicio de espacio.
– Dificultad para entender el significado.
– Aplicación de funciones agregadas (count,sum).
– Múltiples interpretaciones.

Pauta 3

Hasta donde sea posible, evite incluir en una relación atributos cuyos valores pueden ser nulos. Si no es posible, asegúrese de que se apliquen solo en casos excepcionales y no a la mayoría de las tuplas de una relación.

• Tuplas erróneas

Ejemplo 1: Se aplica proyección a EMP-PROY.

EMP_PROY

NSS	NÚMEROP	HORAS	NOMBREE	NOMBREPR	LUGARP
123456789	1	32.5	Silva, José B.	ProductoX	Belén
123456789	2	7.5	Silva, José B.	ProductoY	Sacramento
666884444	3	40.0	Nieto, Ramón K.	ProductoZ	Higueras
453453453	1	20.0	Esparza, Josefa A.	ProductoX	Belén
453453453	2	20.0	Esparza, Josefa A.	ProductoY	Sacramento
333445555	2	10.0	Vizcarra, Federico T.	ProductoY	Sacramento
333445555	3	10.0	Vizcarra, Federico T.	ProductoZ	Higueras
333445555	10	10.0	Vizcarra, Federico T.	Automatización	Santiago
333445555	20	10.0	Vizcarra, Federico T.	Reorganización	Higueras

LUGARES_EMP

NOMBREE	LUGARP
Silva, José B.	Belén
Silva, José B.	Sacramento
Nieto, Ramón K.	Higueras
Esparza, Josefa A.	Belén
Esparza, Josefa A.	Sacramento
Vizcarra, Federico T.	Sacramento
Vizcarra, Federico T.	Higueras
Vizcarra, Federico T.	Santiago

EMP_PROY1

NSS	NÚMEROP	HORAS	NOMBREPR	LUGARP
123456789	1	32.5	ProductoX	Belén
123456789	2	7.5	ProductoY	Sacramento
666884444	3	40.0	ProductoZ	Higueras
453453453	1	20.0	ProductoX	Belén
453453453	2	20.0	ProductoY	Sacramento
333445555	2	10.0	ProductoY	Sacramento
333445555	3	10.0	ProductoZ	Higueras
333445555	10	10.0	Automatización	Santiago
333445555	20	10.0	Reorganización	Higueras

Ejemplo 2: Se aplica join natural a EMP-PROY1 y LUGARES-EMP.

NSS	NÚMEROP	HORAS	NOMBREPR	LUGARP	NOMBREE
123456789	1	32.5	ProductoX	Belén	Silva, José B.
* 123456789	1	32.5	ProductoX	Belén	Esparza, Josefa A.
123456789	2	7.5	ProductoY	Sacramento	Silva, José B.
* 123456789	2	7.5	ProductoY	Sacramento	Esparza, Josefa A.
666884444	3	40.0	ProductoZ	Higueras	Nieto, Ramón K.
* 666884444	3	40.0	ProductoZ	Higueras	Vizcarra, Federico T.
* 453453453	1	20.0	ProductoX	Belén	Silva, José B.
453453453	1	20.0	ProductoX	Belén	Esparza, Josefa A.
			⋮		
333445555	10	10.0	Automatización	Santiago	Vizcarra, Federico T.
* 333445555	20	10.0	Reorganización	Higueras	Nieto, Ramón K.
333445555	20	10.0	Reorganización	Higueras	Vizcarra, Federico T.

Pauta 4

Diseñe los esquemas de modo que puedan reunirse por condición de igualdad sobre atributos claves, para garantizar que no se formen tuplas erróneas.

Resumen

Problemas a evitar:

– Anomalías en insercion, modificacion y eliminacion de tuplas por *redundancia*.
– Desperdicio de espacio y dificultad para operaciones por *valores nulos*.
– Generación de datos erróneos por joins hechos *relacionando mal* las relaciones.

Entonces se presentarán…

– Conceptos y teorías formales para detectar y evitar estos problemas.

Dependencias Funcionales

Definición:

– Una df X→Y, entre 2 conjuntos de atributos X e Y que son subconjuntos de R especifica una restricción sobre las posibles tuplas que formarían una instancia r de R. La restricción dice que, **para 2 tuplas cualesquiera t_1 y t_2 de r tales que t1[X]=t2[X], debemos tener también $t_1[Y]=t_2[Y]$.**
– Observar:
  » Si X es una clave candidata de R, entonces X→Y para cualquier subconjunto de atributos Y de R.
  » Si X→Y en R, esto no nos dice si Y→X en R o no.
– Las dfs son propiedades de la semántica de los atributos.
– En el ejemplo de EMP_PROY, se cumplen:
  » NSS → NOMBREE, NUMEROP → {NOMBREPR,LUGARP}, {NSS,NUMEROP} → HORAS

Ejemplo 1 – Deducir atributos y dfs.

Una empresa de alquiler de vehículos desea implementar una base de datos con la información de su negocio. Se tienen vehículos identificados por su numero de matrícula, y de los que se conoce su marca, color, modelo y año. También se tienen clientes identificados por su número de cédula de identidad, y de los que se conoce su nombre, dirección y teléfono. Un contrato de alquiler de vehículo está identificado por un número de contrato y se realiza en una fecha dada entre un cliente y un vehículo, registrándose el periodo de alquiler en días y el precio del servicio. Se considera que en una misma fecha no se puede alquilar más de una vez el mismo vehículo al mismo cliente.

• Clausura de F - F+

Definición:

F - conjunto de dfs que se especifican sobre el esquema relación R.

F+ - conjunto de todas las dfs que se cumplen en todas las instancias que satisfacen a F.

Inferencia de dfs

Ejemplo:

F = { NSS → {NOMBREE,FECHAN,DIRECCION,NUMEROD},
NUMEROD → {NOMBRED,NSSGTED} }

Podemos inferir:

NSS → { NOMBRED, NSSGTED}, NUMEROD→NOMBRED

Reglas de inferencias para las dfs:

- **Reglas:** **(siendo X,Y,W,Z conjuntos de atributos)**
  - (RI1) reflexiva - Si $X \supseteq Y$, entonces $X \rightarrow Y$
  - (RI2) de aumento - $\{X \rightarrow Y\} \models XZ \rightarrow YZ$
  - (RI3) transitiva - $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
  - (RI4) descomposición - $\{X \rightarrow YZ\} \models X \rightarrow Y$
  - (RI5) unión - $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$
  - (RI6) pseudotransitiva - $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Reglas de Armstrong**: RI1 a RI3**

- Minimales: Las demás se pueden derivar a partir de estas tres.

Clausura de X bajo F - X+

- **Definición**:
  - X+ es el conjunto de atributos determinados funcionalmente por X

Algoritmo - Determinar X+ bajo F

```
X+ := X
repetir
  viejoX+ := X+;
  para cada df Y→Z en F hacer
        si Y⊆ X+ entonces X+ := X+ ∪ Z;
hasta que (viejoX+ = X+);
```

Ejemplo:

Dado EMP_PROY(NSS, NUMEROP, HORAS, NOMBREE, NOMBREPR, LUGARP)

F = { NSS → NOMBREE
NUMEROP → NOMBREPR,LUGARP
NSS,NUMEROP → HORAS }

podemos calcular:

```
{ NSS }+ = {
        NSS,
        NOMBREE
}
{ NUMEROP }+ = {
        NUMEROP,
        NOMBREPR,
        LUGARP
}
```


Ejercicio: Clausuras de Atributos

Hallar la clausura de los siguientes conjuntos de atributos:

- {nro_mat}, {nro_mat, ci_cli}, {nro_contrato}, {marca}, {fecha, ci_cli, nro_mat}

Dar alguna dependencia funcional que pertenezca a F+ y no a F.

Equivalencia de conjuntos de dfs

Definición:

- Dos conjuntos de dfs E y F son equivalentes sii E+ = F+.

Podemos decir...

- Todas las dfs en E se pueden inferir de F y todas las dfs en F se pueden inferir de E.
- E cubre a F y F cubre a E.

¿Cómo determinamos si F cubre a E?

- Para cada df $X \rightarrow Y \in E$, calculamos X+(F) y verificamos que X+ incluya los atributos en Y.

Equivalencia de conjuntos de dfs

Ejemplo:

- F={AB →C, B→D,D→GC,CG→H}
- F_1={D→H,B→C,AD→GH}
  - »¿F_1 cubre a F?
  - »¿F cubre a F_1?
  - »¿F es equivalente a F_1?
- F_2={B→D,D→G,D→C,CG→H}
  - »¿Qué pasa entre F_2 y F?
  - »¿Qué pasa entre F_1 y F?
- Observar que F2 es más “simple” que F. Dado F, ¿siempre se puede encontrar un conjunto con estas características?

Conjunto minimal de dfs

F es minimal si:

- Toda df en F tiene un solo atributo a la derecha.
- No podemos reemplazar ninguna df $X \rightarrow A \in F$ por una df $Y \rightarrow A$, donde $Y \subset X$, y seguir teniendo un conjunto de dfs equivalente a F.
- No podemos quitar ninguna df de F y seguir teniendo un conjunto de dfs equivalente a F.

Conjunto minimal de dfs

Definición:

– Un **cubrimiento minimal** de F es un conjunto minimal F_{min} que es equivalente a F.

Encontrar un cubrimiento minimal

Algoritmo

1. Hacer G := F;
2. Reemplazar cada df $X \rightarrow A_1, A_2, \ldots, A_n$ en G por las n dfs $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$;
3. Para cada df $X \rightarrow A$ en G
   { calcular X+ respecto a (G - ($X \rightarrow A$));
   si X+ contiene a A, eliminar $X \rightarrow A$ de G };
4. Para cada df restante $X \rightarrow A$ en G
   para cada atributo B que sea un elemento de X
   { calcular (X - B)+ respecto a G;
   si (X - B)+ contiene a A,
   reemplazar $X \rightarrow A$ por (X - B)$\rightarrow A$ en G };

Encontrar un cubrimiento minimal

Ejemplo:

– F={AB →C, B→D,D→GC,CG→H}
– Paso 1: Cada dependencia que tiene varios atributos a la derecha, es sustituida por las dependencias a los atributos individuales.
  » F_1= {AB →C, B→D,D→G,D→C,CG→H}
– Paso 2: Estudiamos atributos redundantes.
  » B+={B,D,G,C,H} entonces F_2={B→C, B→D,D→G,D→C,CG→H}
– Paso 3: Estudiamos dependencias redundantes.
  » Con respecto a F_2-{B→C}, B+={B,D,G,C,H} entonces F_3={B→D,D→G,D→C,CG→H} Minimal.

6. Arquitectura del Computador y Sistemas Operativos

Ing. Pablo Gestido

Estructura de los sistemas de computación

Agenda

- Componentes de un sistema:
  - Introducción.
  - CPU (procesador).
  - Memoria.
  - Dispositivos de Entrada/Salida (IO).
- Protección de hardware:
  - Modo dual.
  - Protección de E/S.
  - Protección de Memoria.
  - Protección de CPU.
- Red:
  - Local Area Networks.
  - Wide Area Networks.
  - Topologías de red.

Componentes de un sistema

CPU (procesador)

Unidad central de procesamiento (procesador). Permite ejecutar un conjunto de instrucciones. Su velocidad es varios ordenes mayor con respecto al acceso a la memoria.

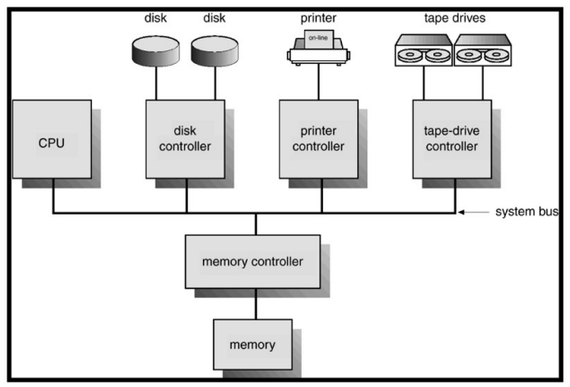
Memoria

Permite mantener la información disponible. Existe una jerarquía de memoria: registros, caches, memoria física de tipo RAM (*Random Access Memory*), dispositivos magnéticos y ópticos.

Dispositivos de Entrada/Salida (IO)

Permite interactuar con el sistema. Algunos dispositivos más comunes: Impresoras, teclados, ratón, video, disco, red, etc.

Esquema gráfico:



CPU (Procesador)

- La unidad central de procesamiento es la que ejecuta los programas. En un sistema pueden haber más de una.

- El ciclo básico consiste en tomar la instrucción apuntada por el PC (*program counter*) (*fetching*), decodificarla para determinar su tipo y operandos (*decoding*), ejecutarla (*executing*) y luego continuar con la siguiente instrucción.

- Arquitecturas modernas aumentan la *performance* ejecutando las operaciones en paralelo (*fetching, decoding, executing*). Esta técnica es conocida como *pipeling*.

- Existen varias arquitecturas de procesador que se clasifican en RISC (*Reduced Instruction Set Computer*) o CSIC (*Complex Instruction Set Computer*). Algunas arquitecturas: SPARC, POWER, x86, Itanium.

- La velocidad del procesador es varios órdenes de magnitud mayor que la velocidad de acceso a información que está en la memoria volátil (RAM).

- Esto implicó la creación de registros a nivel del procesador y, finalmente, una *cache* de memoria (*caches* de 1er Nivel, 2do Nivel y hasta 3er Nivel).

- Los registros son la memoria más rápida que accede un procesador y están integrados al *chip*.

- En los últimos años han surgido procesadores que en un mismo *chip* contienen varios *cores* de ejecución. Esto ha llevado a una nueva terminología: single-core, dual-core, quad-core, etc.

Instrucciones

Operador Operandos...

- Los operandos pueden ser: inmediatos, registros, relativos o de memoria DS:[SI] según diferentes técnicas (vistos en Arquitectura de sistemas).

- Las familias de instrucciones incluyen: aritméticas, lógicas, transferencia control (Jmp, Call, Loop, etc), de memoria, de stack, de sincronización (loopz Lock:XChg ax) y de entrada salida.

CPU (Procesador)

- Dentro del mismo *chip* del procesador se incluyen registros de rápido acceso:
  - Registros punto fijo y punto flotante (decimal o no).
  - Registros de direccionamiento ES, SS, DS, CS, etc.
  - Registro de Estado. Incluye PC y banderas con zero, carry.
  - Caches:
    - 1er Nivel (del orden de 20 Kb).
    - 2do Nivel (del orden de 512Kb a 2Mb).
    - 3er Nivel (del orden de 8Mb).

<u>Instrucciones Privilegiadas</u>

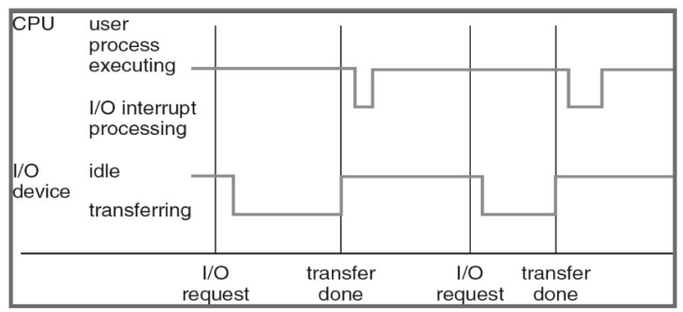
- Se establecen niveles de ejecución y conjunto de instrucciones para cada nivel.

- Un protocolo seguro para aumentar el nivel de ejecución que se basa en siempre transferir el control a código autenticado (*trusted*) para aumentar el nivel de ejecución.

- Familias de E/S, protección de memoria y memoria virtual.

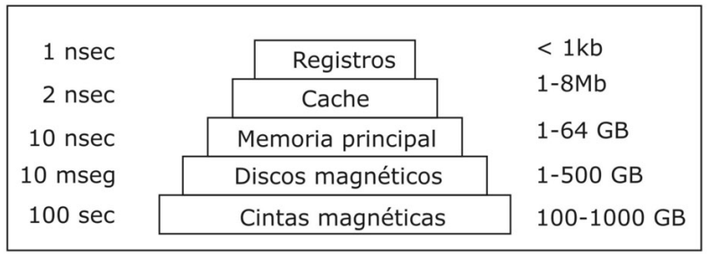
<u>Interrupciones</u>

- Capacidad de procesar interrupciones.
- El sistema operativo preserva el estado actual (previo a la interrupción) del procesador (registros, etc.).
- Se determina que tipo de interrupción ocurrió.
- Se ejecuta la rutina de atención correspondiente.



Memoria

- El sistema de memoria es construido en base a una jerarquía que permite mejorar la utilización del procesador:



Memoria - Cache

- El *cache* es un principio muy importante, es utilizado a varios niveles en el sistema de computación (hardware, sistema operativo, software).

- El concepto es mantener una copia de la memoria que está siendo utilizada en un medio temporal de mayor velocidad de acceso.

- El medio de memoria *cache* es mucho menor en capacidad pero más veloz que el dispositivo principal. Esto genera que el manejo de *cache* es un problema de diseño importante.

- El tamaño del *cache* y sus políticas de reemplazo tienen un alto impacto en la mejora real de la *performance*.

Memoria – Coherencia de cache

- Un problema que introduce la memoria *cache* en ambientes de multiprocesadores es la coherencia y consistencia de los datos que están replicados. *Caches* en multiprocesadores:

  - Mayor rendimiento: no se satura el bus del sistema (cuello de botella).
  - Aun en un mono procesador, hay que contemplar a los controladores de dispositivos.
  - Problemas de coherencia entre *caches*, ya que una palabra puede estar replicados en diferentes *cache* de los procesadores.
  - A su vez, en sistemas multiprocesadores, el problema de coherencia se torna mucho más complicado.
  - Surgen técnicas como *write-update* y *write-invalidated*.

Memoria – Memoria Principal (RAM)

- Memoria de tipo volátil, con direcciones de palabra o byte.

- Palabra de 32, 48, 64 bits con paridad.

- Transferencia en un ciclo del bus y acceso en paralelo (*interleaving*) a más de un módulo de memoria.

- Existen instrucciones que toman como argumentos direcciones de memoria.

- Es útil también para hacer transferencias con controladoras de dispositivos. Las controladoras tienen su propio *buffer* de memoria, y existen instrucciones de E/S que permiten la transferencia directa desde el *buffer* a memoria principal.

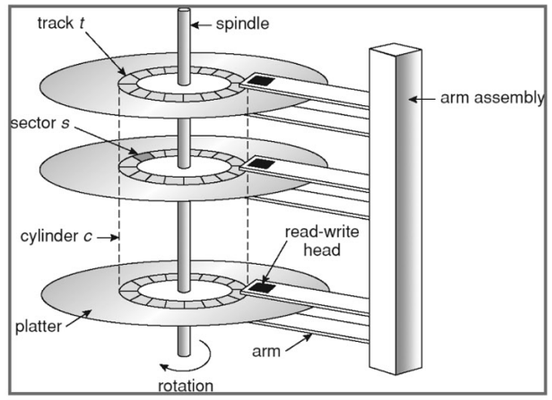
Memoria – Discos magnéticos (*hard disk*)

- Dispositivos de velocidad de acceso mucho menor que la memoria principal, pero de mayor capacidad.

- Tiene componentes mecánicas a diferencia de la memoria principal, *cache* y registros. Consta de platos de metal (10000 más *rpm*) y un brazo mecánico que contiene las cabezas de lectura/escritura para cada plato:
  - Pistas (*tracks*): La superficie de los platos es dividida lógicamente en pistas circulares.
  - Sectores (*sectors*): Cada pista es dividida en un conjunto de sectores.
  - Cilindros (*cylinder*): El conjunto de pistas (de todos los platos) que están en una posición del brazo mecánico forman un cilindro.

Memoria – Discos magnéticos (*hard disk*)

Esquema de discos magnéticos:



Memoria – Discos magnéticos (*hard disk*)

- La velocidad del disco tiene dos componentes:
  - Tasa de transferencia (*transfer rate*): Es la tasa con la cual los datos van entre el disco y la computadora.
  - Tiempo de posicionamiento (*positioning time*): Es el tiempo que se tarda en ubicar el brazo en el cilindro adecuado (*seek time*) más el tiempo de rotar el plato al sector adecuado (*rotational latency*).

- La unidad de transferencia es el bloque. Ocasionalmente, los bloques pueden estar con *interleaving*.

- Existen distintos tipos de *buses* de conexión:
  - IDE (*Integrated drive electronics*).
  - ATA (*Advanced Technology Attachment*).
  - SCSI (*Small Computer-Systems Interface*).
  - SAS (*Serial Attached SCSI*).

Dispositivos de Entrada/Salida (IO)

- Los dispositivos, por lo general, se componen de una controladora y el dispositivo en sí.

- La controladora es un *chip* que controla físicamente al dispositivo. Acepta comandos del sistema operativo y los ejecuta (genera las correspondientes señales sobre el dispositivo para realizar la tarea).

- La interfaz que le presenta la controladora al sistema operativo es bastante más simple que la provista por el dispositivo.

- En un sistema existen distintas controladoras (de discos, red, etc.), por eso es necesario distintos componentes de software para manejar cada uno.

Entrada/Salida – Device Drivers

- Al software que se comunica con la controladora se le denomina *device driver*.

- Para cada controladora se debe proveer el *device driver* adecuado. Estos son incorporados al sistema operativo dado que son la vía de comunicación con los dispositivos.

- Los *device drivers* son cargados de diferentes formas:
  - Ensamblados estáticamente al núcleo del sistema.
  - Cuando se carga el sistema se lee un archivo de configuración que menciona cuales *device drivers* cargar.
  - Cargar dinámicamente a demanda.

Dispositivos de Entrada/Salida (IO)

- Las controladoras contienen un conjunto de registros que sirven para comunicarse con ella y ejecutar comandos. Ej.: la controladora de un disco podría tener registros para especificar la dirección en disco, la dirección en memoria principal, el número de sectores y el sentido (lectura y escritura).

- Acceso a los registros de la controladora:
  - *Memory-mapped-IO*: Los registros son "mapeados" a direcciones de memoria principal.
  - *Direct IO instructions*: A los registros se le asigna una dirección de puerto (*IO port address*).

Entrada/Salida – *Memory-mapped-IO*

- Para facilitar el acceso a registros de los dispositivos, se reserva un espacio de la memoria principal que "mapea" a los registros del dispositivo.

- Leer o escribir en los registros de los dispositivos se traduce en leer o escribir sobre las direcciones de memoria. Al operar sobre estas direcciones de memoria se genera la transferencia a los registros del dispositivo en forma transparente.

- Las direcciones de memoria deben ser puestas fuera del alcance de los procesos del usuario.

- Ej.: La pantalla es "mapeada" a un lugar de memoria. Para desplegar un carácter en pantalla solo basta con escribir sobre el lugar correcto de la memoria principal.

Entrada/Salida – *IO port address*

- A cada registro se le asigna una dirección de puerto.
- El sistema cuenta con instrucciones privilegiadas *IN* y *OUT* que permiten a los *device drivers* leer o escribir en los registros de la controladora.
- La instrucción genera señales en el *bus* del sistema para seleccionar el dispositivo adecuado.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Entrada/Salida – Comparación de acceso

- **Los sistemas manejan los dos métodos de accesos a los registros de la controladora.**
- **El acceso *memory-mapped IO* no necesita de instrucciones privilegiadas.**
- **El acceso a través de instrucciones tiene la ventaja de no consumir memoria principal.**

E/S – Interacción con la controladora

- **Métodos para efectuar una operación de entrada-salida:**
  - Espera activa (*Polling*): **El procesador le comunica un pedido a la controladora del dispositivo y queda en un '*busy waiting*' consultando a la controladora si está listo el pedido.**
  - Interrupciones (*Interrupts*): **El procesador le comunica el pedido a la controladora y se libera para realizar otras tareas. Al culminar el pedido el dispositivo, la controladora genera una interrupción al procesador.**
  - Acceso directo a memoria (*DMA – Direct Memory Access*): **Se utiliza un *chip* especial que permite transferir datos desde alguna controladora a memoria sin que el procesador tenga que intervenir en forma continua.**

Dispositivos de E/S – Espera activa

- **El sistema queda en *busy waiting* consultando un registro del controlador para saber si está listo.**
- **Ej.: Imprimir un *buffer* en una impresora.**

```
p = copy_from_user(buffer, k_buffer, count);
for (i = 0; i < count; i++) {
   while (*printer_status_reg != READY) ;
   *printer_data.register = p[i];
}

return_to_user();
```


Dispositivos de E/S - Interrupciones

- **El sistema se independiza del controlador, que genera una interrupción cuando finaliza el pedido.**
- **Es necesario tener un vector de rutinas de atención de interrupciones (*interrupt vector*), que es cargado cuando se inicia el sistema operativo.**
- **Ej.: Imprimir un *buffer* en una impresora.**

```
p = copy_from_user(buffer, k_buffer, count);
while (*printer_status_reg != READY) ;
i = 0;
*printer_data.register = p[i];

scheduler();
```

- **Ej.: Rutina de atención de la interrupción.**

```
if (i == count)
   unblock_user();
else {
   i++;
   *printer_data.register = p[i];
}

return_from_interrupt();
```

Dispositivos de E/S - DMA

- **Se dispone de un dispositivo especializado que permite realizar transferencias desde ciertos dispositivos a memoria. La transferencia se hace en paralelo mientras el procesador realiza otras tareas.**
- **El procesador carga ciertos registros en el controlador DMA para realizar el pedido. El controlador DMA se encarga de la tarea de transferencia, interrumpiendo al procesador cuando finalizó.**
- **Ej.: Imprimir un *buffer* en una impresora.**

```
p = copy_from_user(buffer, k_buffer, count);
set_up_DMA_controller();

scheduler();
```

Dispositivos de E/S - DMA

- Ej.: Rutina de atención de la interrupción de DMA.

```
unblock_user();
return_from_interrupt();
```

Protección de Hardware

- Con la introducción de sistemas multiprogramados y multiusuarios se empezaron a generar problemas en el uso de los recursos debido a procesos "mal programados" o "mal intencionados".

- Fue necesario la introducción de protección entre los distintos procesos que se ejecutaban en un sistema.

- El hardware fue suministrando a los sistemas operativos de mecanismos para la protección:
  - Modo Dual: **Se provee de al menos, dos modos de operación.**
  - Protección de E/S: **Todas las instrucciones de Entrada/Salida son privilegiadas.**
  - Protección de Memoria: **Evaluación de las direcciones de memoria a través de la MMU.**
  - Protección de CPU: **Introducción de un *timer* que permite limitar el uso de CPU.**

Modo Dual

- El hardware provee, al menos, dos modos de ejecución:

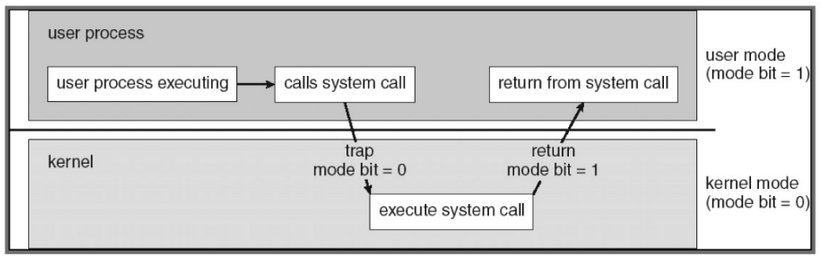
  - Modo usuario (*user mode*): **en este modo de ejecución se puede ejecutar un conjunto reducido de instrucciones de *hardware*. Los procesos a nivel de usuarios ejecutan en este modo.**
  - Modo monitor (*monitor mode*): **en este modo todas las instrucciones de *hardware* están disponibles. El sistema operativo es el único que debe ejecutar en este modo.**

- Un *bit*, llamado *mode bit*, es agregado al hardware para indicar el modo actual.

Modo Dual

- La ejecución de instrucciones privilegiadas en el modo monitor garantiza que los procesos, a nivel de usuario, no accedan directamente a los dispositivos de E/S.
- El acceso a un dispositivo se realiza a través de los servicios que brinda el sistema operativo (*syscall*).
- La solicitud de un servicio al sistema operativo es tratado como una interrupción a nivel de software (*trap*), y en ese momento el sistema pasa de modo usuario a modo monitor.
- En Intel la instrucción `int $0x80` genera el cambio de modo.
- Posteriormente, se ejecuta el *handler* de la excepción 0x80 (128 decimal).

- Esquema gráfico del cambio de modo:



Protección de E/S

- Es necesario restringir que los procesos a nivel de usuario no accedan directamente a los dispositivos, sino que deban hacerlo a través del sistema operativo.

- Por eso, se define que todas las instrucciones de E/S son privilegiadas.

- De esa forma, se asegura que un programa a nivel de usuario nunca pueda lograr cambiar el modo a monitor.

- Un usuario podría ingresar una nueva interrupción, modificar una ya existente o cambiar el vector de interrupción y luego generar un *trap* (interrupción por software) para que ejecute.

Protección de memoria

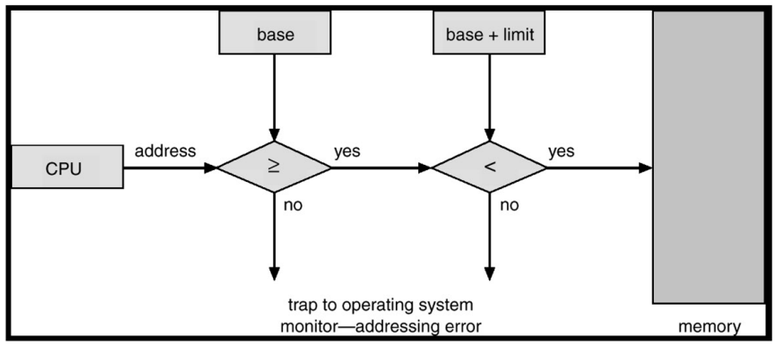
- Es necesario proteger la memoria del núcleo (p.ej.: el vector de interrupciones) y, a su vez, proteger el acceso de memoria entre los distintos procesos (un proceso no debería acceder a la memoria de otro).

- El sistema debe lograr saber si cada dirección generada por un proceso es válida.

- Una forma es utilizar dos registros:

  - Base: Contiene la dirección de memoria física más baja que puede acceder.
  - Límite: Contiene el tamaño del bloque de memoria a partir del registro base.

- Esquema gráfico de la protección a través de registro base y límite:



- Cada dirección física generada por la CPU es controlada para comprobar si es una dirección válida.

- En caso de un acceso inválido se genera un *trap* al sistema operativo.

- La unidad que convierte direcciones lógicas a físicas es la MMU (*Memory Management Unit*), y es la que controla el acceso a memoria. Esta es un dispositivo de *hardware*.

- La unidad MMU únicamente debe ser administrada en modo monitor. Por ejemplo, cargar los registros base y límite.

Protección de CPU

- Una vez que a un proceso se le asigna un recurso procesador puede entrar en una iteración infinita (*infinite loop*) y no retornar nunca más el control al sistema.

- Deben existir mecanismos de protección de uso del procesador.

- Una alternativa es la utilización de un *timer* que interrumpa el procesador cada cierto tiempo (*watch dog timer*).

- El sistema operativo al asignar la CPU carga un contador. Cada vez que la interrupción de *timer* se genera se ejecuta la rutina de atención correspondiente.

- En la rutina de atención de la interrupción el contador es disminuido. Si alcanza al valor 0 se le quita el recurso procesador al proceso y se invoca al planificador para que seleccione otro.

- La instrucción que permite cargar el contador debe ser privilegiada.

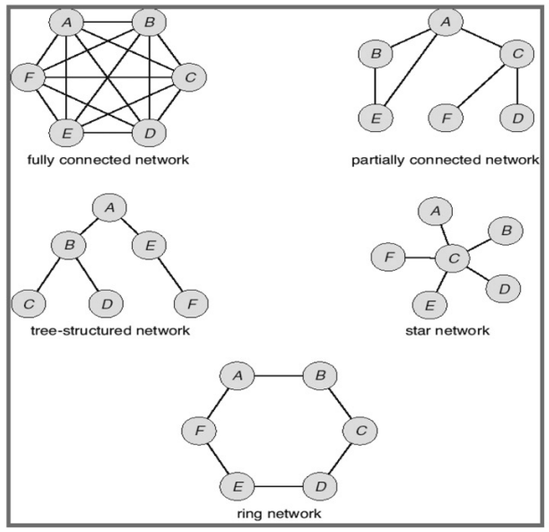
Red

- Las redes se pueden clasificar, básicamente, en dos tipos:
  - Red LAN (*Local Area Network*):
    - Las redes LAN son pequeñas y su alcance está limitado, por lo general a no más de un edificio.
    - Velocidades de 10, 100, 1000 Mbits/s, o más.
  - Red WAN (*Wide Area Network*):
    - Las redes WAN son redes distribuidas sobre una región grande.
    - 1,5 a 100 Mbits/s.

- La diferencia principal es cómo están geográficamente distribuidas.

Topologías de red

- Esquema gráfico de algunas topologías más comunes:



Estructura de los sistemas operativos

Agenda

- Componentes de un sistema operativo.
- Servicios del sistema operativo (*system services*).
- Llamados a sistema (*system calls*).
- Estructura del sistema.

Componentes del sistema operativo

- Por su complejidad un sistema operativo debe ser, en su diseño, modularizado en varios componentes:

  - Administración de procesos.
  - Administración de memoria.
  - Subsistema de Entrada/Salida.
  - Administración de almacenamiento secundario.
  - Subsistema de archivos.
  - Subsistema de red.
  - Sistema de protección.

Administración de procesos

Proceso:

Un programa en la memoria + CPU + acceso a dispositivos + recursos, constituyen un proceso.

Un programa es una entidad pasiva, mientras que un proceso es una entidad activa.

Cada proceso cuenta con un contador de programa (*PC program counter*) que determina la próxima instrucción de código a ejecutar.

El proceso necesita de ciertos recursos (CPU, memoria, archivos y dispositivos de E/S) para realizar su tarea.

Administración de procesos

- El sistema albergará muchos procesos compitiendo por los recursos y será el responsable de proveer de medios o servicios para que realicen su tarea:
  - Crear y destruir procesos.
  - Suspensión y reanudación de procesos.
  - Proveer mecanismos para la cooperación (sincronización) y comunicación entre los procesos.
  - Proveer mecanismos para prever la generación de *dead-locks* o lograr salir de ellos.

Administración de la memoria

- La memoria principal es un arreglo de palabras o *bytes*.
- Es un repositorio de datos de rápido acceso compartido por los CPUs y los dispositivos.
- La memoria es un área de almacenamiento común a los procesadores y dispositivos del sistema donde se almacenan programas, para su ejecución, y datos.
- El vincular programas a direcciones absolutas es fuertemente dependiente del hardware, igual que la posibilidad de reubicación.
- El sistema deberá administrar el lugar libre y ocupado, decidir qué proceso podrá comenzar cuando es cargado en memoria.

Administración de memoria

- Para lograr la multiprogramación es necesario mantener varios programas en memoria al mismo tiempo.
- Existen varios esquemas para la administración de la memoria y requieren distinto soporte del hardware.
- El sistema operativo es responsable de las siguientes tareas:
  - Mantener qué partes de la memoria están siendo utilizadas y por quién.
  - Decidir cuáles procesos serán cargados a memoria cuando exista espacio de memoria disponible.
  - Asignar y quitar espacio de memoria según sea necesario.

Subsistema de Entrada/Salida

- El sistema operativo deberá encapsular y ocultar las características específicas de los diferentes dispositivos de almacenamiento y ofrecer servicios comunes para todos los medios de almacenamiento.
- Para ello proveerá de:
  - Un conjunto de servicios que provean la interfase con el subsistema e implementen técnicas de *cache, buffering* y *spooling*.
  - Una interfase cliente con el sistema operativo para los manejadores de dispositivos o *device drivers* que permitirá interactuar (mediante cargas dinámicas) con cualquier modelo de dispositivo.
  - *Device drivers* específicos.
  - Montaje y desmontaje (*Mount/Dismount*) de dispositivo.

Administración de almacenamiento secundario

- La memoria principal es volátil y demasiado pequeña para guardar todos los datos y programas que son necesarios para el funcionamiento del sistema.
- La mayoría de los sistemas actuales utilizan discos como principal medio para guardar toda la información.
- El sistema operativo es responsable de las siguientes actividades en administración de almacenamiento secundario:
  - Administrar el espacio libre.
  - Asignación del lugar de la información.
  - Algoritmos de planificación de disco.

Subsistema de archivos

- Proporciona una vista uniforme de todas las formas de almacenamiento en los diferentes dispositivos, implementando el concepto de archivo como una colección arbitraria de *bytes* u otras clases u organizaciones más sofisticadas, aunque habitualmente obsoletas.
- Implementará los métodos de:
  - Abrir, Cerrar, Extender.
  - Leer, Escribir.

Red

- En un sistema distribuido (no se comparten físicamente memoria ni dispositivos) los conjuntos de procesos interactúan a través de un canal de comunicación en el contexto de una red de comunicación.

- Comúnmente se generaliza el concepto de dispositivo virtual implementando un manejador (*driver*) que encapsula el acceso a dispositivos remotos.

Sistema de protección

- En un sistema multiusuario donde se ejecutan procesos en forma concurrente se deben tomar medidas que garanticen la ausencia de interferencia entre ellos.

- Por protección nos referimos a los mecanismos por los que se controla al acceso de los procesos a los recursos.

- El mecanismo debe incorporar la posibilidad de definir reglas de acceso y asegurar su verificación en toda ocasión que corresponda.

Servicios del SO

- El sistema brindará un entorno de ejecución de programas donde se dispondrá de un conjunto de servicios que serán accesible mediante una interfase bien definida.

- Servicios básicos que debe brindar un sistema operativo:

  - Ejecución de programas.
  - Operaciones de Entrada/Salida.
  - Manipulación de sistemas de archivos.
  - Comunicación entre procesos.
  - Manipulación de errores (excepciones).

Comunicación entre procesos

- Los procesos deberán poder comunicarse.

- Se deberá proveer mecanismos de comunicación entre ellos, ya sea que estén en el mismo computador (a través de memoria compartida) o en diferentes computadores (a través de transferencias de paquetes de red entre los sistemas operativos involucrados).

Detección de errores

- El sistema deberá tomar decisiones adecuadas ante eventuales errores que ocurran:

  - Fallo en un dispositivo de memoria.
  - Fallo en la fuente de energía.
  - Fallo en un programa.
  - Etc.

Servicios del SO

- Otros servicios de propósito general que deberá brindar el sistema operativo son:

  - Asignación de recursos.
  - Contabilización.
  - Protección.

- Una vez que están definidos los servicios que brindará el sistema operativo se puede empezar a desarrollar la estructura del sistema.

Llamados al sistema

- Los llamados al sistema (*system calls*) son una intefaz, provista por el núcleo, para que los procesos de usuarios accedan a los diferentes servicios que brinda el sistema operativo.

- Al principio los *system calls* estaban desarrollados en lenguaje de la arquitectura de la máquina.

- En los sistemas modernos están programados en lenguajes de programación de alto nivel como C o C++. De esta forma, los programas de usuario tienen un acceso más directo a los servicios.

- Los servicios son invocados por los procesos en modo usuario, cuando ejecutan lo hacen en modo monitor y, al retornar, vuelven al modo usuario.

- Típicamente, a los *system call* se les asocia un número que los identifica.

La llamada a *system call* incluye las siguientes tareas:

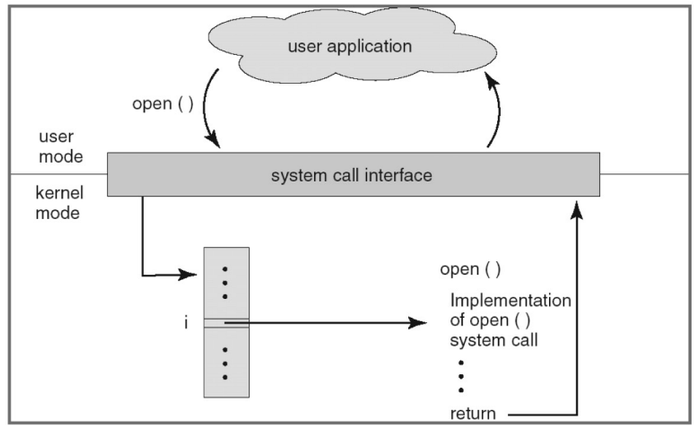
  - Cargar los parámetros en el lugar adecuado (*Stack*).
  - Cargar el número de *system call* en algún registro específico (Ej: *eax* en Intel).
  - Invocar a la interrupción por software (*trap*) adecuada (*system call handler*).
  - La interrupción cambia el *bit* de modo a monitor, controla que el número de *system call* pasado en el registro sea menor que el mayor del sistema y, finalmente, invoca al *system call* correspondiente.
  - El valor retornado por el *system call* es puesto en un registro específico (Ej.: *eax* en Intel).

- Existen tres formas de pasar los parámetros al sistema operativo:

  - A través de los registros: Se utilizan un conjunto de registros para pasar los parámetros. Tiene el problema de la cantidad de parámetros fija y que restringe el tamaño del valor.

    En Intel se utilizan cinco registros: *ebx*, *ecx*, *edx*, *esi*, y *edi*.
  - Un bloque de memoria apuntado a través de un registro.
  - En el *stack* del proceso que realiza el llamado. El proceso guarda los parámetros con operaciones *push* sobre el *stack* y el sistema operativo los saca con la operación *pop*.

Llamados al sistema



- Los *system calls* se clasifican en distintos tipos:
  - Control de procesos
    - Cargar, ejecutar, finalizar, abortar, conseguir atributos, cargar atributos, esperar por tiempo, esperar por un evento o señal, conseguir o liberar memoria, etc.
  - Gestión de archivos
    - Crear, borrar, abrir, cerrar, leer, escribir, conseguir o cargar atributos, etc.
  - Gestión de dispositivos
    - Requerir o liberar un dispositivo, leer o escribir, conseguir o cargar atributos de un dispositivo, etc.
  - Gestión de información
    - Conseguir o cargar la hora del sistema, datos del sistema, de procesos, etc.
  - Comunicaciones
    - Crear o destruir conexiones, enviar o recibir mensajes, etc.

Estructura del sistema

- La estructura interna de los sistemas operativos pueden ser muy diferentes.

- Se deben tener en cuenta:

  - Metas de los usuarios: ser amigable, intuitivo, confiable, seguro, rápido, etc.
  - Metas del sistema: fácil de diseñar, implementar y mantener, también flexible, confiable y eficiente.

- Diseño del sistema:

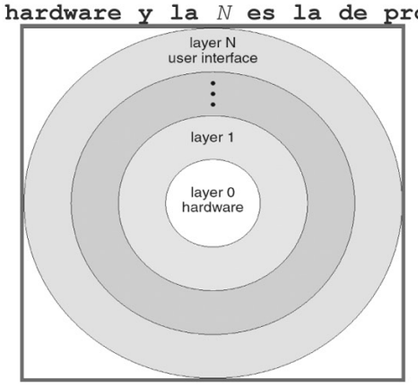
  - Sistema Monolítico.
  - Sistema en capas.
  - Sistema con micronúcleo (*microkernel*).

Sistema Monolítico

- No se tiene una estructura definida.
- El sistema es escrito como una colección de procedimientos que pueden ser invocados por cualquier otro.
- No existe "ocultación de información", ya que cualquier procedimiento puede invocar a otro.
- Si bien todo procedimiento es público y accesible a cualquiera, es posible tener buenos diseños y lograr, de esa forma, buena eficiencia en el sistema.
- Ej.: MS-DOS.
  - Los componentes pueden invocar procedimientos de cualquiera.
- Ej.: Linux
  - Linux es un núcleo monolítico que a logrado un buen diseño orientado a objetos (sistema modular).

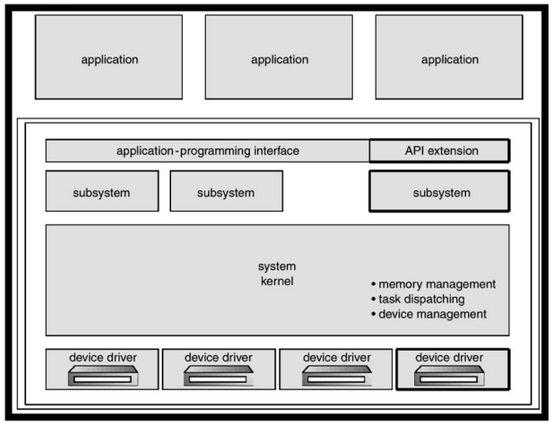
Sistema en capas

- Se organiza el diseño en una jerarquía de capas construidas una encima de la otra.
- Los servicios que brinda cada capa son expuestos en una interfase pública y son consumidos solamente por los de la capa de arriba.
- La capa *0* es el hardware y la *N* es la de procesos de usuario.



Sistema en capas

- Ventajas:
  - Modularidad.
  - Depuración y verificación de cada capa por separado.
- Desventajas:
  - Alto costo de definición de cada capa en la etapa de diseño.
  - Menos eficiente frente al sistema monolítico, ya que sufre de *overhead* al pasar por cada capa.
- Ej.: en capas – OS/2.



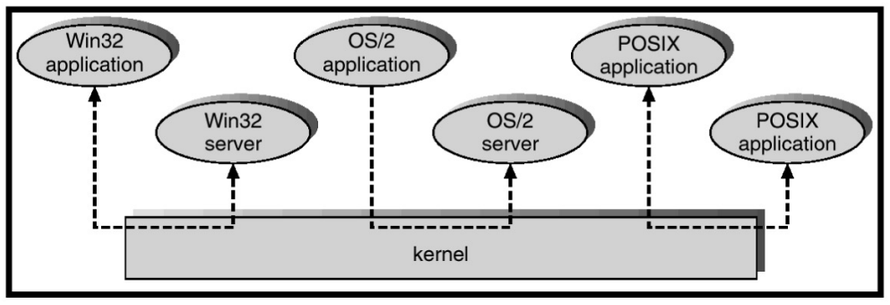
Sistema con micronúcleo (*microkernel*)

- Se constituye de un núcleo que brinde un manejo mínimo de procesos, memoria y, además, provea de una capa de comunicación entre procesos.
- La capa de comunicación es la funcionalidad principal del sistema.
- Los restantes servicios del sistema son construidos como procesos separados al micronúcleo que ejecutan en modo usuario.
- El acceso los servicios del sistema se realiza a través de pasaje de mensajes.

Sistema con micronúcleo

- Ventajas:
  - Aumenta la portabilidad y escalabilidad ya que encapsula las características físicas del sistema.
  - Para incorporar un nuevo servicio no es necesario modificar el núcleo.
  - Es más seguro ya que los servicios corren en modo usuario.
  - El diseño simple y funcional típicamente resulta en un sistema más confiable.

Ej.: Windows.



Introducción a procesos

Agenda

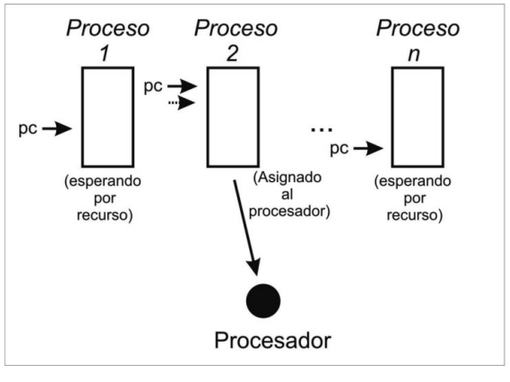
- Proceso.
  - Definición de proceso.
  - Contador de programa.
  - Memoria de los procesos.
- Estados de los procesos.
  - Transiciones entre los estados.
- Bloque descriptor de proceso (*PCB*).
- Creación de procesos.
- Listas y colas de procesos.

Definición de Proceso

- El principal concepto en cualquier sistema operativo es el de proceso.
- Un proceso es un programa en ejecución, incluyendo el valor del *program counter*, los registros y las variables.
- Conceptualmente, cada proceso tiene un hilo (*thread*) de ejecución que es visto como un CPU virtual.
- El recurso procesador es alternado entre los diferentes procesos que existan en el sistema, dando la idea de que ejecutan en paralelo (multiprogramación).

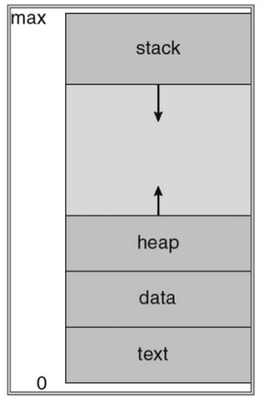
Contador de programa

Cada proceso tiene su *program counter* y avanza cuando el proceso tiene asignado el recurso procesador. A su vez, a cada proceso se le asigna un número que lo identifica entre los demás: identificador de proceso (*process id*).



Memoria de los procesos

Un proceso en memoria se constituye de varias secciones:



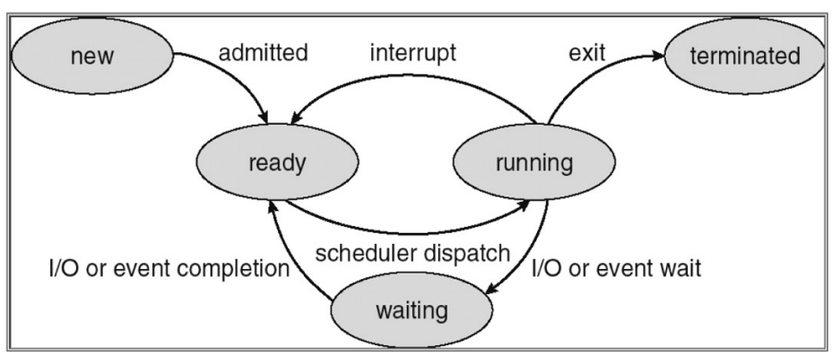
- Código (*text*): Instrucciones del proceso.
- Datos (*data*): Variables globales del proceso.
- Memoria dinámica (*Heap*): Memoria dinámica que genera el proceso.
- Pila (*Stack*): Utilizado para preservar el estado en la invocación anidada de procedimientos y funciones.

El estado de un proceso es definido por la actividad corriente en que se encuentra.

Los estados de un proceso son:

- Nuevo (*new*): Cuando el proceso es creado.
- Ejecutando (*running*): El proceso tiene asignado un procesador y está ejecutando sus instrucciones.
- Bloqueado (*waiting*): El proceso está esperando por un evento (que se complete un pedido de E/S o una señal).
- Listo (*ready*): El proceso está listo para ejecutar, solo necesita del recurso procesador.
- Finalizado (*terminated*): El proceso finalizó su ejecución.

- Diagrama de estados y transiciones de los procesos.



Transiciones entre estados

- Nuevo -> Listo
  - Al crearse un proceso pasa inmediatamente al estado listo.

- Listo -> Ejecutando
  - En el estado de listo el proceso solo espera para que se le asigne un procesador para ejecutar (tener en cuenta que puede existir más de un procesador en el sistema). Al liberarse un procesador el planificador (*scheduler*) selecciona el próximo proceso, según algún criterio definido, a ejecutar.

- Ejecutando -> Listo
  - Ante una interrupción que se genere, el proceso puede perder el recurso procesador y pasar al estado de listo. El planificador será el encargado de seleccionar el próximo proceso a ejecutar.

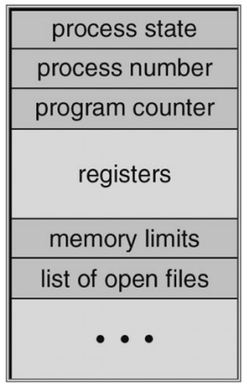
- Ejecutando -> Bloqueado
  - A medida que el proceso ejecuta instrucciones realiza pedidos en distintos componentes (ej.: genera un pedido de E/S). Teniendo en cuenta que el pedido puede demorar y, además, si está en un sistema multiprogramado, el proceso es puesto en una cola de espera hasta que se complete su pedido. De esta forma, se logra utilizar en forma más eficiente el procesador.

- Bloqueado -> Listo
  - Una vez que ocurre el evento que el proceso estaba esperando en la cola de espera, el proceso es puesto nuevamente en la cola de procesos listos.

- Ejecutando -> Terminado
  - Cuando el proceso ejecuta su última instrucción pasa al estado terminado. El sistema libera las estructuras que representan al proceso.

Bloque descriptor de proceso

- El proceso es representado, a nivel del sistema operativo, a través del *bloque descriptor de proceso* (*Process Control Block*).



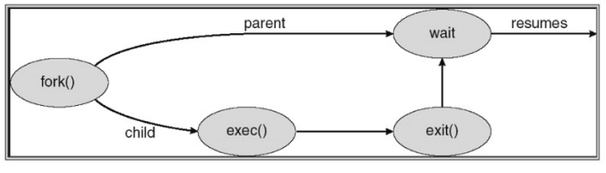
- Los procesos de un sistema son creados a partir de otro proceso.

- Al creador se le denomina padre y al nuevo proceso hijo. Esto genera una jerarquía de procesos en el sistema.

- En el diseño del sistema operativo se debe decidir, en el momento de creación de un nuevo proceso, cuales recursos compartirán el proceso padre e hijo. Las opciones son que compartan todo, algo o nada.

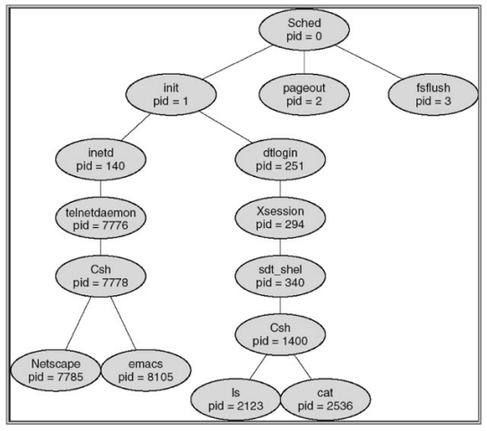
- Una vez creado el nuevo proceso tendrán un hilo (*pc*) de ejecución propio. El sistema genera un nuevo *PCB* para el proceso creado.

- Ej.: UNIX
  - UNIX provee el *system call* *fork* para la creación de un nuevo proceso.
  - La invocación a esta función le retorna al padre el número de *process id* del hijo recién creado y al hijo el valor 0. El hijo comienza su ejecución en el retorno del *fork*.
  - Además, se provee del *system call* *exec* que reemplaza el espacio de memoria del proceso por uno nuevo.



Creación de procesos

```
int main() {
      pid_t  pid;

      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            exit(-1);
      }
      if (pid == 0) /* child process */
            execlp("/bin/ls", "ls", NULL);
      else {  /* parent */
            wait (NULL);
            printf ("Child Complete");
            exit(0);
      }
}
```



Listas y colas de procesos

Los procesos, en los distintos estados que tienen, son agrupados en listas o colas:

- Lista de procesos del sistema (*job queue*): En esta lista están todos los procesos del sistema. Al crearse un nuevo proceso se agrega el PCB a esta lista. Cuando el proceso termina su ejecución es borrado.
- Cola de procesos listos (*ready queue*): Esta cola se compondrá de los procesos que estén en estado listo. La estructura de esta cola dependerá de la estrategia de planificación utilizada.
- Cola de espera de dispositivos (*device queue*): Los procesos que esperan por un dispositivo de E/S particular son agrupados en una lista específica al dispositivo. Cada dispositivo de E/S tendrá su cola de espera.

Procesos

Agenda

- Cambio de contexto (*context switch*).
- Hilos (*Threads*).
  - *Threads* a nivel de usuario.
  - *Threads* a nivel de núcleo del sistema.
  - Modelos de *threads*.

Definición de Proceso

- El principal concepto en cualquier sistema operativo es el de proceso.

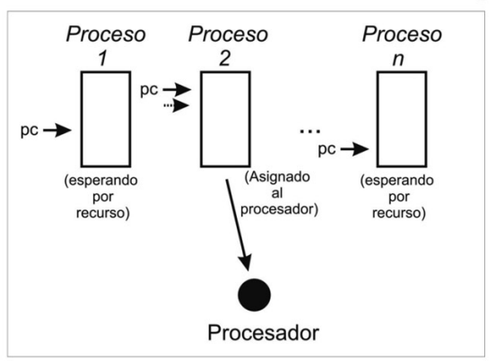
- Un proceso es un programa en ejecución, incluyendo el valor del *program counter*, los registros y las variables.

- Conceptualmente, cada proceso tiene un hilo (*thread*) de ejecución que es visto como un CPU virtual.

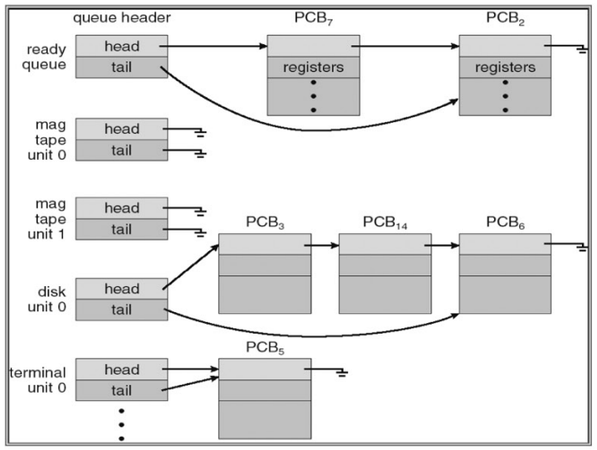
- El recurso procesador es alternado entre los diferentes procesos que existan en el sistema, dando la idea de que ejecutan en paralelo (multiprogramación).

Contador de programa

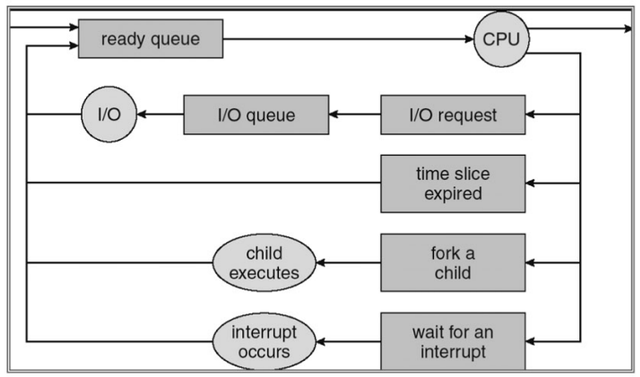
Cada proceso tiene su *program counter*, y avanza cuando el proceso tiene asignado el recurso procesador. A su vez, a cada proceso se le asigna un número que lo identifica entre los demás: identificador de proceso (*process id*).



Listas y colas de procesos



- Diagrama de transición de un proceso entre las colas del sistema.



Cooperación entre procesos

- Procesos concurrentes pueden ejecutar en un entorno aislado (se debe asegurar la ausencia de interferencias) o, eventualmente, podrán interactuar cooperando en pos de un objetivo común compartiendo objetos comunes.

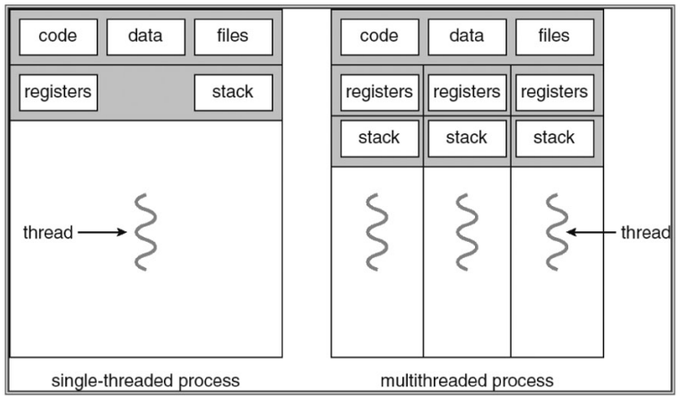
- Es necesario que el sistema operativo brinde unas herramientas específicas para la comunicación y sincronización entre los procesos (*Inter Process Communication – IPC*).

- IPC es una herramienta que permite a los procesos comunicarse y sincronizarse sin compartir el espacio de direccionamiento en memoria.

Cooperación entre procesos

- Hay aplicaciones donde es necesario utilizar y compartir recursos en forma concurrente.
- IPC brindan una alternativa a nivel de sistema operativo.
- Los sistemas operativos modernos están proporcionando servicios para crear más de un hilo (*thread*) de ejecución (control) en un proceso.
- Con las nuevas tecnologías multi-core esto se hace algo necesario para poder sacar mayor provecho al recurso de procesamiento.
- De esta forma, se tiene más de un hilo de ejecución en el mismo espacio de direccionamiento.
- Cada *thread* contendrá su propio *program counter*, un conjunto de registros, un espacio para el *stack* y su prioridad.

- Todos los recursos, sección de código y datos son compartidos por los distintos *threads* de un mismo proceso.



Ventajas del uso de *threads*

- Repuesta: **Desarrollar una aplicación con varios hilos de control (*threads*) permite tener un mejor tiempo de respuesta.**
- Compartir recursos: **Los *threads* de un proceso comparten la memoria y los recursos que utilizan. A diferencia de *IPC*, no es necesario acceder al *kernel* para comunicar o sincronizar los hilos de ejecución.**
- Economía: **Es más fácil un cambio de contexto entre *threads* ya que no es necesario cambiar el espacio de direccionamiento. A su vez, es más "liviano" para el sistema operativo crear un *thread* que crear un proceso nuevo.**
- Utilización de arquitecturas con multiprocesadores: **Disponer de una arquitectura con más de un procesador permite que los *threads* de un mismo proceso ejecuten en forma paralela.**

Threads

- **Los *threads* pueden ser implementados tanto a nivel de usuario como a nivel de sistemas operativo:**

  - Hilos a nivel de usuario (*user threads*): **Son implementados en alguna librería de usuario. La librería deberá proveer soporte para** crear, planificar y administrar **los *threads* sin soporte del sistema operativo. El sistema operativo solo reconoce un hilo de ejecución en el proceso.**

  - Hilos a nivel del núcleo (*kernel threads*): **El sistema es quien provee la creación, planificación y administración de los *threads*. El sistema reconoce tantos hilos de ejecución como *threads* se hayan creado.**

Ventajas de *user threads* sobre *kernel threads*:

- Desarrollo de aplicaciones en sistemas sin soporte a hilo: **Se pueden aprovechar todos los beneficios de programar orientado utilizando *threads*. Además se puede portar la aplicación a un sistema operativo que carezca de la noción de varios hilos de ejecución.**

- Cambio de contexto: **El cambio de contexto entre *threads* de usuario es más simple ya que no consume el *overhead* que tendría en el sistema operativo (guardar registros).**

- Planificación independiente: **Se puede crear una nueva estrategia de planificación diferente a la que tenga el sistema operativo.**

Threads

Ventajas de *kernel threads* sobre *user threads*:

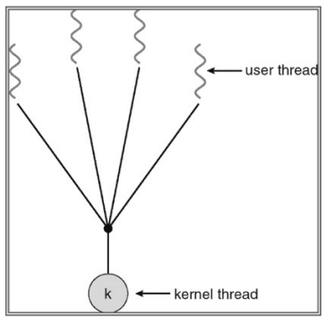
- Mejor aprovechamiento de un sistema multiprocesador: **el sistema operativo puede asignar *threads* del mismo proceso en distintos procesadores. De esta forma, un proceso puede estar consumiendo más de un recurso procesador a la vez.**
- Ejecución independiente: **Al ser independientes los hilos de ejecución, si un *thread* se bloquea (debido a p.ej. una operación de E/S) los demás *threads* pueden seguir ejecutando.**

- **La mayoría de los sistemas proveen *threads* tanto a nivel de usuario como de sistema operativo.**

- **De esta forma surgen varios modelos:**
  - Mx1 (*Many-To-One*): **Varios *threads* de a nivel de usuario a un único *thread* a nivel de sistema.**
  - 1x1 (*one-to-One*): **Cada *threads* de usuario se corresponde con un *thread* a nivel del núcleo (*kernel thread*).**
  - MxN (*Many-To-Many*): **Varios *threads* a nivel de usuario se corresponde con varios *threads* a nivel del núcleo.**

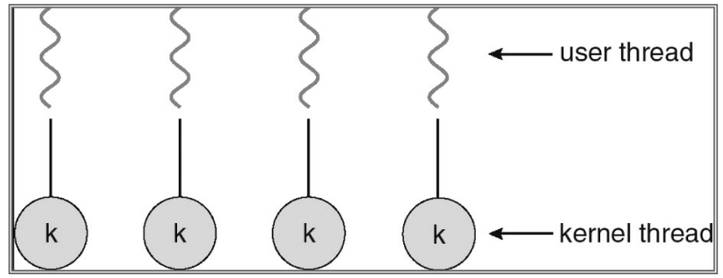
Mx1 (*Many-To-One*)

- **Este caso se corresponde al de tener los *threads* implementados a nivel de usuario.**
- **El sistema solo reconoce un *thread* de control para el proceso.**
- **Los *threads* de usuario ejecutarán cuando estén asignados al *kernel thread* del proceso (tarea llevada a cabo por el planificador a nivel de usuario) y, además, a este le asigne la CPU el planificador del sistema operativo.**



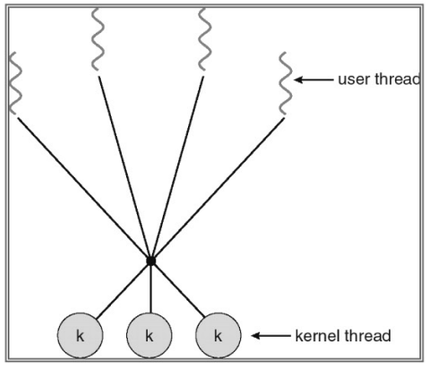
1x1 (*One-To-One*)

- **Cada *thread* que es creado a nivel de usuario se genera un nuevo *thread* a nivel de sistema que estará asociado mientras exista.**
- **El sistema reconoce todos los *threads* a nivel de usuario y son planificados independientemente. En este caso, no hay planificador a nivel de usuario.**



MxN (*Many-To-Many*)

- **Cada proceso tiene asignado un conjunto de *kernel threads* independiente de los *threads* a nivel de usuario que el proceso haya creado.**
- **El planificador a nivel de usuario asigna los *threads* en los *kernel threads*.**
- **El planificador de sistema solo reconoce los *kernel threads*.**



Planificación

Agenda

- Introducción.
- Clases de procesos.
- Esquemas de planificación.
- Despachador.
- Criterios de planificación.
- Algoritmos de planificación.
  - FCFS.
  - SJF.
  - Prioridad.
  - Round-Robin.
  - Multilevel-Queue.
  - Multilevel-Feedback-Queue.
  - Sistemas multiprocesadores.

Introducción

- La planificación (*scheduling*) es la base para lograr la multiprogramación.
- Un sistema multiprogramado tendrá varios procesos que requerirán el recurso procesador a la vez.
- Esto sucede cuando los procesos están en estado *ready* (pronto).
- Si existe un procesador disponible y existen procesos en estado *ready*, se debe elegir el que será asignado al recurso para ejecutar.
- El componente del sistema operativo que realiza la elección del proceso es llamada planificador (*scheduler*).

Clases de procesos

- Existen distintas políticas de planificación que serán exitosas según la clase de procesos que ejecuten.
- En general, los procesos tienden a ser o más intensivos en el uso de procesador o más intensos en el uso de operaciones de E/S.
- Los procesos tenderán a tener ciclos de ráfagas de ejecución (*CPU-burst*) y ciclos de ráfagas de espera de operaciones de E/S (*I/O burst*):
  - Procesos CPU-bound: Los procesos que contienen un alto uso de procesador son llamados *CPU-bound* o *compute-bound*.
  - Procesos I/O-bound: Los procesos que realizan muchos accesos a operaciones de E/S son llamados *I/O-bound*.
- La prioridad que tenga un proceso frente a los demás para acceder al recurso será inversamente proporcional al uso que haga del recurso.

Esquemas de planificación

- Los momentos en que el planificador es invocado son:
  - Cuando un proceso se bloquea en una operación de E/S o un semáforo, etc.
  - Cuando un proceso cambia del estado *ejecutando* al estado *pronto*. Por ejemplo, al ocurrir una interrupción o se crea un nuevo proceso.
  - Cuando ocurre una interrupción de E/S y un proceso pasa del estado *bloqueado* a *pronto*.
  - Cuando un proceso finaliza su ejecución.
- Cuando ocurre 1 o 4, el planificador es invocado debido a que el proceso en ejecución libera el procesador.
- Si el planificador es invocado cuando ocurre 2 o 3, se dice que este es expropiativo (*preemptive*), ya que puede quitar el procesador al proceso que estaba en ejecución.
- Sistemas operativos con planificadores no expropiadores (*non-preemptive*) son los que asignan el recurso procesador a un proceso y hasta que este no lo libere, ya sea porque finaliza su ejecución o se bloquea, no se vuelve a ejecutar el planificador.
- Sistemas operativos con planificadores expropiativos (*preemptive*) son que los que pueden expropiar el recurso procesador a un proceso cuando otro proceso entra en estado *pronto* (ya sea porque es nuevo o porque se desbloqueó) o porque se le impone una limitante de tiempo para ejecutar.
- Los esquemas de planificación son útiles según el ambiente donde sean aplicados:
  - Sistemas por lotes: Como no existe interacción con usuarios, los planificadores no expropiadores son ideales.
  - Sistemas interactivos: Debido a que existen procesos de usuarios ejecutando a la vez, los planificadores expropiadores son ideales para mantener un buen tiempo de respuesta para los usuarios.
  - Sistemas de tiempo real: No es necesario planificadores expropiadores, ya que los procesos puede que no ejecuten por un buen tiempo, pero cuando lo hacen es por un período muy corto.

Criterios de planificación

- **Los algoritmos de planificación tendrán distintas propiedades y favorecerán cierta clase de procesos.**

- **Es necesario definir criterios para poder evaluar los algoritmos de planificación:**

  - Utilización de CPU (*CPU utilization*): **Es el porcentaje de uso (en cuanto a ejecución de tareas de usuario o del sistema que son consideradas útiles) que tiene un procesador.**

  - Rendimiento (*Throughput*): **Es el número de procesos que ejecutaron completamente por unidad de tiempo (una hora p.ej.).**

  - Tiempo de retorno (*Turnaround time*): **Es el intervalo de tiempo desde que un proceso es cargado hasta que este finaliza su ejecución.**

    - Tiempo de espera (*Waiting time*): **Es la suma de los intervalos de tiempo que un proceso estuvo en la cola de procesos listos (*ready queue*).**

    - Tiempo de respuesta (*Response time*): **Es el intervalo de tiempo desde que un proceso es cargado hasta que brinda su primer respuesta. Es útil en sistemas interactivos.**

First Come First Served (FCFS)

- **Los procesos son ejecutados en el orden que llegan a la cola de procesos listos.**

- **La implementación es fácil a través de una cola FIFO.**

- **Es adecuado para sistemas por lotes (*batch*).**

- **Es un algoritmo no expropiador: una vez que el procesador le es asignado a un proceso este la mantiene hasta que termina o genera un pedido de E/S.**

- **El tiempo de espera promedio por lo general es alto.**

First Come First Served

Proceso	Burst Time
P1	24
P2	3
P3	3

- **Tiempo de espera: *P1* = 0; *P2* = 24; *P3* = 27.**
- **Tiempo de espera promedio: (0 + 24 + 27)/3 = 17.**

- **El algoritmo asocia a los procesos el largo de su próximo *CPU-burst*.**

- **Cuando el procesador queda disponible se le asigna al proceso que tenga el menor *CPU-burst*.**

- **Si dos procesos tienen el mismo *CPU-burst* se desempata de alguna forma.**

- **Su funcionamiento depende de conocer los tiempos de ejecución, que en la mayoría de los casos no se conoce.**

- **Es adecuado para sistemas por lotes (*batch*).**

- **Dos esquemas:**

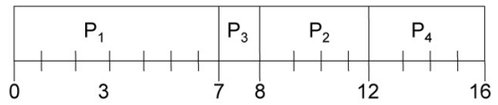
  - No expropiador: **una vez que se le asigna el procesador a un proceso no se le podrá quitar.**
  - Expropiador: **Si un nuevo proceso aparece en la lista de procesos listos con menor *CPU-burst*, se le quita la CPU para asignarla al nuevo proceso.**

- **Este algoritmo es óptimo para el tiempo de espera, pero requiere que todos los procesos participantes estén al comienzo (además de saber el tiempo del próximo *CPU-burst*).**

Shortest Job First (SJF) – No expropiativo

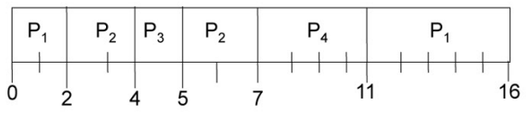
Proceso	Tiempo de arribo	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Tiempo de espera promedio: (0 + 6 + 3 + 7)/4 = 4.



Proceso	Tiempo de arribo	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Tiempo de espera promedio: (9 + 1 + 0 +2)/4 = 3.



Basados en Prioridad

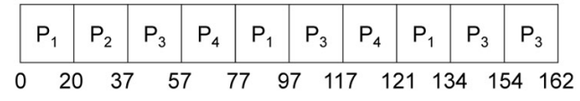
- **A cada proceso se le asigna un número entero que representa su prioridad.**
- **El planificador asigna el procesador al proceso con la más alta prioridad.**
- **Se utiliza en general un esquema expropiador ya que, si un proceso con mayor prioridad que el que esta ejecutando arriba a la lista de procesos listos (*ready queue*), será asignado al procesador.**
- ***SJF* se puede ver como un algoritmo de prioridad donde la prioridad esta dada por el próximo *CPU-burst*.**
- **Es adecuado para sistemas interactivos.**

Basados en Prioridad

- **Sufre de posposición-indefinida ya que un proceso de baja prioridad quizás no pueda ejecutar nunca.**
- **La solución es utilizar prioridades dinámicas de envejecimiento: incrementar la prioridad según pasa el tiempo sin ejecutar.**
- **La prioridad de un proceso para el uso del recurso procesador deberá ser inversamente proporcional al uso que el proceso haga del mismo.**
- **Por lo tanto,un proceso tipo *I/O-bound* deberá tener, en general, mayor prioridad que uno tipo *CPU-bound*.**
- **A cada proceso se le brinda un intervalo de tiempo para el uso del procesador (*time quantum*).**
- **Al finalizar el tiempo, el procesador le es expropiado y vuelve al estado *pronto* (*ready*) al final de la cola.**
- **Es fácil de implementar ya que solamente es necesario una cola de procesos listos. Cuando un proceso consume su *quantum* es puesto al final de la cola.**
- **El *quantum* debe ser bastante mayor a lo que lleva realizar un cambio de contexto, sino se tendrá mucho *overhead*. A su vez, el tiempo de *quantum* incide en los tiempos de retorno.**
- **Es ideal para sistemas de tiempo compartido.**

Proceso	Burst Time
P1	53
P2	17
P3	68
P3	24

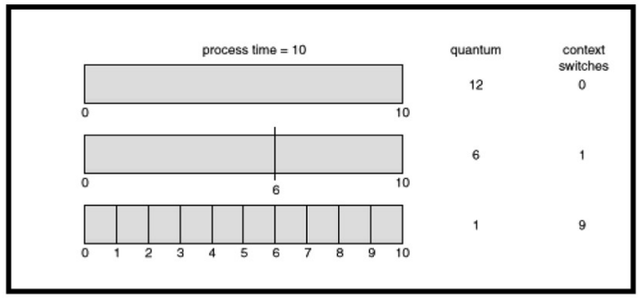
quantum = 20



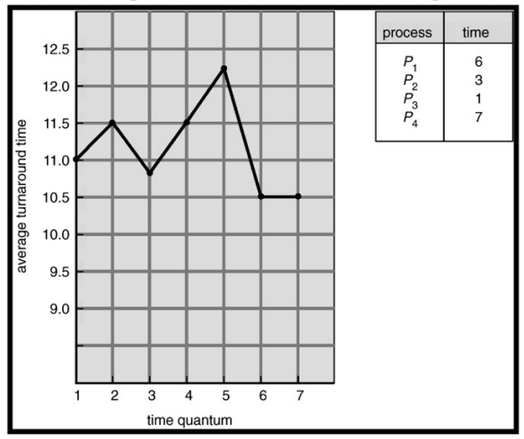
- **Por lo general, tiene un mayor tiempo de retorno que el *SJF*, pero mejora el tiempo de respuesta.**

Round Robin (RR)

- Es necesario asignar un ajustado tiempo de *quantum:*
  - Si es muy chico generará muchos cambios de contexto.
  - Si es muy grande, el sistema tenderá a un FCFS.



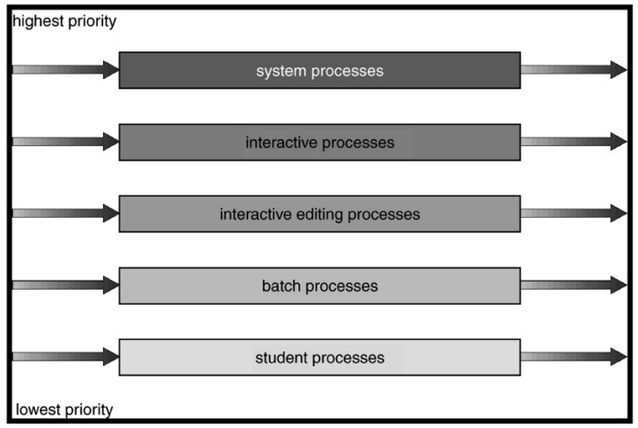
- El promedio del tiempo de retorno varía según el *quantum*.



Multilevel Queue

- Si los procesos se pueden clasificar según sus cualidades, es posible dividir la lista de procesos listos (*ready queue*) en varias colas (una para cada clasificación).
- Los procesos son asignados permanentemente a una de las colas.
- Cada cola tendrá su propio algoritmo de planificación propio.
- Además, se debe tener una estrategia de planificación entre las diferentes colas. Por ejemplo, una cola tendrá prioridad sobre otra.

Multilevel Queue



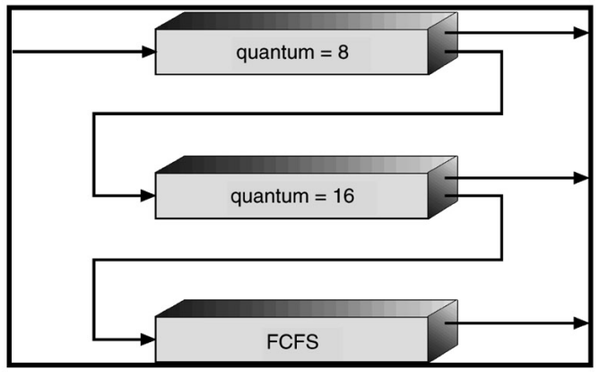
Multilevel Feedback Queue

- Se diferencia con el anterior en que procesos pueden cambiar de cola (nivel).
- Se basa en categorizar los procesos según el uso de CPU (*CPU-burst*) que tengan.
- La cola de mayor prioridad será la de los procesos *I/O-bound* y la de menor la de procesos con alto *CPU-bound*.
- De esta forma, se garantiza que los procesos con poco uso de procesador tengan mayor prioridad y los que consumen mucho procesador tendrán baja prioridad.
- Los procesos, según el consumo de CPU que hagan, serán promovidos a una cola de mayor prioridad o rebajados a una de menor prioridad.

Un planificador Multilevel-Feedback-Queue es definido por:

- El número de colas.
- El algoritmo de planificación para cada cola.
- El método utilizado para promover a un proceso a una cola de mayor prioridad.
- El método utilizado para bajar a un proceso a una cola de menor prioridad.
- El método utilizado para determinar a que cola será asignado un proceso cuando este *pronto*.

Multilevel Feedback Queue



Sistemas multiprocesadores

- En un sistema simétrico cualquier procesador podrá ejecutar procesos de usuario.
- Una posibilidad es asignar una cola de procesos listos para cada procesador y de esa forma mantener los procesos asignados a un procesador (afinidad de procesador).
- Esto es conveniente para aprovechar los datos que están frescos en la memoria cache del procesador, ya que al ejecutar un proceso en un procesador se nutre su cache con datos del proceso.
- De esta forma, se logra mantener un mayor índice de *cache hit* y, por lo tanto, un mayor rendimiento en el sistema.
- Un problema que puede surgir es un desbalance en la cantidad de trabajo por procesador. En estos casos se migrarán procesos de cola para lograr balancear nuevamente la carga.

Despachador

- Una vez que el planificador ejecuta y elige el proceso a asignar al procesador, se invoca al despachador (*dispatcher*) que es el encargado de asignar el proceso al procesador.

- La tarea que realiza es:
  - Cambiar el contexto: Salvar registros del procesador en PCB del proceso saliente. Cargar los registros con los datos del PCB del proceso entrante.
  - Cambiar el *bit* de modo a usuario.
  - Saltar a la instrucción adecuada que había quedado el proceso que se asigno a la CPU (registro *program counter*).

Administración de memoria I

Agenda

- Introducción.
- Conceptos básicos.
  - Preparación de un programa para ejecutar.
  - Áreas de la memoria de un proceso.
  - Asociación de direcciones.
  - Ensamblaje dinámico y bibliotecas compartidas.
  - Asociación dinámica de la memoria a nivel de proceso.
  - Carga dinámica.
- Direccionamiento.
  - Tipos de direccionamiento.
  - Protección de memoria.
- Asignación de memoria.
- Swapping.

Introducción

- La administración de la memoria es una de las más importantes tareas del sistema operativo.
- En los sistemas operativos multiprogramados es necesario mantener varios programas en memoria al mismo tiempo.
- Existen varios esquemas para la administración de la memoria y requieren distinto soporte del hardware.

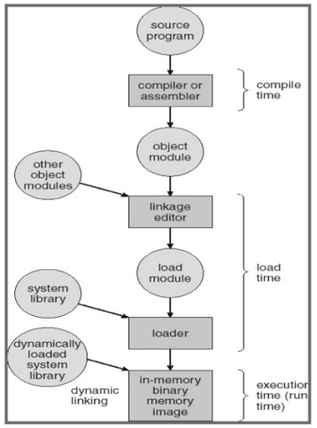
- El sistema operativo es responsable de las siguientes tareas:
  - Mantener qué partes de la memoria están siendo utilizadas y por quién.
  - Decidir cuales procesos serán cargados a memoria cuando exista espacio de memoria disponible.
  - Asignar y quitar espacio de memoria según sea necesario.

Conceptos básicos

- Preparación de un programa para ejecutar.
- Los programas son escritos, por lo general, en lenguajes de alto nivel y deben pasar por distintas etapas antes de ser ejecutados:
  - Compilación (*compile*): Traducción del código fuente del programa a un código objeto.
  - Ensamblaje (*linker*): Ensamblaje de varios códigos objetos en un archivo ejecutable.
  - Carga (*load*): Asignación del archivo ejecutable a la memoria principal del sistema.

- Un programa ejecutable consta de secciones de instrucciones y de datos.
- El *linker* surge ante la necesidad de modularizar y reutilizar código. Se resuelven las referencias externas, así como las posiciones relativas de los símbolos en los diferentes módulos, formando uno consolidado.

Conceptos básicos



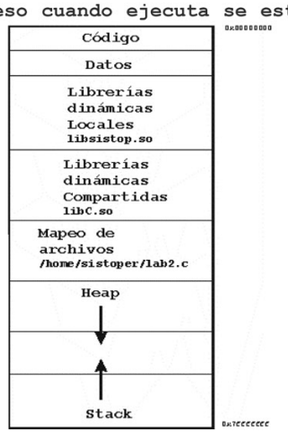
Conceptos básicos

- Cuando un proceso es creado el cargador (loader) del sistema crea en memoria el espacio necesario para las diferentes áreas y la carga con la información.
- El compilador, ensamblador, sistema operativo y librerías dinámicas deben cooperar para administrar la información y realizar la asignación.

  - Compilador: genera un archivo objeto para cada archivo fuente. La información está incompleta, ya que se utilizan informaciones de otros archivos (como llamados a funciones externas).
  - Ensamblador: combina todos los archivos objetos de un programa dentro de un único archivo objeto.
  - Sistemas operativos: Carga los programas en memoria, permite compartir la memoria entre varios procesos y brinda mecanismos a los procesos para obtener más memoria en forma dinámica.
  - Librerías dinámicas: proveen rutinas de asignación dinámica (malloc, free).

Conceptos básicos

- La memoria de un proceso cuando ejecuta se estructura en diferentes áreas:



Asociación de direcciones (*Address binding*)

- La asignación de la ubicación de un programa en memoria principal puede ser realizada en varios tiempos:
  - Tiempo compilación (*compile time*): El programa será asignado a un lugar específico y conocido de la memoria física. La direcciones de memoria son referenciadas en forma absoluta (***static relocation***).
  - Tiempo de carga (*load time*): La asignación del lugar de memoria donde será cargado el programa es hecho al momento de la carga. Las direcciones de memoria deben ser referenciadas en forma relativa (***dynamic relocation***).
  - Tiempo de ejecución (*execution time*): Un programa puede variar su ubicación en memoria física en el transcurso de la ejecución.

Ensamblaje dinámico (*dynamic linking*)

- En la etapa de ensamblaje de un programa las bibliotecas compartidas pueden incorporarse al archivo ejecutable generado (ensamblaje estático – ***static linking***).

  Ej. en Linux: */usr/lib/libc.a*

- Otra alternativa es que las bibliotecas compartidas sean cargadas en tiempo de ejecución (ensamblaje dinámico – ***dynamic linking***).

  Ej. en Linux */lib/libc.so*, **en windows** *system.dll*

- En los archivos ejecutables las bibliotecas estáticas son incorporadas, mientras que para las dinámicas se mantiene una referencia.

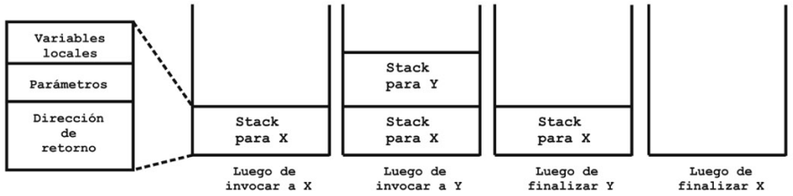
  Ej. en Linux comando `ls`:

```
$ ldd /bin/ls
        librt.so.1 => /lib/librt.so.1 (0x4001c000)
        libc.so.6 => /lib/libc.so.6 (0x40030000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40149000)
        /lib/ld-linux.so.2 (0x40000000)
```

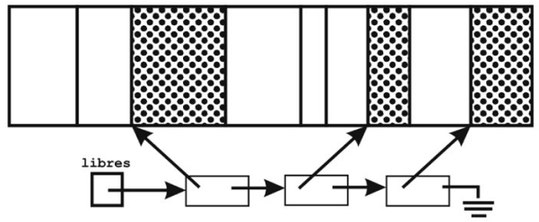
- Esto permite, junto con la carga dinámica, hacer un uso más eficiente de la memoria, ya que las bibliotecas dinámicas se cargan una única vez en memoria principal.

Asignación dinámica a nivel de proceso

- La asignación dinámica en un proceso se da a través de:
  - Asignación en el *Stack*.
  - Asignación en el *Heap*.
- A nivel del *stack* la memoria se comporta en forma más predictiva.



- La asignación en el *heap* no es predictiva como en el caso del *stack*:



- En este caso se genera fragmentación de la memoria.
- Los sistemas operativos optan por delegar la administración de esta memoria a librerías de usuario.

Carga dinámica (*dynamic loading*)

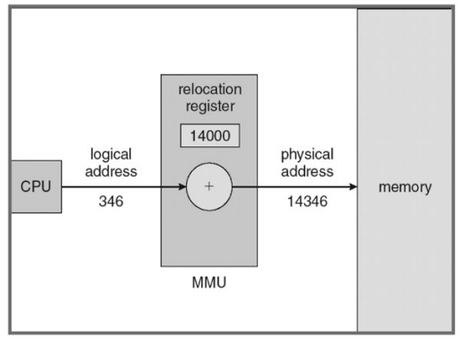
- El tamaño de un proceso en memoria está limitado por la cantidad de memoria física del sistema.
- Con el fin de lograr un mayor aprovechamiento de la memoria se puede utilizar la carga dinámica.
- La carga dinámica dispone que una rutina no es cargada en memoria física hasta que no sea invocada.
- La ventaja de la carga dinámica es que las rutinas que no son utilizadas no son cargadas en memoria física y, por lo tanto, no consumen recursos innecesariamente.

Tipos de direccionamiento

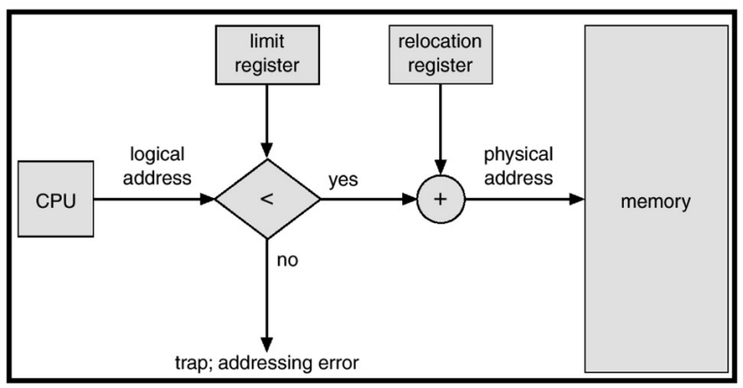
- Se definen varios tipos de direccionamientos:
  - Direccionamiento físico (*physical address*): **La unidad de memoria manipula direcciones físicas.**
  - Direccionamiento virtual (*virtual address*): **Son las direcciones lógicas que se generan cuando existe asociación de direccionamiento en tiempo de ejecución.**
- Para la asociación de direccionamiento en tiempo de compilación o carga, las direcciones lógicas o físicas coinciden. No es así para la asociación en tiempo de ejecución.

Tipos de direccionamiento

- Las traducciones de direcciones lógicas a físicas son hechas por la MMU (*Memory Management Unit*). Los procesos solo manipulan direcciones lógicas y no visualizan las físicas, que solamente son vistas por la MMU.



Protección de memoria



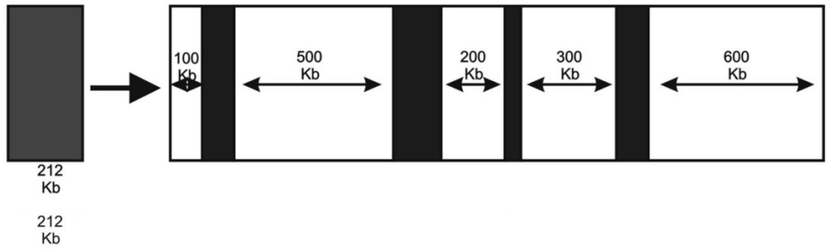
Asignación de memoria a nivel del sistema

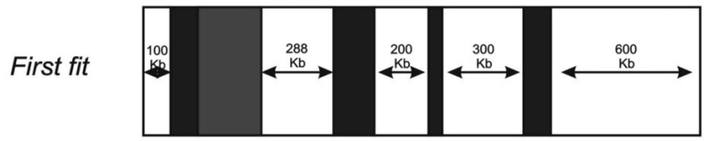
- La memoria, por lo general, es dividida en dos particiones:
  - Sistema operativo residente.
  - Procesos de usuarios.
- Es necesario un mecanismo de protección de memoria entre los procesos entre sí y el sistema operativo.
- El registro de ubicación (*relocation register*) y el registro límite son utilizados para realizar la verificación de accesos válidos a la memoria.
- Toda dirección lógica debe ser menor al valor del registro límite.

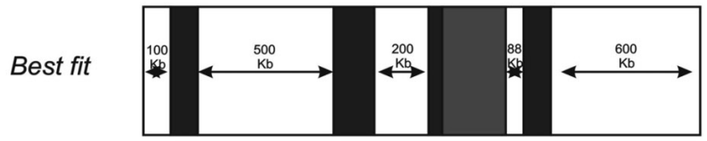
Estructuras para asignación

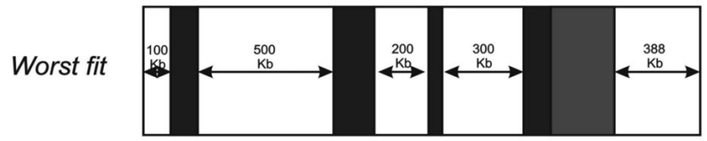
- El sistema operativo debe llevar cuenta de las particiones ocupadas y libres.
- Los métodos más comunes utilizados son a través de:
  - Mapa de bits.
  - Lista encadenada.
  - Diccionarios (*hash*).
- En la asignación de memoria a un proceso existe varias estrategias:
  - *First fit*: Asigna el primer "agujero" de memoria libre que satisface la necesidad.
  - *Best fit*: Asigna el mejor "agujero" de memoria libre que exista en la memoria principal.
  - *Worst fit*: Asigna el requerimiento en el "agujero" más grande que exista en la memoria principal.
- Estudios de simulación han mostrado que *first-fit* y *best-fit* lograron mejores rendimientos en tiempo de asignación y utilización de la memoria que la estrategia *worst-fit*.

Estrategia de asignación









Fragmentación

- Las estrategias de asignación presentadas muestran problemas de fragmentación externa.
- En la memoria van quedando una gran cantidad de "agujeros" chicos que no son asignados. La memoria libre está fragmentada en una gran cantidad "agujeros" chicos.
- La fragmentación externa existe cuando existe suficiente memoria libre en el sistema para satisfacer un requerimiento de memoria, pero no es posible asignarlo debido a que no es contiguo.

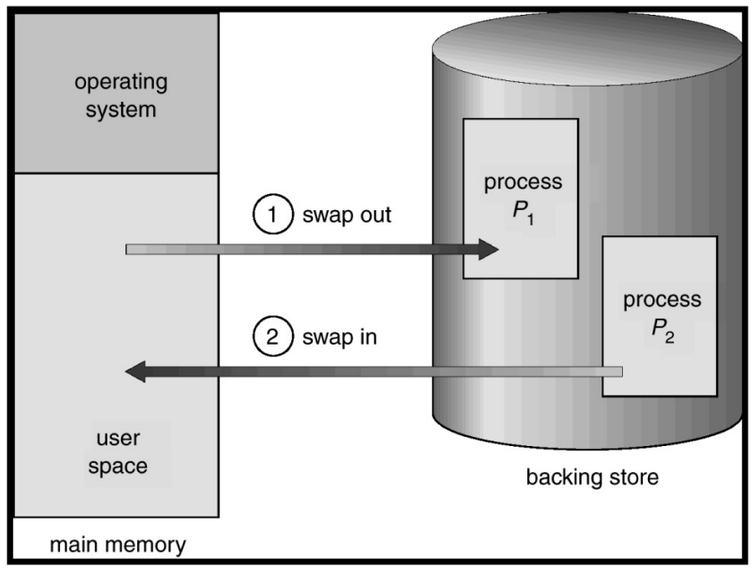
Swapping

- En sistemas multiprogramados más de un proceso está cargado en memoria principal. Para obtener un mayor nivel de multiprogramación, los procesos que no están ejecutando pueden ser llevados a disco temporalmente.

- El disco (*backing store*) es un espacio donde se dispondrán las imágenes de memoria de los procesos.

- Al mecanismo de llevar un proceso desde memoria principal a disco se le denomina *swap-out*. Al inverso se le denomina *swap-in*.

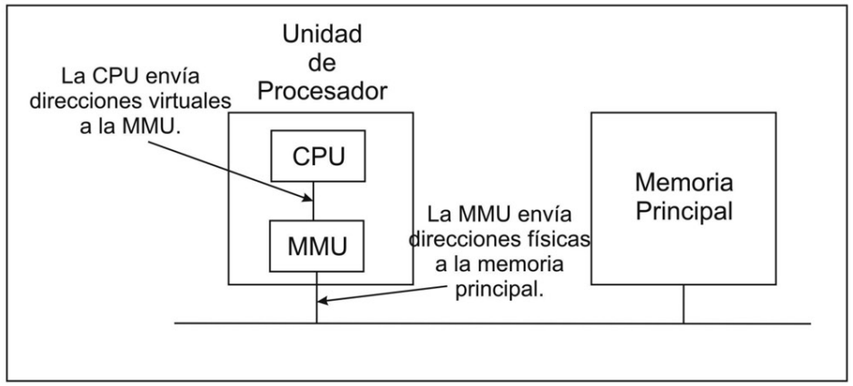
- El mayor tiempo consumido en el *swaping* es el tiempo de transferencia.



- El lugar de memoria donde será asignado un proceso en el momento de *swap-in* depende del método de asociación de direccionamiento (*address binding*) utilizado.

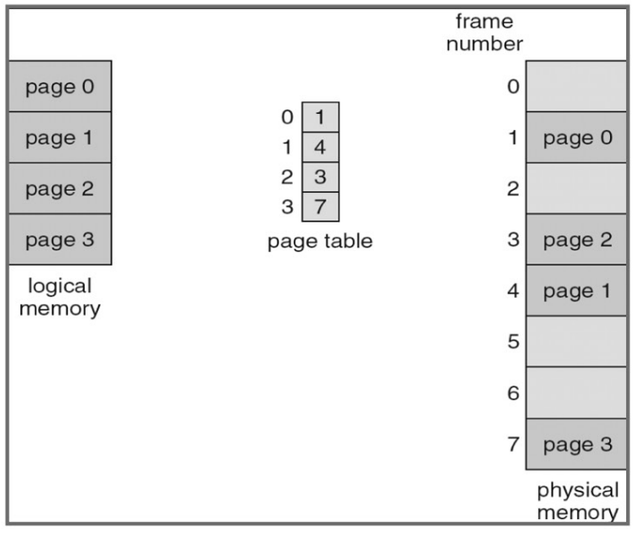
- En la asociación en tiempo de compilación o de carga (*compile, load time*) debe ser el mismo lugar, mientras que, si la asociación es en tiempo de ejecución la asignación del lugar es libre.

Administración de memoria II



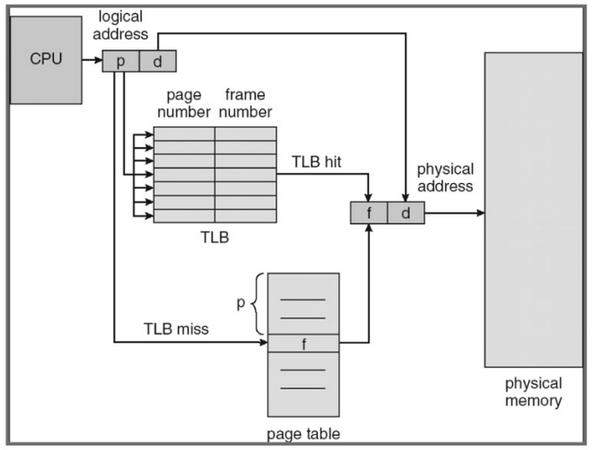
Paginación

- La paginación es una técnica que divide a la memoria física en particiones de tamaño fijo llamados *frames*.
- A su vez, el espacio de direccionamiento virtual es divido en unidades fijas del mismo tamaño que los *frames* (*page size*) denominadas páginas (*pages*).
- Las páginas tienen un tamaño que es potencia de 2 y, en general, son desde 512 bytes a 16 Mb.
- En los sistemas que brindan paginación, la transferencia entre la memoria principal y el disco es siempre en unidad de página.
- Cuando un proceso ejecuta sus páginas son cargadas en los *frames* de memoria principal y en disco (sección de *swap*).
- Los *frames* en el *swap* tienen el mismo tamaño que los *frames* de memoria principal.



Soporte a nivel de hardware

- En los sistemas actuales no es posible guardar todas las entradas en registros de rápido acceso.
- La tabla se mantiene en memoria principal y se asigna un registro que apunta a la dirección base de la tabla (*PTBR – Page Table Base Register*).
- La opción del PTBR mejora el cambio de contexto entre procesos ya que solo es necesario cambiar un registro para acceder a la tabla de página.
- Sin embargo, el acceso a memoria se duplica debido a que es necesario primero acceder a la tabla para obtener el número de *frame* y, posteriormente, al lugar de memoria solicitado.
- La solución es utilizar una pequeña *cache* de la tabla de página: *TLB – Translation Look-aside Buffer.*
- El *cache* TLB es asociativa y de rápido acceso.
- Cada entrada consiste de dos partes: una clave (*tag*) y un valor (el número de *frame*).
- La búsqueda de una clave en la *cache* TLB es simultanea entre todas las *tags*.
- Si la clave es encontrada (*TBL hit*), inmediatamente se genera la dirección buscada a partir del valor asociado. En caso contrario (*TBL miss*), es necesario realizar el acceso a memoria para obtener la entrada. Posteriormente, se guarda el valor obtenido en la *cache* TLB para posteriores accesos (principio de localidad).



Soporte a nivel de hardware

- Las TLB, por lo general, tienen pocas entradas (64 a 1024).
- Algunas *caches* TLB agregan a cada entrada un identificador de espacio de direccionamiento (*ASID – Address Space IDentifier*).
- En la búsqueda de una clave solo serán tenidas en cuenta las entradas cuyo ASID coincida con el del proceso que está ejecutando en el procesador.
- El uso del identificador permite que en la *cache* TLB contengan entradas para varios procesos de forma simultanea.
- Si no se utiliza el ASID, en cada cambio de contexto es necesario "limpiar" las entradas de la TLB, sino se realizarían accesos equivocados a memoria.

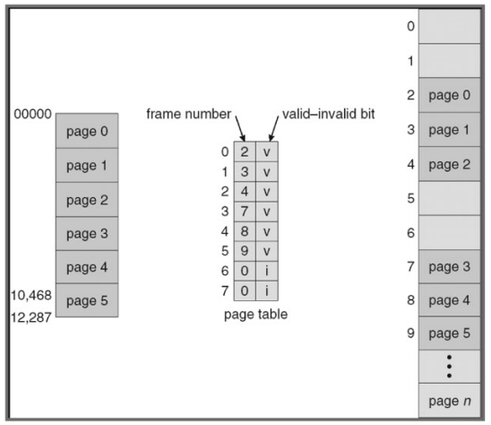
Tiempo efectivo de acceso (*Effective Access Time*)

- El porcentaje de veces que un número de página es encontrado en la *cache* TLB es denominado *hit ratio*.
- El tiempo efectivo de acceso se define mediante la siguiente fórmula:

EAT = *hit ratio * (tiempo de búsqueda en TLB + tiempo de acceso a memoria) + (1 – hit ratio) * (2 * tiempo de acceso a memoria)*

- La medida nos permite saber la ganancia de la utilización de la *cache* TLB.
- La tabla de página tiene una entrada por cada página posible que tenga el proceso.
- Es necesario identificar cuales son entradas válidas y cuales no.
- La utilización de un *bit* de protección en cada entrada determina si la página es válida o inválida (*valid-invalid bit*).
- El acceso a una página cuyo *bit* marca que es inválida genera un *trap* a nivel del sistema operativo.

Protección de memoria

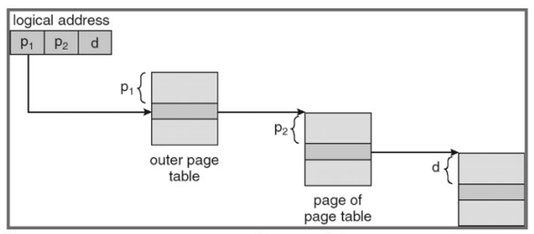


Estructura de la tabla de página

- En un sistema de 32bits que utilice páginas de 4KB se necesitarán cerca de un millón de entradas en la tabla de página.
- Es necesario buscar alguna estructura más eficiente en cuanto al tamaño ocupado por la tabla de página.
- Se proponen las siguientes estructuras:
  - Jerárquica.
  - *Diccionarios* (*hash*).
  - Invertida.

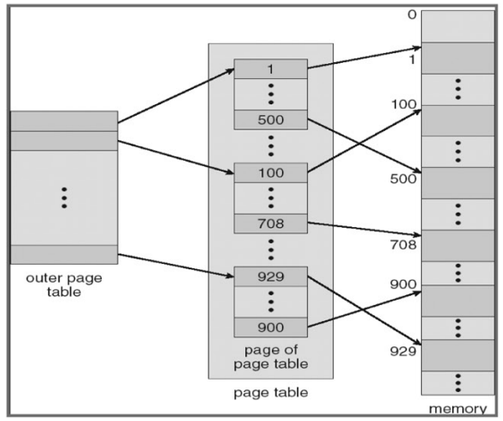
Jerárquica

- La estructura se basa en paginar la tabla de página. De esta forma, existirá una jerarquía de páginas.
- La idea es dividir al componente de indexado en la tabla de página, de la dirección virtual, en varios niveles.



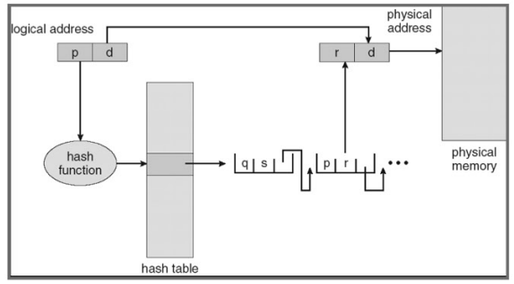
- Este esquema se puede aplicar varias veces y lograr varios niveles de indexación. En el gráfico se tienen dos niveles.

Estructura de la tabla de página - Jerárquica



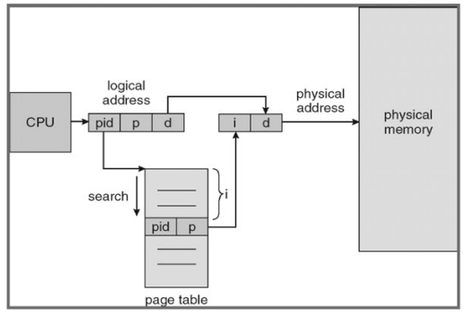
Diccionarios

- Una alternativa es implementar una tabla de *hash* abierto con el valor del componente de número de página.
- Está estructura es conveniente para arquitecturas de más de 32 bits.



Invertida

- Se dispone de una única tabla de página global del sistema. A diferencia de los anteriores que existe una tabla por proceso. Se aumentan los tiempos de búsqueda en favor de la reducción del espacio utilizado en memoria principal.



Compartimiento

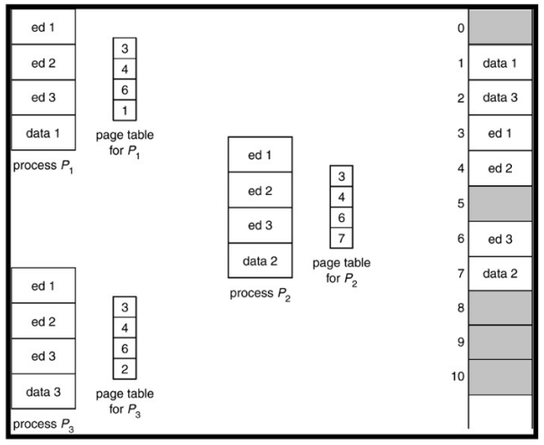
- Los procesos se componen de una parte de códigos y datos privados y de otra que puede ser compartida.

- La posibilidad de dividir el espacio de direccionamiento en páginas, permite a los procesos compartir de forma eficiente las páginas comunes en memoria.

Ejemplo:

  - La sección de código de un mismo proceso.
  - El código de una biblioteca dinámica.
  - Memoria compartida.

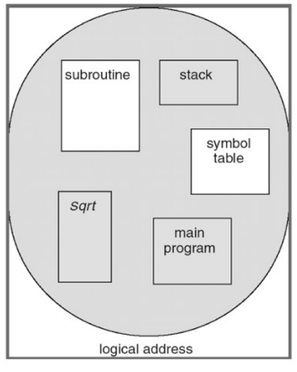
- Esto permite un uso más eficiente de la memoria.



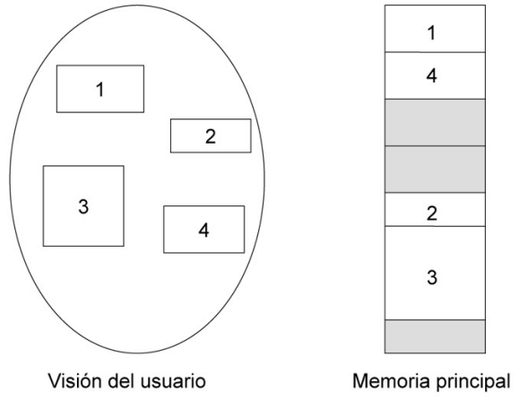
Segmentación

- La segmentación es una técnica que asigna segmentos contiguos de memoria para las áreas de memoria de un proceso.

- De esta forma, logra acomodarse más a la visión de la memoria por parte del usuario.
- Un proceso se compone de una sección de código, una pila (*stack*), un espacio para la memoria dinámica (*heap*), la tabla de símbolos, etc.
- Cada componente se agrupa en un segmento del tamaño necesario.



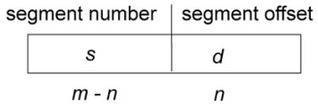
Segmentación



Soporte a nivel de hardware

- Cada segmento tendrá un nombre (o número) y un largo asociado.

- Las direcciones virtuales se componen de un número de segmento y el desplazamiento dentro del segmento.



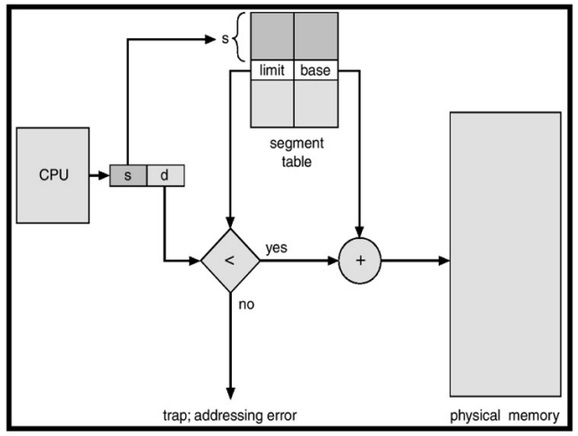
- El desplazamiento debe ser menor que el largo asociado al segmento.

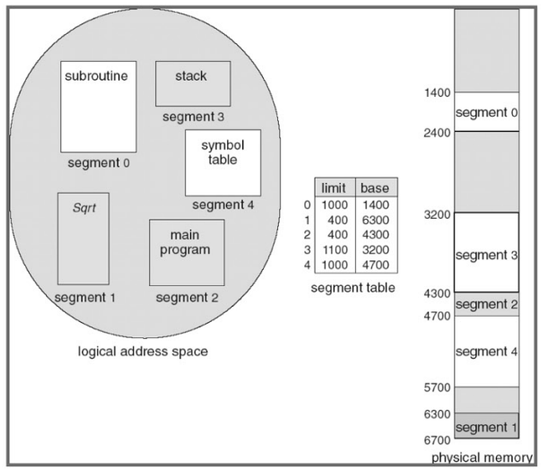
- La tabla de segmentos tendrá una entrada por cada segmento en donde estará la dirección física base del segmento (*base register*) y el largo del mismo (*limit register*).

- En la traducción de dirección virtual a física se controlan el número de segmento con el máximo que tenga el proceso (el registro *STLR – Segment Table length Register* – define el número máximo de segmento utilizado por el proceso) y el desplazamiento contra el registro límite.

- La tabla de segmentos es mantenida en memoria principal y se asigna un registro que apunta a la dirección base de la misma (*STBR – Segment Table Base Register*).

Soporte a nivel de hardware



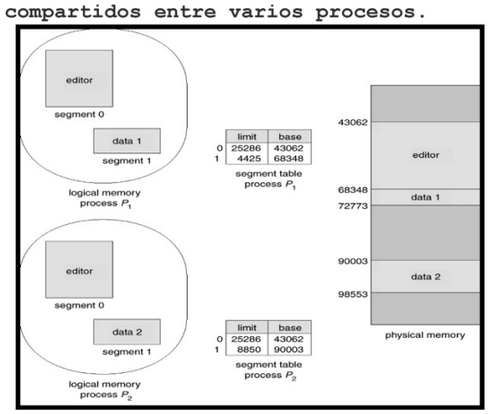


Protección de memoria

- Al dividir la memoria en segmentos se permite que cada uno tenga asociado un conjunto de permisos sobre él.
- Un segmento de código es normal que tenga permisos de lectura y ejecución y no de escritura.
- Un segmento de datos tendrán permisos de lectura y escritura.
- Se define un conjunto de *bits* de protección (*bit protection*) que el hardware controla.

Compartimiento

- Los segmentos permiten una forma clara y sencilla para ser compartidos entre varios procesos.



- La paginación genera fragmentación interna. Dado que la granularidad es a nivel de página, se generará espacio de memoria dentro de las páginas que quedarán sin uso.
- La segmentación sufre de fragmentación externa. Los segmentos son asignados contiguos, pero a medida que son liberados generan huecos que luego, al no ser completados totalmente, generan huecos más chicos que quedan inutilizados. Para solucionar esto es necesario una reorganización de la memoria (tarea no menor).
- La segmentación logra implementar la visión que el usuario tiene de la memoria. En paginación el espacio de direccionamiento de un proceso se distribuye de forma no contigua.

Comparación Paginación - Segmentación

- La segmentación logra compartir y proteger memoria entre procesos de forma más directa. Por ejemplo, en paginación, compartir un espacio de direccionamiento de un proceso implica mantener una gran cantidad de referencias de páginas compartidas, mientras que en segmentación se comparte el segmento.
- En paginación la asignación de una página en memoria es más rápida. Utilizando un vector de *bits* se obtiene de forma sencilla un *frame* libre de memoria donde puede ser asignada la página. En segmentación es necesario mantener una lista y la búsqueda se hace más costosa.

Concurrencia

Agenda

- Conceptos
- El poblema de la Sección-Crítica
- Hardware de Sincronización
- Semáforos
- Problemas Clásicos de Sincronización
- Monitores

Conceptos

▪El acceso concurrente a datos compartidos puede resultar en inconsistencia de los datos.

▪Mantener los datos consistentes requiere de mecanismos para segurar la ejecución de manera "ordenada" de los procesos concurrentes.

▪Supongamos que deseamos dar solución al problema del productor-consumidor. Se puede realizar teniendo un contador (count) entero que lleva la cuenta de la cantidad de buffers llenos. Inicialmente, el contador (count) se inicializa en 0. Es incrementado por el productor cuando produce un nuevo buffer y es decrementado por el consumidor luego que consume el buffer.

▪ Datos Compartidos

```
#define BUFFER_SIZE 10
typedef struct {
 . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
```


Segmentación con paginación

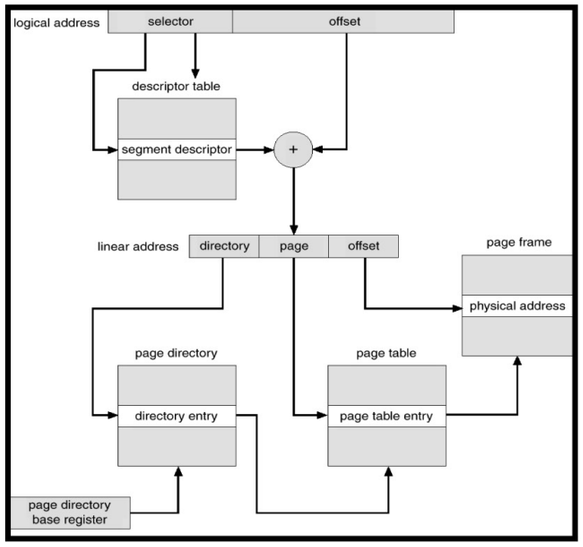
▪ La paginación y la segmentación se pueden combinar para potenciar las ventajas de cada técnica.

▪ Ejemplo de una arquitectura de este tipo es Intel.

▪ La memoria es segmentada, y los segmentos se conforman de páginas.

▪ Las direcciones virtuales contienen un identificador de segmento y un desplazamiento. A partir de ellos se genera una dirección lineal de 32bits (en caso IA32). Luego, la dirección es traducida a una dirección física.

Segmentación con paginación



Conceptos

- Proceso Productor:

```
item nextProduced;

while (1) {
   while (counter == BUFFER_SIZE)
      ; /* do nothing */
   buffer[in] = nextProduced;
   in = (in + 1) % BUFFER_SIZE;
   counter++;
}
```

- Proceso Consumidor:

```
item nextConsumed;

while (1) {
   while (counter == 0)
      ; /* do nothing */
   nextConsumed = buffer[out];
   out = (out + 1) % BUFFER_SIZE;
   counter--;
}
```

- Las sentencias:

```
counter++;
counter--;
```

deben ser ejecutadas de manera atómica.

- Una operación atómica significa que esta debe ejecutar completamente sin ninguna interrupción.

Conceptos

- La sentencia "count++" seguramente se encuentre implementada en lenguaje de máquina como:

```
registro1 = counter
registro1 = registro + 1
counter = registro1
```

- La sentencia "count--" seguramente se encuentre implementada en lenguaje de máquina como:

```
registro2 = counter
registro2 = registro2 - 1
counter = registro2
```

- Si ambos, productor y consumidor, intentan actualizar el buffer de manera concurrente, las sentencias de lenguaje assembler pueden terminar "mezcladas".

- La "mezcla" depende de la manera en que los procesos productor y consumidor son despachados.

- Supongamos que el contador inicialmente vale 5. Una "mezcla" posible de sentencias es:

```
productor:  registro1 = counter (registro1 = 5)
productor:  registro1 = registro1 + 1 (registro1 = 6)
consumidor: registro2 = counter (registro2 = 5)
consumidor: registro2 = registro2 - 1 (registro2 = 4)
productor:  counter = registro1 (counter = 6)
consumidor: counter = registro2 (counter = 4)
```

- El valor de count puede ser 4 o 6, donde el valor correcto debe ser 5.

- Race condition: La situación en la cual varios procesos acceden y manipulan datos compartidos concurrentemente. El valor final de los datos compartidos depende de cuál proceso finaliza último.

- Para prevenir las race conditions, los procesos concurrentes deben estar "sincronizados".

El problema de la Sección Crítica

- Existen n procesos compitiendo por el uso de datos compartidos.

- Cada procesos tiene un trozo de código llamado *sección crítica* donde accedea los datos compartidos.

- Problema: Asegurar que mientras un proceso se encuentra ejecutando su propia sección crítica, ningún otro proceso pueda ejecutar su sección crítica.

- Cualquier solución al problema de la sección crítica debe satisfacer las siguientes condiciones:
  1. Exclusión Mutua. Si un procesos Pi se encuentra ejecutando en su sección crítica, ningún otro proceso puede estar ejecutando su sección crítica.
  2. Progreso. Si ningún proceso está ejecutando su sección crítica y hay procesos que desean ingresar en sus secciones críticas, la selección del próximo proceso que entrará en la sección crítica no puede ser pospuesto de manera indefinida.
  3. Espera limitada. Debe existir un límite de número de veces que se permite a otros procesos ingresar en sus secciones críticas después que un proceso ha solicitado ingresar en su sección crítica y antes que se le otorgue la autorización para hacerlo.
     - Se debe asumir que cada proceso ejecuta a una velocidad distinta de 0.
     - No se deben hacer supuestos acerca de las velocidades relativas de los N procesos.

Intento de solución inicial

- Solo dos procesos, P0 and P1.
- Estructura general del proceso Pi(El otro proceso es Pj).

```
do {
    entry section
        critical section
    exit section
        reminder section
} while (1);
```

- Los Procesos pueden compartir algunas variables comunes para sincronizar sus acciones.

El problema de la Sección Crítica

- Algoritmo 1

- Variables Compartidas:
  - int turn;
    Se inicializa turn = 0
  - turn = i ⇒ P_i puede entrar a su sección crítica

```
Process Pi
        do {
           while (turn != i) ;
              critical section
           turn = j;
              reminder section
        } while (1);
```

- Satisface la exclusión mutua pero no la condición de Progreso.

- Algoritmo 2

- Variables Compartidas:
  - boolean flag[2];
    Se inicializa flag [0] = flag [1] = false.
  - flag [i] = true ⇒ P_i listo para acceder a su sección crítica

```
Process Pi
        do {
           flag[i] := true;
           while (flag[j]) ;
                 critical section
           flag [i] = false;
              remainder section
        } while (1);
```

- Satisface la exclusión mutua pero no la condición de Progreso.

- Algoritmo 3

- Variables Compartidas:
  - Combinación de variables compartidas en Algoritmo1 y Algoritmo2:

```
do {
        flag [i]:= true;
        turn = j;
        while (flag [j] and turn = j) ;
           critical section
        flag [i] = false;
           remainder section
} while (1);
```

- Satisface los tres requerimientos. Resuelve el problema de la exclusión mutua para dos procesos.
- ¿Es posible generalizarlo para N procesos?

El problema de la Sección Crítica

- Generalización para N procesos.
- Algoritmo de la "Panadería".
- Antes de entrar a su sección crítica el proceso recibe un número. El que tiene el número menor entra a su sección crítica.
- Si los procesos P_i y P_j reciben el mismo número, si i < j entonces P_i entra primero; sino P_j entra primero.
- El esquema de numeración siempre genera números en orden creciente, Ej: 1,2,3,3,3,3,4,5...
- Notación:
  - Orden lexicográfico (ticket #, process-id #)
  - $(a,b) < (c,d)$ Si $a < c$ o if $a = c$ y $b < d$
  - max $(a_0,\ldots, a_{n-1})$ es un número, k, tal que $k \geq a_i$ para $i = 0, \ldots, n-1$
- Datos compartidos:

```
boolean choosing[n];
int number[n];
```

Las estructuras de datos se inicializan a false o 0 según corresponda.

Hardware de Sincronización

- Muchos sistemas proveen soluciones basadas en hardware para el problema de la sección crítica.
- Monoprocesadores – se puede deshabilitar las interrupciones.
  - El código ejecutará sin "preemption".
  - Solución ineficiente en sistemas multiprocesador.
    - Sistemas Operativos que usan esta solución no escalan.
- El equipamiento moderno provee instrucciones atómicas especiales.
  - Atómico = no-interrumpible
  - Puede ser: verificar una palabra de memoria y escribir un valor
  - O intercambiar (swap) el contenido de dos palabras de memoria.
- Solución al problema de la sección crítica utilizando "locks"

```
do {
  adquirir lock
    critical section
  liberar lock
    remainder section
} while (TRUE);
```


- Instrucción TestAndSet
- Definición:

```
boolean TestAndSet (boolean *target)
        {
             boolean rv = *target;
             *target = TRUE;
             return rv:
        }
```

- Solución utilizando TestAndSet
  - Variable compartida lock inicializada en false:

```
do {
      while ( TestAndSet (&lock )) ;   // do nothing
                //    critical section
        lock = FALSE;
                //      remainder section
  } while (TRUE);
```

- Instrucción Swap
- Definición:

```
void Swap (boolean *a, boolean *b)
        {
             boolean temp = *a;
             *a = *b;
             *b = temp:
        }
```

- Solución utilizando Swap:
  - Variable compartida lock inicializada en false.
  - Cada proceso tiene una variable boolean local key:

```
do {       key = TRUE;
           while ( key == TRUE)
                   Swap (&lock, &key);
                     //    critical section
            lock = FALSE;
                     //    remainder section
  } while (TRUE);
```

Semáforos

- Herramienta de sincronización que no requiere Busy Waiting.
- Semáforo S – variable entera.
- Dispone de dos primitivas estandar S: wait() and signal():
  - Originalmente llamadas P() and V().
- Solamente puede ser accedido mediante dos operaciones indivisibles (atómicas).

```
- wait (S) {
          while S <= 0
             ; // no-op
            S--;
    }

- signal (S) {
       S++;
  }

    S++;
}
```

Semáforos

- Semáforos como herramienta de sincronización.
- Semáforo Binario – El valor entero puede ser 0 o 1; es más sencillo de implementar.
  - Son también conocidos como "mutex locks".
- Semáforo no binario – El valor entero tiene su dominio sobre los números enteros.
- Proveen exclusión mutua:

```
Semaphore mutex;    //  initialized to 1
do {
     wait (mutex);
        // Critical Section
     signal (mutex);
        // remainder section
} while (TRUE);
```

Semáforos - Implementación

- Debe garantizar que dos procesos puedan ejecutar wait() y signal() en el mismo semáforo a la vez.

- La implementación se convierte en el problema de la sección crítica donde el código de signal y wait son colocados en la sección crítica.
  - Existe busy waiting en la implementación de la S. C. pero….
    - El código de la sección crítica es corto.
    - Existe poco Busy Waiting si la S. C. es ocupada eventualmente.

- Las aplicaciones pueden pasar mucho tiempo dentro de las S. C., por lo tanto, no es una buena solución.

- Implementación sin Busy Waiting.

- Con cada semáforo hay una cola de espera asociada. Cada entrada en la lista de espera tiene dos elementos.
  - Value (de tipo integer).
  - Pointer (puntero) al próximo elemento en la lista.

- Existen dos operaciones:
  - block – inserta al proceso que invoca la operación en la lista de espera apropiada.
  - wakeup – Borra uno de los procesos de la lista de espera y lo pone la lista de "ready".

Problema del Buffer Acotado

- N buffers, cada uno puede contener un elemento.

- Semáforo mutex inicializado en 1.

- Semáforo full inicializado en 0.

- Semáforo empty inicializado en N.

- Código del proceso productor:

```
do  {

        //   produce an item in nextp

        wait (empty);
        wait (mutex);

        //  add the item to the  buffer

        signal (mutex);
        signal (full);

} while (TRUE);
```

- Código del proceso consumidor:

```
do {

        wait (full);
        wait (mutex);

        //  remove an item from  buffer to nextc

        signal (mutex);
        signal (empty);

        //  consume the item in nextc

} while (TRUE);
```

Problema de los Lectores - Escritores

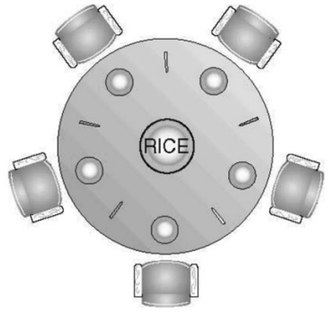
- Código del proceso escritor:

```
do {
        wait (wrt) ;

        //    writing is performed

        signal (wrt) ;

    } while (TRUE);
```

- Código del proceso lector:

```
do {
        wait (mutex) ;
        readcount ++ ;
        if (readcount == 1)
              wait (wrt) ;
        signal (mutex)

              // reading is performed

         wait (mutex) ;
         readcount  - - ;
        if (readcount  == 0)
            signal (wrt) ;
        signal (mutex) ;
} while (TRUE);
```

Problema de los filósofos que cenan



- Datos compartidos:
  - Tazón de arroz (Rice) (datos).
  - Semaforo palito [5] inicializado en 1.

Problema de los filósofos que cenan

- Estructura del filósofo i

```
do  {
        wait ( palito[i] );
        wait ( palito[ (i + 1) % 5] );

              //  comer

         signal (palito[i] );
        signal (palito[ (i + 1) % 5] );

              //  pensar

} while (TRUE);
```

Posibles problemas con semáforos

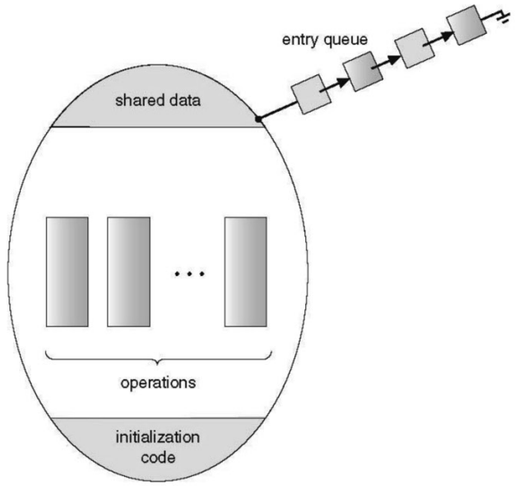
- Uso incorrecto de las operaciones sobre semáforos:
  - signal (mutex) …. wait (mutex).
  - wait (mutex) … wait (mutex).
  - Omitir un wait (mutex) o signal (mutex) (o ambos).

Monitores

- Una abstracción de alto nivel que provee un mecanismo efectivo para la sincronización de procesos.
- Solamente un proceso puede estar activo dentro del monitor a la vez.

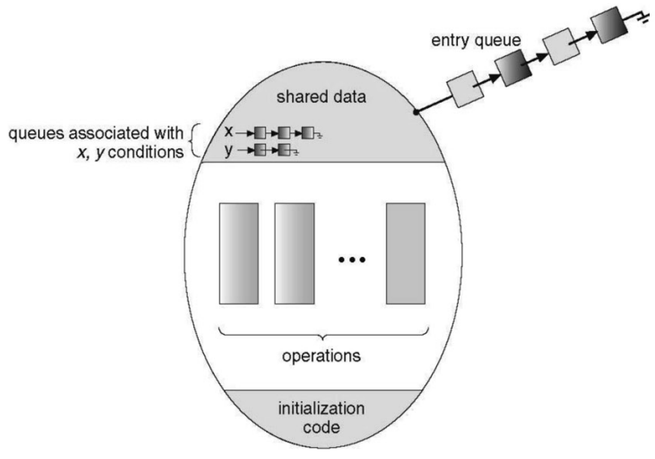
```
monitor nombre-del-monitor
{
 // Declaración de variables compartidas
 procedure P1 (…) { …. }
  …

 procedure Pn (…) {……}

     Código-de-inicialización ( ….) { … }
  …
 }
}
```

Monitor - Internas



Monitores – Variables Condition

- condition x, y;
- Existen dos operaciones sobre las variables condition:
  - x.wait () – Un proceso que invoca la operación se bloquea.
  - x.signal () – desbloquea uno de los procesos (si hay alguno) que previamente ejecutaron x.wait ()



Monitores – Filósofos que cenan

```
monitor DP
 {
    enum { PENSANDO; HAMBRIENTO, COMIENDO) state [5] ;
    condition self [5];

    void pickup (int i) {
        state[i] = HAMBRIENTO;
        test(i);
        if (state[i] != COMIENDO) self [i].wait;
    }

    void putdown (int i) {
        state[i] = PENSANDO;
                  // verificar vecinos derecho e izquierdo
         test((i + 4) % 5);
         test((i + 1) % 5);
    }


     void test (int i) {
           if ( (state[(i + 4) % 5] != COMIENDO) &&
           (state[i] == HAMBRIENTO) &&
              (state[(i + 1) % 5] != COMIENDO) ) {
                 state[i] = COMIENDO;
                 self[i].signal () ;
              }
     }

     initialization_code() {
           for (int i = 0; i < 5; i++)
              state[i] = THINKING;
     }
}
```

- Cada filósofo invoca a las operaciones pickup() y putdown() en el siguiente orden:

```
DiningPhilosophters.pickup (i);

    COMER

DiningPhilosophers.putdown (i);
```

Subsistema de entrada-salida

Agenda

- Introducción.
- Métodos para realizar una E/S.
- Interfaz de aplicación de E/S.
- Subsistema de E/S.

Introducción

- Una de las principales funciones de un sistema operativo es controlar todos los dispositivos de Entrada/Salida que estén conectados al mismo: teclado, ratón, impresora, monitor, red, etc.

- Para encapsular los detalles de los diferentes dispositivos el núcleo del sistema operativo es estructurado en el uso de módulos de dispositivos.

- Los manejadores de dispositivos (devices drivers) presentan un acceso uniforme al subsistema de Entrada-Salida.

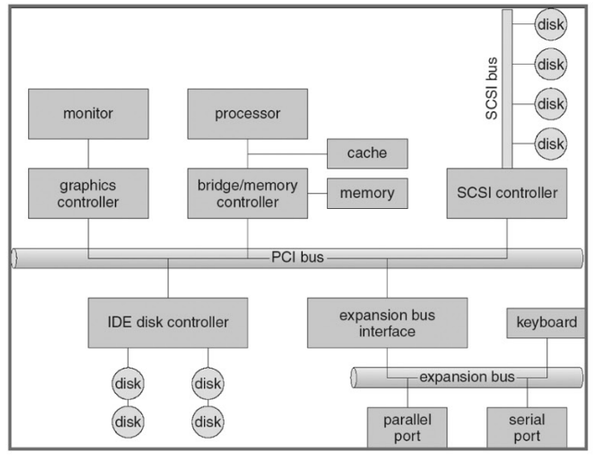
- Los dispositivos se comunican con la computadora a través de señales sobre un puerto.

- Si varios dispositivos utilizan el mismo medio de comunicación, la conexión es llamada bus.

- Un bus es un conjunto de líneas (wires) y un protocolo que especifica un conjunto de mensajes que son enviados a través de él.

- Los buses son comúnmente usados en los sistemas de computación para interconectar los dispositivos.

Introducción



Introducción

- Un controlador (o adaptador) es un chip electrónico que puede operar en un puerto, bus o dispositivo.

- Por ejemplo, un controlador de puerto serial es un dispositivo que controla las señales sobre un puerto serial.

- Un controlador de bus SCSI es más complejo debido al protocolo SCSI.

- Muchos dispositivos tienen implementado su propio controlador que está incorporado al dispositivo (ej.: discos).

- La comunicación con el controlador por parte del procesador es a través de registros de datos y señales de control. El procesador se comunica con el controlador escribiendo y leyendo información en los registros del controlador.

- Una vía de comunicación es a través de instrucciones de E/S especiales, que especifican una transferencia de un byte, o palabra hacia una dirección de puerto de E/S (I/O port address).

- La instrucción de E/S genera adecuadas señales sobre las líneas del bus para lograr la comunicación con el dispositivo adecuado y mover bits hacia y fuera de los registros del dispositivo.

- Otra alternativa es utilizar mapeo de memoria del dispositivo de E/S (*memory-mapped I/O*). Los registros del dispositivos son "mapeados" a memoria principal.

- Un ejemplo de este tipo de comunicación es el controlador de una tarjeta de video.

Introducción

Un puerto de E/S consiste, generalmente, en cuatro registros:

- Estado (status): estos registros pueden ser leídos por el equipo. Informan del estado del dispositivo (completitud de una operación, disponibilidad de leer del registro de datos de entrada, existencia de un error en el dispositivo.
- Control (control): pueden ser escritos por el equipo para generar un pedido o cambiar de modo el dispositivo.
- Datos de entrada (data-in): son leídos por el equipo para obtener la entrada.
- Datos de salida (data-out): son escritos por el equipo para enviar una salida.

Métodos para efectuar una E/S

- E/S Programada (*Programmed I/O*): El procesador le comunica un pedido a la controladora del dispositivo y queda en un 'busy waiting' consultando a la controladora para verificar el estado del pedido.
- Interupcciones (*Interrupts-Driven I/O*): El procesador le comunica el pedido a la controladora y se libera para realizar otras tareas. Al culminar el pedido el dispositivo, la controladora genera una interrupción al procesador.
- Acceso directo a memoria (*DMA – Direct Memory Access*): Se utiliza un chip especial que permite transferir datos desde alguna controladora a memoria sin que el procesador tenga que intervenir en forma continua.

E/S Programada (*Programmed I/O*)

- El procesador genera una solicitud de E/S y luego se encarga de controlar la completitud de la misma controlando algún registro del controlador de dispositivo.
- La consulta la realiza en una iteración continua denominada *polling* o *busy waiting*.
- La técnica de *busy waiting* consume ciclos de procesador en forma innecesaria.

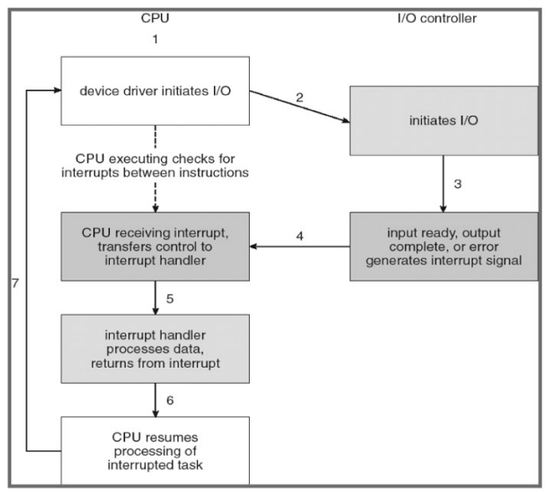
E/S Programada (Programmed I/O)

```
copy_from_user(buffer,p,count);
for (i=0; i < count; i++) {
    while(*print_status_reg != READY) ;
    *printer_data.register = p[i];
}
return_to_user();
```

- Si bien permite una programación simple, tiene como gran desventaja el desperdicio de ciclos de procesador que no deben ser desperdiciados.

Interrupciones (*Interrupt-Driven I/O*)

- El proceso que se está ejecutando realiza la solicitud de E/S, se agrega a la cola de espera del dispositivo y, finalmente, invoca al planificador (*scheduler*) para que asigne el procesador a otro proceso.
- El controlador de dispositivo avisará la completitud de la solicitud a través de una interrupción. Una rutina de atención de la interrupción será invocada interrumpiendo la ejecución del proceso asignado al procesador. Es necesario salvar el estado del proceso que estaba ejecutando.
- Una vez que la completitud de la E/S es registrada por el manejador de la interrupción (*interrupt handler*), el proceso que generó la solicitud es desbloqueado y se lo asigna a la lista de procesos listos.



Interrupciones (*Interrupt-Driven I/O*)

```
copy_from_user(buffer,p,cuenta);
habilitar_interrupciones();
While (*printer_status_reg != READY) ;
*printer_data_register = p[0];
add_to_queue(current);
scheduler();
remove_from_queue(current);
```

- La rutina *scheduler* asignará el procesador a otro proceso. Cuando este proceso vuelva a ejecutar, lo hará luego de la instrucción *scheduler*.

```
if (count == 0)
    unblock_user();
else {
    *printer_data_register = p[i];
    count--;
    i++;
}
acknowledge_interrupt();
return_from interrupt();
```

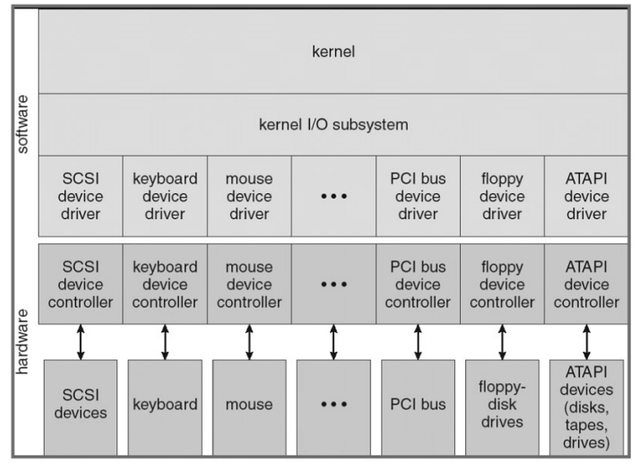
- La gran ventaja es que el procesador queda disponible para otro proceso que requiera el recurso. Es necesario soporte de hardware, además, se deben codificar los manejadores de interrupciones (*interrupts handlers*).

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Interfaz de aplicación de E/S

- El sistema operativo propone estructuras e interfaces para lograr manipular dispositivos de E/S de forma estándar y uniforme.
- Esta aproximación genera abstracción, encapsulamiento y utilización de capas en el software.
- Se abstraerán los dispositivos identificando sus características y proponiendo una forma uniforme de acceso a los dispositivos del mismo tipo.
- Cada dispositivo (si bien pueden pertenecer a un mismo tipo) encapsulará la comunicación con la controladora en módulos llamados *device drivers* independientes.
- Lograr independizar el subsistema de E/S del hardware simplifica el trabajo a los desarrolladores de un sistema operativo.

Interfaz de aplicación de E/S



- Características de los dispositivos de E/S

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Interfaz de aplicación de E/S

- Dispositivos de bloques (*Block devices*)

  Son dispositivos cuya granularidad de información es a nivel de bloques. Se especializan en transferir grandes volúmenes de datos.

  Ej.: Discos.

- Dispositivos de caracteres (*Char devices*)

  La granularidad es a través de caracteres.

  Ej.: teclados, ratón, puertos serial.

- Operaciones bloqueantes (*Blocking*)

  Cuando un proceso requiere de un servicio de E/S a través de una rutina bloqueante el proceso se suspende hasta que la operación haya finalizado.

  Es fácil de utilizar y entender.

- Operaciones no bloqueantes (*Nonblocking*)

  El llamado al servicio de E/S es devuelto tan pronto como sea posible.

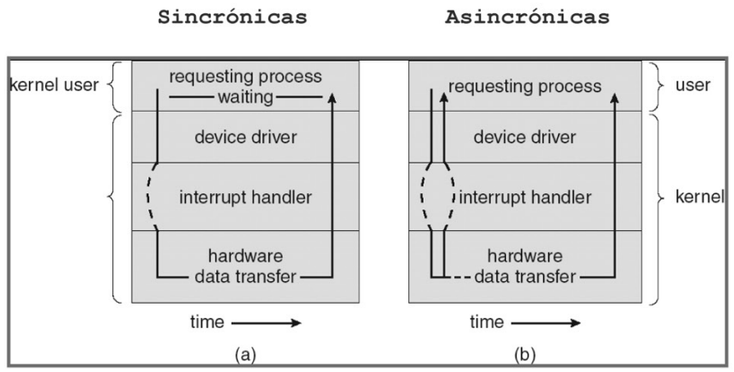
- Operaciones asincrónicas (*asynchronous*)

  La operación es ejecutada en paralelo. Cuando finaliza le avisa a través de una señal. Son difíciles de utilizar.

- Operaciones sincrónicas (*synchronous*)

  Los procesos que realizan pedidos de E/S se bloquean hasta que el pedido finalice.

Interfaz de aplicación de E/S



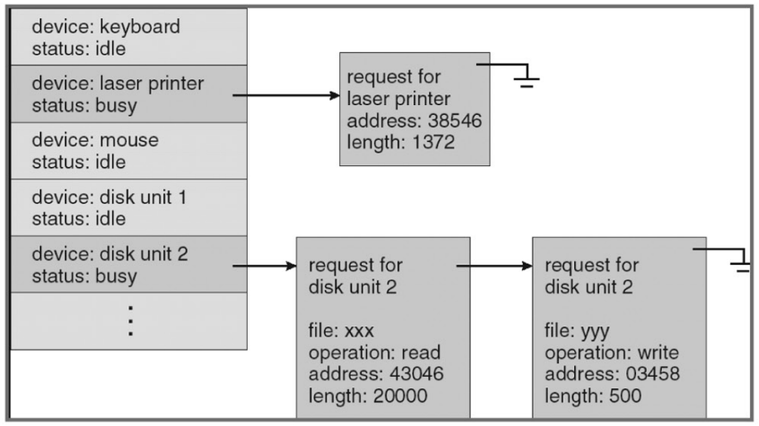
Subsistema de E/S

- El núcleo del sistema operativo brinda varios servicios para el manejo de E/S que están desarrollados en la infraestructura de hardware y device drivers:
  - Planificación de E/S (*I/O Scheduling*).
  - Buffering.
  - Caching.
  - Spooling.
  - Manejo de errores (*Error handling*).

Planificación de E/S (*I/O Scheduling*)

- La planificación de requerimientos de E/S debe ser hecha para lograr un buen rendimiento del dispositivo.
- Seguramente, los pedidos generados a través de llamados a sistema por parte de los procesos no generan una buena secuencia de planificación para el dispositivo.
- El sistema operativo implementa la planificación manteniendo una cola de pedidos por cada dispositivo.
- Cuando un proceso genera una E/S bloqueante, es puesto en la cola del dispositivo correspondiente y el subsistema de E/S reorganiza los pedidos para lograr un mayor rendimiento.

Planificación de E/S (*I/O Scheduling*)



Buffering

- Es un lugar de memoria que guarda información (datos) mientras son transferidos entre dos dispositivos o un dispositivo y una aplicación.
- Existen tres razones para realizar buffering:
  - Normalizar las velocidades entre diferentes dispositivos.
  - Adaptarse entre dispositivos que difieren en los tamaños de transferencia.
  - Mantener la semántica de aplicaciones que realizan E/S: en una operación write el buffer de usuario es copiado a un buffer del sistema operativo. De esa forma, el sistema logra independizarse de la aplicación.

Caching

- La cache es una región de memoria más rápida que contiene copias de datos.
- Su utilidad es acelerar el acceso a la información.
- En buffering se tienen los datos originales, mientras que caching tiene una o varias copias en un medio de memoria más rápido.
- El caching introduce problemas de consistencia de la información.

Spooling

- Es un buffer que mantiene salida para un dispositivo que no se pueda intercalar.
- El sistema captura la salida para el dispositivo y la va guardando para brindarla en forma correcta (sin intercalar).
- El spooling es una forma que el sistema operativo tiene para coordinar salida concurrente para dispositivos.
- Tiene la ventaja que libera al proceso, permitiendo continuar su ejecución. Su trabajo es guardado y será enviado al dispositivo cuando el subsistema lo crea conveniente.
- Ej.: impresora.

Manejo de errores (*Errors handling*)

- Cada llamado a sistema retorna un bit que informa el éxito o fracaso de la operación sobre una E/S.
- En UNIX se utiliza una variable errno que es utilizada para codificar el error.

Estructuras de dispositivos masivos de datos

Agenda

- Planificación de disco.
- Estructuras RAID.

Planificación de disco

- El sistema operativo es responsable de usar el hardware de forma eficiente. Desde la perspectiva del disco esto significa obtener un rápido acceso a los datos y aprovechar al máximo el ancho de banda al disco.

- Es por eso que el planificador de disco es uno de los más importantes.

- El acceso a disco tiene dos grandes componentes:

  - Tiempo de posicionamiento (*seek time*): Es el tiempo que el brazo del disco necesita para posicionar la cabeza en el cilindro que contiene el sector.
  - Latencia de rotación (*rotational latency*): Es el tiempo que demora rotar el plato al sector correcto.

- El ancho de banda (*bandwidth*) de un disco es la cantidad de bytes transferidos, dividido por tiempo total de la transferencia (desde el comienzo del pedido hasta la última transferencia).

- Para mejorar el acceso a los datos se debe minimizar el tiempo de búsqueda. De esa forma, surgen varios métodos de planificación de disco:

  - FCFS.
  - SSTF.
  - SCAN.
  - C-SCAN.
  - LOOK.
  - C-LOOK.

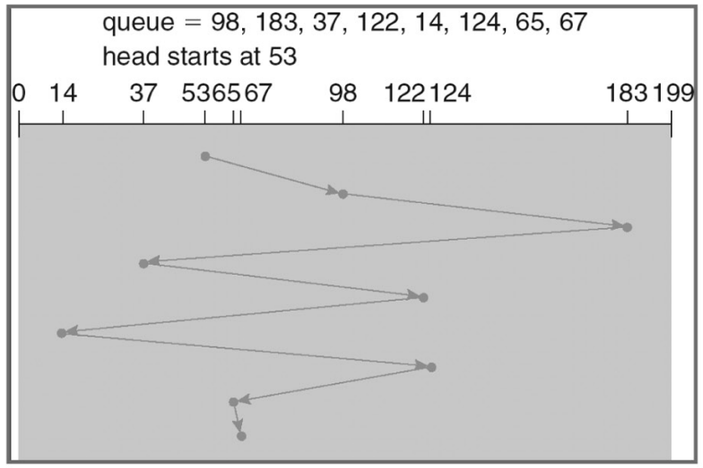
FCFS – First-Come, First-Served

- En este caso, la planificación es realizar los pedidos como vayan llegando.

- En un disco con 200 cilindros (numerados del 0 al 199), la cabeza posicionada en el cilindro 53 y una lista de pedidos:

  98, 183, 37, 122, 14, 124, 65, 67

- Genera un movimiento de la cabeza sobre 640 cilindros.



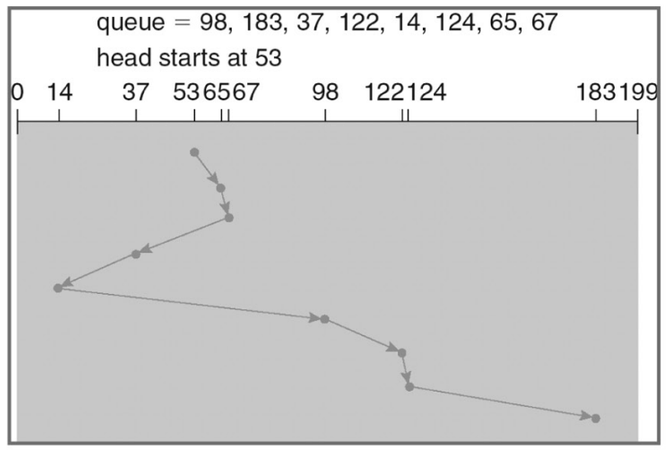
SSTF – Shortest Seek Time First

- Se elige como próximo pedido a realizar el que genere menos tiempo de búsqueda (*seek time*).

- Esto puede generar que pedidos nunca sean ejecutados o demorados mucho tiempo.

- Para el ejemplo anterior tenemos que la cabeza recorrerá un total de 236 cilindros.

SSTF - Shortest Seek Time First



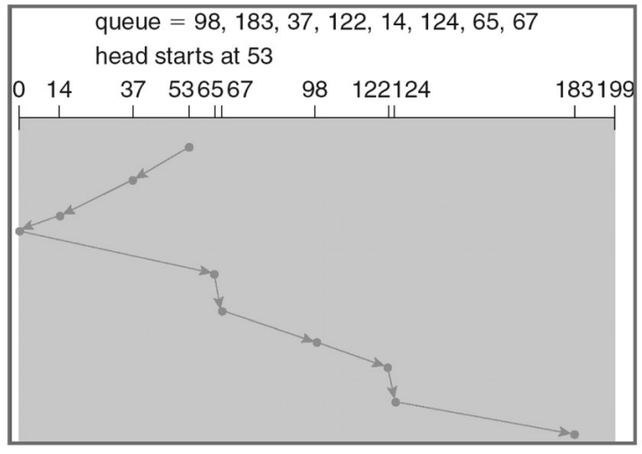
SCAN - C-SCAN

SCAN

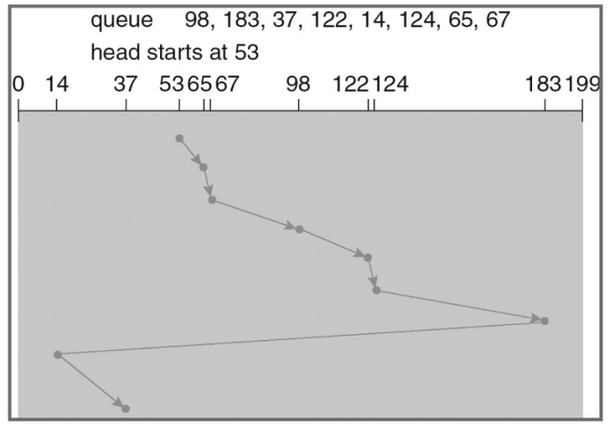
- El brazo posiciona la cabeza al comienzo del disco y la mueve hacia el otro extremo resolviendo los pedidos mientras pasa por los cilindros. Al llegar al final hace el camino inverso resolviendo las solicitudes.

Es también llamado el algoritmo del elevador.

En este caso se recorre un total de 208 cilindros.

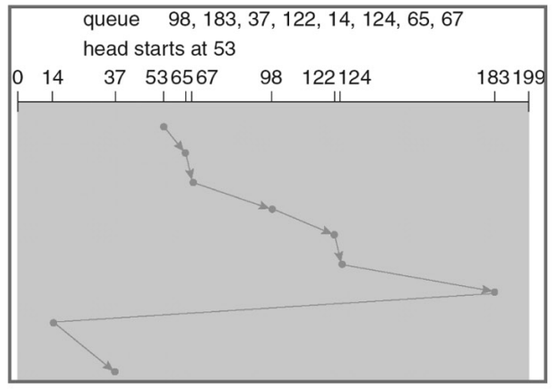


C-SCAN



C-LOOK

- Es parecido al C-SCAN, pero el brazo va hasta donde haya pedidos y retorna hacia el otro sentido.



Comparación

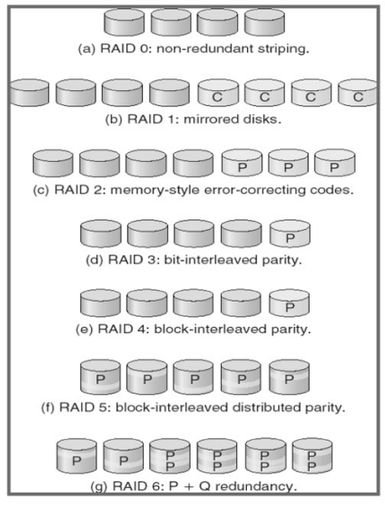
- SSTF y LOOK son los más utilizados.
- SCAN y C-SCAN han demostrado ser mejores ante sistemas que tienen una alta carga en disco.
- El servicio de disco es muy dependiente del método de asignación de archivos.

Estructuras RAID

- Los sistemas de almacenamiento de disco (*storage*) presentan tecnologías orientadas a mejorar el servicio.
- Las mejoras implementadas se basan en dar:
  - Confiabilidad: Casos de fallo de discos.
  - Performance: Lograr mejorar los tiempos de transferencia.
- Las técnicas denominadas RAID (*Redundant Arrays of Inexpensive Disks*) tienen una amplia aceptación.
- La confiabilidad es lograda a través de redundancia de la información.
- La redundancia puede darse duplicando discos (*mirror*) o a través de bits de control.
- La mejora de los tiempos de respuesta (*performance*) se logra a través de la disposición de la información en los diferentes discos.
- Técnicas de *striping* son utilizadas a nivel de bit, byte, sectores, bloque de sectores y bloque.

Diferentes niveles de RAID

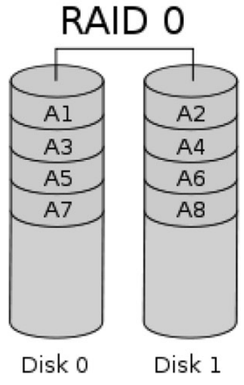
- RAID 0.
- RAID 1.
- RAID 2.
- RAID 3.
- RAID 4.
- RAID 5.
- RAID 6.



RAID 0

- Un RAID 0 (*stripe set* o *striped volume*) divide los datos de forma homogénea en dos o más discos (rayadas) sin información de paridad para la redundancia.
- Es importante señalar que el RAID 0 no era uno de los niveles RAID originales y no proporciona redundancia de datos.
- RAID 0 se utiliza normalmente para aumentar el rendimiento, aunque también se puede utilizar como una forma de crear un pequeño número de grandes discos virtuales de un gran número de pequeñas unidades físicas.
- A pesar de RAID 0 no se ha especificado en el documento original de RAID. Una aplicación idealizada de RAID 0 iba a dividir operaciones E / S en bloques de igual tamaño y repartirlos uniformemente a través de dos discos.

RAID 0 - Diagrama



RAID 0 - Performance

- Si bien el tamaño del bloque técnico pueden ser tan pequeño como un byte, es casi siempre un múltiplo del tamaño del disco duro de sector de 512 bytes.
- De este modo, cada unidad será independiente cuando buscan al azar leyendo o escribiendo datos en el disco. Para las lecturas y escrituras que son más grandes que el tamaño de strip, tales como copiar archivos o la reproducción de video, los discos deben entrar en la misma posición en cada disco, así que el tiempo de búsqueda de la matriz será el mismo que el de una sola unidad.
- Para las lecturas y escrituras que son más pequeñas que el tamaño del strip, tales como el acceso de base de datos, las unidades se pueden buscar de forma independiente.

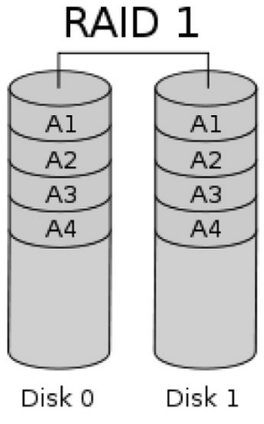
RAID 0 - Performance

- Si los sectores accedidosse distribuyan de forma equilibrada entre las dos unidades, la búsqueda al azar será N veces mas rápida.
- La velocidad de transferencia de la matriz será la velocidad de transferencia de todos los discos sumados, limitado sólo por la velocidad de la controladora RAID.

RAID 1

- Un RAID 1 crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos.
- Esto es útil cuando el rendimiento de lectura o la confiabilidad son más importantes que la capacidad de almacenamiento de datos.
- Este tipo de arreglo sólo puede ser tan grande como el disco más pequeño. Un clásico RAID 1 par duplicado contiene dos discos.
- Puesto que cada miembro contiene una copia completa de los datos, que pueden tratarse de forma independiente, la fiabilidad es incermentada por la potencia del número de copias.
- Vale la pena señalar que mientras que RAID 1 puede ser una protección eficaz contra la falta de disco físico, no ofrece protección contra la corrupción de datos debido a virus, cambios o eliminaciones accidentales de archivos o cualquier otro cambio de datos específicos.

RAID 1 - Diagrama

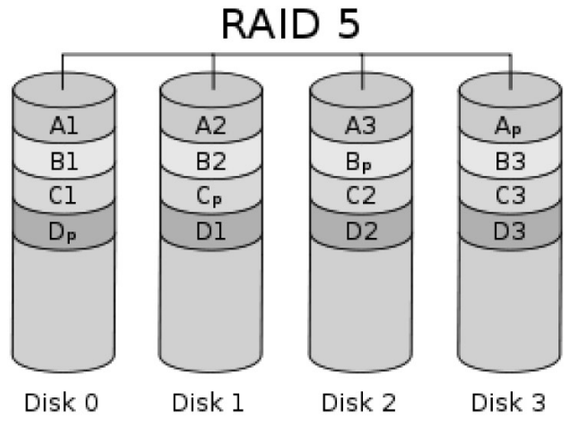


RAID 5

- Aborda las dos mejoras de servicio planteadas.
- Un RAID 5 usa paridad a nivel de bloque con los datos de paridad distribuido entre todos los discos miembros.
- RAID 5 ha logrado popularidad debido a su bajo costo de redundancia.
- Cuatro unidades de 1 TB pueden ser usadas para construir un arreglo de 3-TB en RAID 5.

RAID 5 - Diagrama

- Paridad distribuida (cada color representa el conjunto c/bloque de paridad).



- En el ejemplo, una solicitud de lectura para el bloque A1 sería servida por el disco 0.
- Una solicitud de lectura simultánea para el bloque B1 tendría que esperar, pero una solicitud de lectura de B2 podrían ser revisados al mismo tiempo por el disco 1 logrando mejor tiempo de respuesta.

RAID 5 – Manejo de paridad (*parity handling*)

- Una serie de bloques simultáneos (uno en cada uno de los discos de una matriz) se le llama colectivamente una raya. Si otro bloque, o alguna parte del mismo, está escrito en esa misma franja, el bloque de paridad, o alguna parte del mismo, se vuelve a calcular y reescrito.

- Para las pequeñas escribe, esto requiere:
  1. Leer el bloque de datos antiguos.
  2. Leer el bloque de paridad de edad.
  3. Comparar el bloque de datos de edad con la solicitud de escritura. Por cada bit que se ha invertido (cambia de 0 a 1, o de 1 a 0) en el bloque de datos, dar la vuelta el bit correspondiente en el bloque de paridad.
  4. Escribe el bloque de datos nuevos.
  5. Escriba el nuevo bloque de paridad.

RAID 5 – Manejo de paridad (*parity handling*)

- El disco utilizado para el bloque de paridad está escalonado de una banda a otra, de ahí el término bloques de paridad distribuida.
- RAID 5 escribe son costosas en términos de operaciones de disco y el tráfico entre los discos y el controlador.
- Los bloques de paridad no se leen en las lecturas de datos. Los bloques de paridad se leen, sin embargo, cuando una lectura de bloques en el strip y en el bloque de paridad en la franja se utilizan para reconstruir el sector andantes.
- Leer y escribir al arreglo sigue sin problemas, aunque con alguna degradación en el rendimiento ...

RAID 5 - Performance

- El error CRC se oculta así de la computadora principal. Del mismo modo, si un disco falla en la matriz los bloques de paridad de los discos sobrevivientes son combinados matemáticamente con los bloques de datos desde los discos sobrevivientes para reconstruir los datos de la unidad que ha fallado en la marcha.

- A veces se denomina interino Data Recovery Mode. El equipo sabe que una unidad de disco ha fallado, pero esto es solo para que el sistema operativo pueda notificar al administrador de las necesidades de reemplazo de una unidad; aplicaciones que se ejecutan en el equipo son conscientes de la falla. Leer y escribir a la matriz unidad sigue sin problemas, aunque con alguna degradación en el rendimiento.

Sistema de archivos

Agenda

- Interfaz.
  - Archivos.
  - Directorios.
  - Seguridad en archivos.
- Implementación.
  - Definiciones.
  - Sistema de archivos virtual.
  - Estructura de los directorios.
  - Métodos de asignación.
  - Administración del espacio libre.
- Ejemplo UNIX.

- Los dispositivos masivos (discos duros, cintas, etc.) permiten guardar información de forma no volátil.

- El sistema operativo se abstrae de las propiedades físicas de los dispositivos para definir una unidad lógica de almacenamiento: el archivo.

- Los archivos son "mapeados" por el sistema a los distintos dispositivos.

- Un archivo es una agrupación de información que es guardada en algún dispositivo no volátil.

- Desde la perspectiva del usuario, es la unidad mínima de almacenamiento que el sistema le provee.

Los archivos poseen atributos:

- Nombre: un nombre simbólico que permite identificar el archivo a los usuarios. Pueden existir más de un archivo con el mismo nombre en un sistema de archivos jerárquico (directorios).
- Identificador: símbolo que lo identifica de forma única a nivel global del sistema de archivos. Usualmente es un número.
- Tipo: programa ejecutable, archivo de datos, etc.
- Ubicación: Puntero al dispositivo y lugar donde reside el archivo.
- Tamaño: El tamaño actual del proceso (en *bytes*, palabras o bloques).

Archivos

Los archivos poseen atributos:

- Protección: información de control para el acceso al archivo. Ej.: usuarios que pueden acceder, en qué forma, etc.
- Información de conteo: fecha de creación, último acceso, etc.

- El sistema operativo brinda servicios para la manipulación de archivos:

- Crear y abrir: provee la creación de un archivo en el sistema de archivos. Se debe proveer un nombre del nuevo archivo. Además, se provee la apertura de una archivo ya existente para acceder o modificar la información.
- Escribir: poder escribir información en un archivo previamente abierto.
- Leer: poder leer información en un archivo previamente abierto.
- Reposicionar dentro de un archivo: lograr acceder a cualquier parte del archivo.

- Eliminar: destruir el archivo a nivel del sistema de archivo.
- Truncar: eliminar la información que está dentro del archivo, pero sin eliminar el archivo.

- Por lo general, los sistemas tienen una tabla de archivos abierto por proceso. Estos archivos se abren a través de un llamado al sistema y, de esa forma, se puede operar con ellos (leer, escribir, etc.). Finalmente, el archivo es cerrado antes que finalice la ejecución del proceso.

- Tener un archivo abierto para el sistema implica mantener una estructura que tenga por lo menos: puntero de archivo (*file pointer*).

Operaciones sobre los archivos

- Tener un archivo abierto para el sistema implica mantener una estructura que tenga por lo menos: puntero de archivo (*file pointer*) para operaciones de lectura y escritura, contador de archivos abiertos, ubicación del archivo en el dispositivo, derechos de acceso.

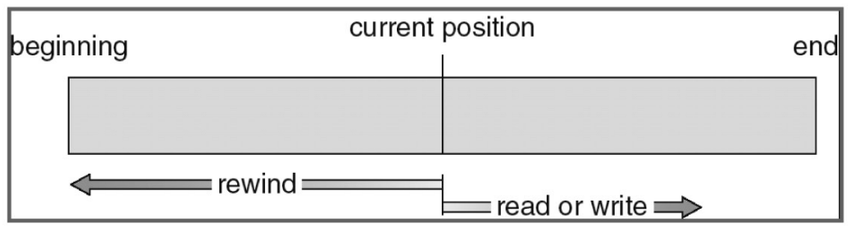
- Algunos sistemas proveen sistema de acceso único a un archivo (*lock*) por parte de los procesos.

- A su vez, varios sistemas implementan el mapeo de archivos al espacio de usuario del proceso. De esta forma no es necesario realizar *read* y *write* para operar sobre el archivo, sino accederlo directamente. Esto trae el beneficio de no hacer el llamado a sistema para operar sobre el archivo.

- Existen varios métodos de acceso a los archivos:

  - Secuencial: La información es accedida en orden, registro a registro. El registro depende del tipo de archivo (texto plano sería *byte*). Las operaciones de lectura accederán a la información en forma secuencial e incrementando el puntero de archivo *file pointer*. El acceso secuencial es basado en el modelo de cinta (*tape*), en donde los archivos son accedidos de a uno a la vez y en forma secuencial.
  - Directo: La información es accedida en cualquier orden. No existen restricciones sobre el orden de escritura y lectura de un archivo. Es basado en el modelo de disco de un archivo, que está estructurado en bloques.

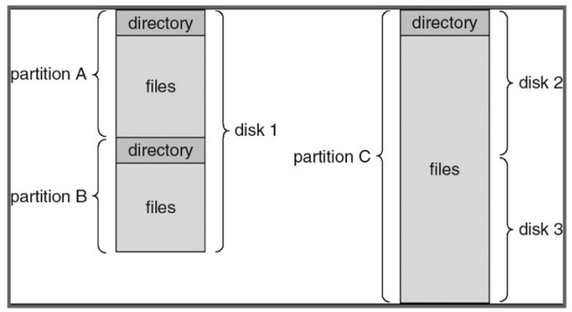
- Método secuencial:



Directorios

- El sistema de archivos es, por lo general, estructurado en directorios que contienen archivos.

- Los directorios permiten a los usuarios del sistema tener una organización lógica del sistema de archivo.

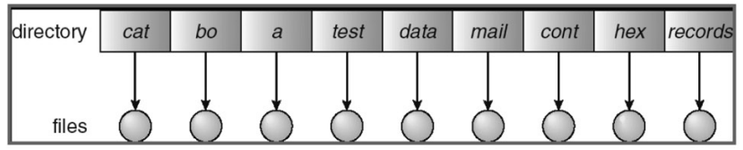


Operaciones sobre directorios

- Búsqueda: es necesario poder buscar un archivo en un directorio.
- Crear un archivo: archivos nuevos deben ser creados e incorporados al directorio.
- Eliminar: borrar un archivo del directorio.
- Listar: visualizar los archivos que están en un directorio.
- Renombrar un archivo: cambiar el nombre de un archivo dentro del directorio.
- Permitir la navegación: lograr acceder a todos los directorios del sistema de archivos.

Estructura de directorios – Nivel único

- El esquema más sencillo es tener un único nivel de directorios en el sistema de archivos.



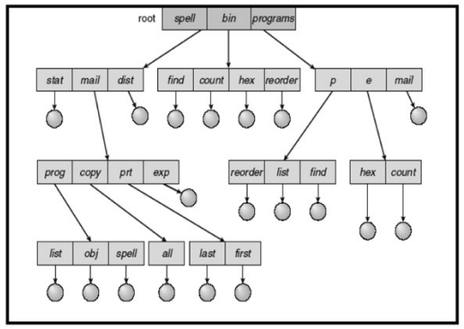
- A medida que el sistema crece trae limitaciones. Por ejemplo, no permite archivos con el mismo nombre en un mismo directorio.

Estructura de directorios - Árbol

▪ Es ideal permitir varios niveles de directorios. Esto se logra permitiendo tener archivos de tipo directorio dentro de los directorios.

•Se genera una estructura jerárquica de directorios en forma de árbol.

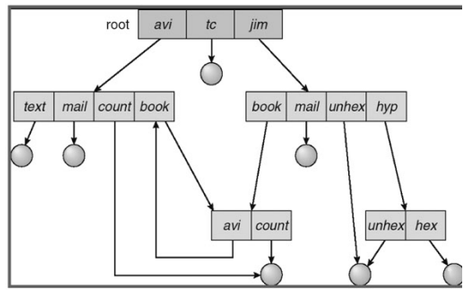
•Se denomina ruta (*path*) absoluta de un archivo al camino desde la raíz hasta el archivo.



Estructura de directorios - Grafo

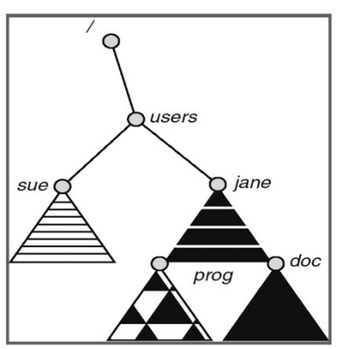
▪ Para potenciar la estructura anterior de árbol sería deseable tener caminos de acceso directo a otros directorios.

▪ Estos caminos se logran a través de archivos de tipo enlace simbólicos (***soft links***). A su vez, se permite que un archivo esté en más de un directorio (***hard link***).



Montaje de directorios

▪ Dada la estructura de grafo, los sistemas de archivos se pueden solapar en un único sistema de archivos.



Seguridad en archivos

▪ Debido a que el sistema es multiusuario es necesario proteger la información de cada usuario.

▪ En muchos casos, los usuarios se agrupan según el uso que tienen sobre un sistema.

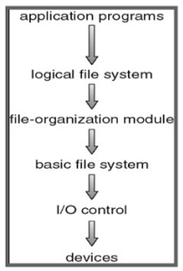
▪ Se definen permisos sobre los archivos tanto a nivel de usuario como de grupo.

▪ Los permisos más comunes son de escritura, lectura, ejecución, eliminar y listar.

Implementación

▪ El diseño de un sistema de archivo enfrenta dos problemas:
  - Como se verá, para el usuario: Implica definir los atributos, las operaciones válidas sobre los archivos y la estructura de directorios para la organización de los archivos.
  - La creación de algoritmos y estructura de datos para hacer corresponder el sistema de archivos lógico con los dispositivos físicos de datos.

▪ El sistema de archivos está compuesto de varias capas. Cada una de las cuales utiliza la funcionalidad de la capa inferior.



▪ Los dispositivos físicos (discos) contienen la siguiente estructura:

  - Bloque de control para el boot (*boot control block*): **es necesario para lograr iniciar el sistema operativo.**
  - Bloque de control de partición (*partition control block*): **contiene la información de las particiones que existen en el disco, bloques utilizados y libres, cantidad de archivos, etc.**
  - Estructura de directorios: **para la organización de los archivos.**
  - Bloque de control del archivo (*File Control Block*): **los bloques de control de los archivos que están en el sistema de archivos.**

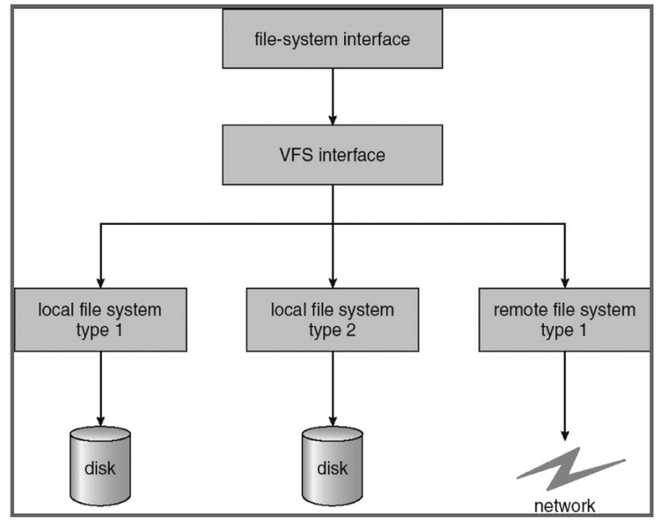
Implementación

- El sistema operativo en memoria mantiene las siguientes estructuras:
  - La tabla de partición con los sistema de archivos cargados.
  - La estructura de directorio de los accedidos últimamente.
  - Tabla de descriptores de archivos abiertos a nivel global del sistema.
  - Tabla de descriptores de archivos abiertos por proceso del sistema.
- Por cada archivo en el sistema se tendrá un bloque de control (*File Control Block*).
- El bloque de control contiene varios atributos de conteo, permisos y donde están los datos del mismo:
  - Permisos del archivo.
  - Fechas (creación, acceso, modificación).
  - Propietario, grupo propietario, lista de acceso.
  - Tamaño del archivo.
  - Bloques de datos del archivo.

Sistema de archivos virtual

- Es común que un sistema operativo acceda a más de una implementación de sistema de archivos (ufs, ext2, ext3, jfs, jfs2, ntfs, etc.).
- Se utilizan técnicas de orientación a objetos para lograr mantener una estructura independiente del sistema de archivos que se utilice.
- Se genera una estructura en tres capas:
  - Interfaz del sistema de archivo (*open*, *read*, etc.).
  - Sistema de archivos virtual (*Virtual File Server*).
  - Implementación especifica del sistema de archivo.

Sistema de archivos virtual



- El sistema de archivos virtual provee de dos funcionalidades importantes:
  - Propone una interfaz genérica de sistema de archivo que es independiente del tipo de sistema de archivo. De esta forma, se logra un acceso transparente al sistema de archivos.
  - Propone un bloque de control de archivo virtual que puede representar tanto archivos locales como remotos.

Estructura de los directorios

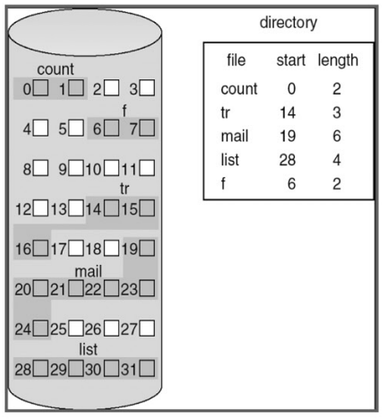
- Los directorios contienen la información de los archivos que pertenecen a él. Para organizar la información existen varias alternativas:
  - Lista encadenada: los nombres de los archivos y un puntero al bloque de control son dispuestos en una lista encadenada. En la búsqueda, inserción o borrado es necesario un acceso lineal. Es usual el uso de *caches* en memoria principal para acelerar el acceso.
  - Tabla de *hash* abierto: con el nombre del archivo se genera la clave utilizada.

Métodos de asignación

- Para la disposición de los datos de los archivos en disco se tienen, en general, tres métodos:
  - Asignación contigua (*Contiguous Allocation*): Los datos son dispuestos en forma contigua. Para mantener la información es necesario saber en qué bloque comienza y la cantidad de bloques que tiene el archivo.
  - Asignación en forma de lista (*Linked Allocation*): Los bloques de datos forman una lista encadenada. Es necesario una referencia al primer y último bloque de datos en el bloque de control de archivo.
  - Asignación indexada (*Indexed Allocation*): Se mantiene una tabla en donde cada entrada referencia a un bloque de datos.

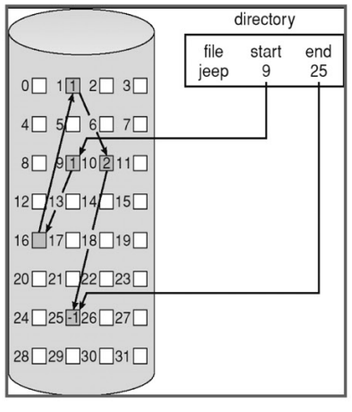
Asignación contigua (*Contiguous Allocation*)

- Sufre de fragmentación externa.
- Es necesario reubicar continuamente los archivos si crecen en tamaño.
- Se utilizan técnicas de asignación de tamaños más grandes para prever el crecimiento futuro de los archivos.



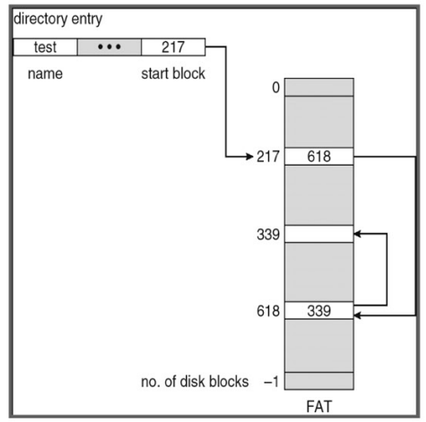
Asignación en forma de lista (*Linked Allocation*)

- Soluciona el problema de la fragmentación.
- El acceso a los bloques es lineal.
- Los punteros ocupan espacio en los bloques.
- La pérdida de una referencia genera la pérdida de gran parte de información del archivo.



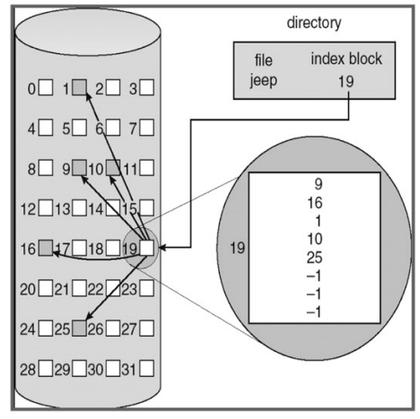
Asignación en forma de lista (*Linked Allocation*)

- Ej.: FAT
- Al comienzo de cada partición existe una tabla de asignación de archivos (*File Allocation Table*), que contiene la lista de bloques.
- La tabla tiene una entrada por cada bloque de disco, y es indexada por el número de bloque.

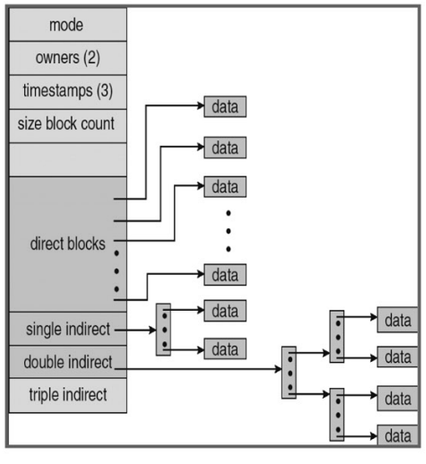


Asignación indexada (*Indexed Allocation*)

- Los bloques son accedidos directamente a través del bloque de indexación (*index block*).
- El bloque de indexación ocupa lugar. Se trata de que sea lo más pequeño posible, pero limita la cantidad de bloques.
- Una posible alternativa es indexación en varios niveles. Algunos índices hacen referencia a bloques directos y otros a bloques de indexación.



- En UNIX los bloques de control de archivos tienen bloques de indexación directa, de uno, dos y hasta de tres niveles de indexación.
- Esto permite representar archivos muy grandes.



Administración del espacio libre

- **En el sistema de archivos es necesario mantener qué bloques están ocupados y cuáles están libres.**

Alternativas posibles para la administración de los bloques:

  - Vector de bits (*Bit Vector, Bit Map*)**: se dispone de un *bit* para cada bloque de datos del sistema, que representa si está ocupado o libre.**
  - Lista de bloques libres (*Linked list*)**: Se mantiene una lista encadenada con los bloques libres a través de los bloques. Es necesario una referencia al primer bloque.**
  - Agrupación (*Grouping*)**: es una variación de la lista encadenada. En cada bloque de la lista se contiene un grupo de bloques libres.**
  - Conteo (*Counting*)**: se mantiene una lista en donde cada bloque contiene información de cuantos bloques contiguos, a partir de él, están libres.**

Ejemplo UNIX

- **Cada partición contiene un bloque descriptor del sistema de archivo llamado *super-block*.**

- **El *super-block* contiene:**
  - **Nombre del volúmen.**
  - **Cantidad máxima de archivos (inodos). Cantidad de archivos utilizados y libres.**
  - **Cantidad de bloques de datos, cantidad de bloques utilizados y libres.**
  - **Referencia a comienzo de bloques de datos, de indexación y de vector de bits.**
  - **Información de conteo.**
  - **Etc.**

- **La administración del espacio libre se realiza a través de mapa de bits (*bit vector*). Se disponen varios bloques al comienzo de la partición.**
- **El bloque de control de archivo es a la estructura *inode*. Los inodos son identificados por un número, que es único a nivel del sistema de archivos. Los inodos poseen un tipo: archivo común, directorio, enlace simbólico, *pipes* y *socket*. Utiliza un método de asignación por indexación.**
- **Los directorios son representados como un archivo (*inodo*), en donde los datos son entradas que tienen los nombres de los archivos y el número de inodo correspondiente.**

- **Si es un *soft link,* se tiene la ruta (*path*) del archivo al cual referencian. Los *hard links* son tratados en forma natural, ya que la pertenencia de un archivo a un directorio está en los datos del directorio y se referencia al número de inodo.**

8. Taller de Formación para sistemas de Información geográficos

Ing. Bruno Rienzi
Ing. Flavia Serra
Ing. Raquel Sosa

OGC Web Services

1.1 Introducción

A medida que el análisis de datos geográficos se ha ido convirtiendo en una actividad necesaria dentro de múltiples disciplinas, también se ha ido incrementando la necesidad de compartir e intercambiar esos datos. Los estándares elaborados por el Open Geospatial Consortium (OGC) [1] son un elemento fundamental para que los desarrolladores puedan crear software que permita a los usuarios acceder y procesar datos geo-espaciales de múltiples y heterogéneas fuentes utilizando un conjunto de interfaces genéricas.

Los servicios Web del OGC, denominados por su sigla OWS, están definidos utilizando principios fundamentales de una arquitectura orientada a servicios (Service Oriented Architecture, SOA). Dentro de esta arquitectura, podemos definir los conceptos de servicio, interfaz y operación de la siguiente manera:

- Un servicio es una funcionalidad que ofrece una entidad a través de interfaces.
- Una interfaz es un conjunto de operaciones que caracterizan el comportamiento de una entidad.
- Una operación es la especificación de una transformación o consulta que un objeto es capaz de ejecutar. Cada operación se caracteriza por su nombre y su lista de parámetros.

Los OWS han sido definidos en base a estándares de Internet no propietarios tales como HTTP [2], URL, tipos MIME [4] y XML [3]. Más recientemente, los OWS han comenzado a definirse utilizando también otros estándares más específicos de los Web services empresariales tales como WSDL (Web Service Description Language) y SOAP, aunque aún no se han convertido en estándares.

Dentro de los estándares de Web services definidos por OGC encontramos:

- Web Map Service (WMS), que permite la creación y visualización de mapas en base a superponer capas geográficas provenientes de múltiples fuentes remotas. [8]
- Web Feature Service (WFS), que permite que un cliente reciba y actualice datos geo-espaciales codificados en el lenguaje GML (Geography Markup Language) desde múltiples fuentes remotas. [6]
- Web Coverage Service (WCS), que permite a un cliente acceder a cierta parte de una capa raster ofrecida por el servidor (codificada en algún formato de imagen binario).
- Catalogue Service for the Web (CSW), que define interfaces para descubrir, navegar y consultar metadata sobre datos, servicios y otros recursos potenciales.

Por otro lado, también se están definiendo OWS para aplicaciones de mercado masivo utilizando Web Services REST basados en GeoRSS y KML, por ejemplo. Estas aplicaciones tienen el propósito de ampliar el uso de tecnologías basadas en la localización geográfica para el público en general.

El lenguaje KML [5] define una gramática XML para codificar y transportar representaciones de datos geográficos para desplegar en un navegador terráqueo, tal como Google Earth ¡Error! No se encuentra el origen de la referencia. El lenguaje KML fue entregado a OGC por parte de la empresa Google para ser estandarizado y extendido.

GeoRSS (Geographically Encoded Object for RSS feeds) es una propuesta para etiquetar RSS con información que permita la localización geográfica.

Los Web services son considerados clave para la implementación de arquitecturas orientadas a servicios (SOA), ya que estos permiten alcanzar un alto grado de interoperabilidad entre aplicaciones con una complejidad menor a la de otras alternativas. Los Web services están basados en protocolos para descubrimiento, descripción y llamadas remotas a servicios, así como en los conocidos protocolos como HTTP y TCP/IP.

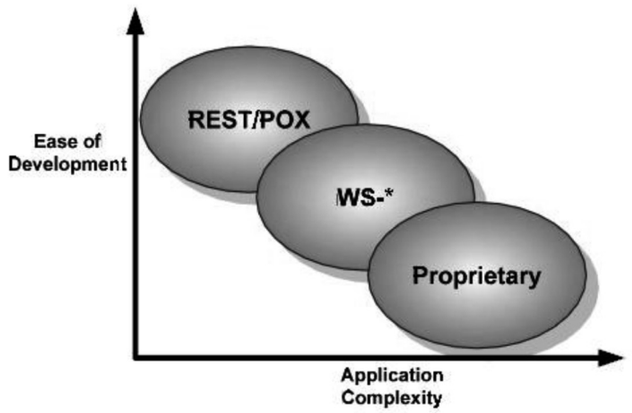
Una posible división en estándares de Web services, para los propósitos de este documento, puede ser la siguiente:

- SOAP: Estándares WS-*: los más utilizados (especialmente en aplicaciones B2B).
- POX (Plain Old XML): Muy utilizados en Internet, similares a los REST.
- REST (Representational State Transfer): API simples, muy utilizados en sitios de e-commerce.

Otros estándares de Web services basados en SOAP son los llamados Web services de segunda generación como, por ejemplo, WS-Security y BPEL (Business Process Execution Language).

Los Web services POX utilizan HTTP como protocolo de transferencia, con un contenido en XML, cuyo esquema y semántica conocen los dos hosts. Los OWS que se analizan en detalle en este documento son un ejemplo de este tipo, aunque un OWS no siempre debe soportar XML, ya que también existen variante con codificación KVP, como se verá a continuación.

Los OWS, al tener una interfaz estandarizada, permiten que sean implementados por clientes y servidores sin tener que programar cada Web Service individualmente, como sucede con Web Services SOAP. Es así que los servidores de mapas, tales como MapServer o GeoServer, permiten publicar datos geográficos mediante estándares tales como WMS o WFS, y clientes como gvSIG u OpenLayers pueden consumirlos. El usuario solo debe configurar las capas geográficas que se publican y consumen.



1.2 Implementación sobre HTTP.

La Plataforma para Computación Distribuída (DCP, Distributed Computing Platform) a la que hacen referencia los estándares de OGC es concretamente la red de hosts en Internet que soportan el protocolo HTTP (Hypertext Transfer Protocol), es decir, la World Wide Web (WWW). De esta manera, todo recurso online (Online Resource) es especificado mediante una URL HTTP (definido en el IETF RFC 2616).

HTTP soporta dos métodos para realizar solicitudes al servidor: GET y POST. Los OWS deben soportar obligatoriamente el método GET. El método POST es obligatorio para WFS pero opcional para WMS.

Cada protocolo OGC posee sus parámetros específicos y los valores válidos para cada parámetro.

HTTP GET

Un URL válido para la operación GET es un prefijo de URL al que se le concatenan parámetros. Un prefijo de URL (según el IETF RFC 2396) es una cadena de caracteres formada por el protocolo (ej. http o https), el nombre de host o dirección, el número de puerto (opcional; cada protocolo tiene su well-known port que se usa por defecto si no se especifica uno), un camino (path) y el signo de interrogación (?), seguidos opcionalmente de una lista de parámetros. La lista de parámetros se forma mediante parejas nombre/valor en la forma "nombre1=valor1&nombre2=valor2&…".

Cuando se utiliza el método GET, la codificación de la solicitud se realiza mediante KVP (Keyword-Value Pair), lo que permite construir la lista de parámetros del URL.

Ejemplo: "REQUEST=GetCapabilites", en donde REQUEST es la keyword o palabra clave y GetCapabilities el valor.

HTTP POST.

Un URL válido para la operación POST es un URL completo (no solamente un prefijo de URL, como en el caso de GET) al que los clientes envían solicitudes codificadas en el cuerpo del documento POST.

Cuando se utiliza el método POST, la codificación de la solicitud se realiza mediante el lenguaje XML (Extensible Markup Language) y no se agregan parámetros al URL.

1.3 Operación GetCapabilities

La operación GetCapabilities es soportada por todos los OWS y permite que el cliente conozca las capacidades del servidor, es decir, mediante esta operación se obtiene la metadata del servicio (ej. datos que posee, formatos que soporta, valores admitidos de los parámetros, etc.). De esta forma es posible hacer un "binding" entre un cliente y el servidor que posea la información que necesita ese cliente.

Los parámetros de una solicitud de esta operación son:

- Service: el tipo de servicio que se desea (ej. WMS). Un mismo servidor puede soportar varios servicios.
- Request: el nombre de la operación (GetCapabilities, en este caso).
- Version: el número de versión del protocolo que se está consultando. Las versiones tienen el formato x.y.z. AcceptVersions: versiones del protocolo que soporta el cliente, en orden de preferencia. El cliente y el servidor negocian la versión del protocolo. El servidor

elige una versión dentro del AcceptVersions y se la envía al cliente. Si no soporta ninguna de las versiones, envía una excepción.

• Sections: secciones de la metadata del servicio que se deben incluir en la respuesta (ServiceIdentification, ServiceProvider, OperationsMetadata, Contents, All).

• UpdateSequence: permite que el cliente averigüe si su caché es consistente con el servidor.

• AcceptFormats: formatos de respuesta (tipos MIME) que soporta el cliente, en orden de preferencia. Por omisión es "text/XML".

Ejemplo de GetCapabilities en KVP:

http://hostname:port/path?SERVICE=WCS&REQUEST=GetCapabilities&ACCEPTVERSIONS=1.0.0,0.8.3&SECTIONS=Contents&UPDATESEQUENCE=XYZ123&ACCEPTFORMATS=text/xml

Ejemplo de GetCapabilities en XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetCapabilitiesxmlns="http://www.opengis.net/ows/1.1" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/ows/1.1 fragmentGetCapabilitiesRequest.xsd" service="WCS" updateSequence="XYZ123">
<AcceptVersions><Version>1.0.0</Version><Version>0.8.3</Version></AcceptVersions>
<Sections><Section>Contents</Section></Sections>
<AcceptFormats><OutputFormat>text/xml</OutputFormat></AcceptFormats>
</GetCapabilities>
```

2 Web Map Service (WMS)

2.1 Introducción

Un Web Map Service (WMS) es un servicio que proporciona una interfaz HTTP simple para obtener un mapa georreferenciado creado dinámicamente a partir de información geográfica proveniente de una o varias fuentes de datos distribuidas. Dentro de este estándar, se entiende como "mapa" a una representación de la información geográfica en forma de un archivo de imagen digital (Ej. PNG, GIF, JPEG, SVG, etc.). A continuación de analizan en detalle todas las operaciones que comprende este protocolo.

2.2 Operaciones

WMS define dos operaciones obligatorias (GetCapabilities y GetMap) y una operación opcional (GetFeatureInfo). Una secuencia posible de envío de mensajes entre el cliente y el servidor WMS puede verse en la Figura 2.

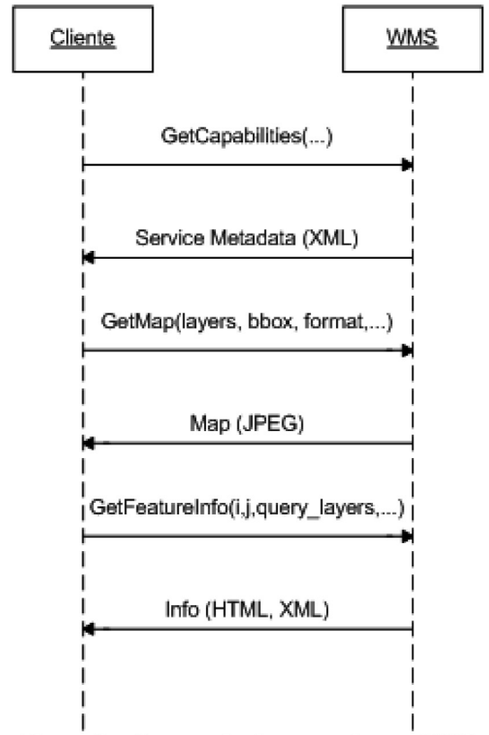


Figura 2 – Secuencia de mensajes en WMS

2.2.1 GetCapabilities

Esta operación permite obtener las capacidades del servicio. Concretamente, obtiene la metadata que describe el contenido de la información que provee el servicio, así como los valores admitidos de los parámetros con los que se realizan las solicitudes.

2.2.1.1 Solicitud

2.2.1.1.1 SERVICE

Indica el tipo de servicio (dentro de los varios tipos que puede brindar un servidor) que se va a utilizar. En este caso, se pasa el valor "WMS" para este parámetro.

2.2.1.1.2 REQUEST

Indica el nombre de la operación (dentro de las operaciones que definen el servicio especificado en el parámetro SERVICE) que se va a invocar. En este caso, se pasa el valor "GetCapabilities" para este parámetro.

2.2.1.1.3 FORMAT

Indica el formato (tipo MIME) en el que se desee recibir la respuesta. El valor por defecto es text/xml.

2.2.1.1.4 UPDATESEQUENCE

Permite que el cliente averigüe si su cache es consistente con el servidor.

2.2.1.1.5 VERSION

Permite negociar la versión del protocolo a utilizar entre el cliente y el servidor. La versión del protocolo WMS, que abarca el esquema XML y la codificación de las solicitudes, está determinado por tres enteros positivos en el rango de 0 a 99 separados por puntos (x.y.z). La versión actual, por ejemplo, es la 1.3.0. La negociación de la versión entre cliente y servidor se realiza mediante una conversación solicitudes GetCapabilites y respuestas del servidor.

2.2.1.2 Respuesta

La respuesta de la solicitud GetCapabilites es un documento XML conforme al esquema XML de la versión negociada entre el cliente y el servidor. El documento contiene un elemento raíz WMS_Capabilities en el namespace http://www.opengis.net/wms.

2.2.1.2.1 Service

La primera parte de la metadata está constituida por un elemento <Service> que proporciona los datos generales del servicio. Los datos obligatorios son Name, Title y OnlineResource. Los datos opcionales son Abstract, Keyword, List, Contact Information, Fees, Access Constraints y límites en la cantidad de capas que puede componer en un mapa, o límite en el tamaño del mapa retornado.

- **Name**

El elemento Name indica el nombre del servico, que en este caso es siempre "WMS".

- **Title**

El elemento Title es un título que permita describir al proveedor.

- **OnlineResource**

El elemento OnlineResource permite referir al sitio Web del proveedor del servicio.

2.2.1.2.2 LayerLimit

El elemento <LayerLimit> es un entero positivo que indica la cantidad máxima de capas que pueden solicitarse en una operación GetMap (Ver 2.2.2.1.3).

2.2.1.2.3 MaxWidth y MaxHeight (opcionales)

Los elementos <MaxWidth> y <MaxHeight> indican el tamaño máximo en pixeles del mapa que se puede solicitar mediante los parámetros WIDTH y HEIGHT de la operación GetMap (ver 2.2.2.1.8).

2.2.1.2.4 Capability

El elemento <Capability> enumera las operaciones que son soportadas por el servidor, los formatos de respuesta de esas operaciones y el prefijo de URL de cada una.

2.2.1.2.5 Layer

La información geográfica ofrecida por un WMS está organizada en capas o "layers": cada capa es descripta individualmente por su propia metadata. Los clientes realizan las solicitudes de un mapa en base a estas capas. Cada servicio debe proporcionar al menos un elemento <Layer>. Si bien cada capa representa una entidad diferente, los elementos <Layer> se organizan en forma jerárquica (cada <Layer> puede tener otros <Layer> hijos). Esto permite que los elementos hijos "hereden" propiedades de su elemento padre y así se reduzca el tamaño de la metadata necesaria.

- **Title**

El elemento <Title> es un título que describe a la capa. No es heredado por los hijos.

- **Name**

El elemento <Name> es un nombre de la capa que puede ser utilizado como parámetro LAYERS de la operación GetMap. Si una capa posee un título pero no un nombre, entonces actúa solo como una agrupación formal de las capas anidadas, y no debería ser solicitada por los clientes. Las capas con hijos también pueden poseer un nombre. En este caso, obtener la capa por su nombre equivale a obtener todas las subcapas juntas. El elemento <Name> no se hereda de padres a hijos.

- **Abstract**

El elemento opcional <Abstract> contiene una descripción de la capa.

- **KeywordList**

El elemento opcional <KeywordList> posee una secuencia (eventualmente vacía) de elementos <Keyword> con palabras clave para ser utilizadas en búsquedas de catálogos.

- **Style**

El elemento <Style> permite definir un posible estilo de presentación de una capa. Dentro de los sub-elementos cabe destacar <LegendURL> que contiene la URL de la leyenda que corresponde a dicho estilo.

- **EX_GeographicBoundingBox**

El elemento <EX_GeographicBoundingBox> define el rectángulo mínimo en grados decimales del area abarcada por la capa. Las coordenadas se definen mediante los elementos <westBoundLongitude>, <eastBoundLongitude>, <southBoundLatitude> y <northBoundLatitude>. Si la capa no se encuentra en coordenadas geográficas, las coordenadas de este rectángulo pueden no ser exactas, ya que sólo se busca facilitar las búsquedas geográficas sin exigir que el motor de búsqueda deba realizar transformaciones de coordenadas.

- **CRS**

El elemento <CRS> de una capa define el sistema de referencia coordenado (Coordinate Reference System) de la capa. Cada capa debe incluir todos los CRS que son comunes a la capa y a todas sus sub-capas.

- **BoundingBox**

El elemento <BoundingBox>, define el rectángulo mínimo que contiene la capa, al igual que <EX_GeographicBoundingBox>, pero a diferencia del anterior, las coordenadas se especifican en un determinado CRS. El EX_GeographicBoundingBox puede verse como un BoundingBox en el atributo CRS="CRS:84" está implícito. Sin embargo, el elemento <EX_GeographicBoundingBox> no debe utilizarse como sustituto de <BoundingBox CRS="CRS:84"> ya que sirven a propósitos diferentes.

Los atributos de un BoundingBox son el CRS, las coordenadas (minx, miny, maxx, maxy) en las unidades especificadas por el CRS y, opcionalmente, resx y resy que indican la resolución de la capa en esas unidades.

Los BoundingBox de una capa se heredan a la sub-capa. Los BoundingBox de una subcapa se agregan a los que hereda de su capa madre.

Una capa no debe proveer un BoundingBox en un CRS que no soporta.

Se debe proveer al menos de un BoundingBox en el CRS nativo de la capa (en el que están almacenados los datos).

No es una exigencia que servidor provea un BoundingBox para cada CRS al que pueda transformar los datos nativos.

Si el CRS es "CRS:1", entonces las unidades están en pixels, el origen de coordenadas es en el vértice superior izquierdo, el sentido positivo de la abscisa es hacia la derecha y el de la ordenada es hacia abajo.

- **ScaleDenominator**

Los elementos <MinScaleDenominator> y <MaxScaleDenominator> permiten definir el rango de escalas en el que es conveniente generar el mapa de una capa.

Por ejemplo, para indicar una escala mayor o igual a 1:1000 y menor a 1:1000000, se utilizan las siguientes definiciones:

<MinScaleDenominator>1e3</MinScaleDenominator>

<MaxScaleDenominator>1e6</MaxScaleDenominator>

<MinScaleDenominator> siempre incluye el valor de borde ("mayor o igual"), mientras que <MaxScaleDenominator> no lo incluye ("menor estricto").

Ambos son elementos opcionales, por lo que la escala puede ser limitada solo en sentido.

Los límites de escala proporcionados de esta manera deben interpretarse como una guía para los clientes, no como límites estrictos. Las unidades de la escala son pixels, ya que el tamaño del píxel en unidades de distancia depende del dispositivo en el que se muestre la imagen. El tamaño de píxel estándar se toma como 0,28 mm x 0,28 mm.

- **Dimension**

El elemento opcional <Dimension> proporciona la metadata sobre datos-multidimensionales. Esta información refiere a dimensiones que se encuentran por fuera de las cuatro dimensiones espacio-temporales habituales, por ejemplo, las bandas de longitud de onda en una imagen satelital. Los elementos <Dimension> se heredan.

- **MetadataURL**

El elemento <MetadataURL> se utiliza para brindar información de la capa utilizando un estándar de metadata. Su atributo type puede tomar los valores "ISO 19115:2003" o "FGDC:1998". Los elementos <MetadataURL> no se heredan.

- **Attribution**

El elemento <Attribution> permite identificar la fuente de donde provienen los datos de una capa, proporcionando información como la URL del proveedor, un título, un logo, etc. Los elementos <Attribution> se heredan.

- **Identifier y AuthorityURL**

Los elementos <Identifier> y <AuthorityURL> se utilizan en conjunto para definir qué valor de un identificador externo posee una capa. Por ejemplo, si una cierta organización identifica sus capas mediante un cierto identificador, con el elemento <AuthorityURL> podemos referenciar esa organización y con el <Identifier> declaramos el valor del identificador que utiliza esa organización. Los elementos <AuthorityURL> se heredan, los <Identifier> no.

- **FeatureListURL**

El elemento <FeatureListURL> se utiliza para referenciar la URL en donde se listan las características geográficas pertenecientes a la capa. El sub-elemento <Format> define el tipo MIME para leer esa lista. El elemento <FeatureListURL> no se hereda.

- **DataURL**

El elemento <DataURL> permite ofrecer el link al archivo de datos representados por la capa. El sub-elemento <Format> define el tipo MIME para leer ese archivo. El elemento <DataURL> no se hereda.

- **queryable**

El atributo booleano queryable indica si el servidor soporta la operación GetFeatureInfo en esta capa.

- **cascaded**

El atributo entero cascaded indica que la capa se obtuvo desde otro servidor diferente al servidor que envía la metadata (el servidor actual). El servidor actual puede estar actuando como otro punto de acceso más a esa capa, o puede ofrecer valor adicional a lo que ofrece el servidor original, por ejemplo, con más formatos de salida o con la capacidad de reproyectar los datos a otros CRS (Ver). El valor del atributo cascaded es 0 (o se omite el atributo) en el servidor original y por cada servidor que obtiene de otro se va incrementando en 1.

- **opaque**

El atributo booleano opaque se utiliza para marcar capas que cubren completamente o casi completamente el mapa sin dejar espacios tranparentes, independientemente de la escala utilizada (es el caso típico de rasters y capas de polígonos, pero no de capas de líneas o puntos). El atributo opaque sirve como una indicación al cliente para que coloque esta capa por debajo de las demás.

- **noSubsets**

El atributo noSubsets indica que el servidor no puede crear un mapa de una porción de la capa (un subconjunto de sus características) sino solo de su BoundingBox.

- **fixedWidth y fixedHeight**

Los atributos fixedWidth y fixedHeight indican que el servidor solo puede proporcionar un mapa con ese ancho y alto en pixels.

- **2.2.2 GetMap**

Esta operación permite solicitar un mapa. El WMS debe devolver el mapa solicitado o una excepción.

2.2.2.1 Solicitud.

2.2.2.1.1 VERSION

La versión del protocolo a utilizar.

2.2.2.1.2 REQUEST

Indica el nombre de la operación (dentro de las operaciones que definen el servicio especificado en el parámetro SERVICE) que se va a invocar. En este caso, se pasa el valor "GetMap" para este parámetro.

2.2.2.1.3 LAYERS

Indica las capas que deben componerse en el mapa resultante. El valor de este parámetro es una lista de nombres de capas separados por comas. Un nombre de capa es el contenido de un elemento <Layer><Name>. Las capas se dibujan empezando con la que está más a la izquierda en la lista.

2.2.2.1.4 STYLES

Indica los estilos en los que deben mostrarse las capas. El valor de este parámetro es una lista de nombres de estilos separados por comas. Un nombre de estilo es el contenido de un elemento <Style><Name>. El estilo de la posición N de la lista se asocia a la capa de la posición N de la lista del parámetro LAYERS. El estilo debe estar contenido en la definición de la capa o ser heredado por esta. Si para una capa se quiere utilizar el estilo por defecto:

Ejemplo 1: Para capa2 no se especifica estilo (se utilizará el estilo por defecto).
"LAYERS=capa1,capa2,capa3"
"STYLES=estilo1,,estilo3"

Ejemplo 2: Para capa3 no se especifica estilo (se utilizará el estilo por defecto).
"LAYERS=capa1,capa2,capa3"
"STYLES=estilo1,estilo2,"
Ejemplo 3: Para ninguna capa se especifica estilo (se utilizará el estilo por defecto).
"LAYERS=capa1,capa2,capa3"
"STYLES=,," o "STYLES="

2.2.2.1.5 CRS

Indica el CRS que corresponde al parámetro BBOX. El CRS debe estar soportado para esa capa.

2.2.2.1.6 BBOX

Indica el Bounding Box que se quiere obtener del mapa. El valor de este parámetro es una lista de cuatro números reales separados por comas (minx,miny,maxx,maxy) que definen las coordenadas del rectángulo. Las unidades, orden y dirección de incremento de la abscisa y la ordenada dependen del CRS especificado. El area especificada en el BBOX debe superponerse en parte con el BoundingBox devuelto por GetCapabilities. Si la capa está marcada por el atributo noSubsets, las coordenadas de BBOX deben coincidir con las de BoundingBox devuelto por GetCapabilities.

2.2.2.1.7 FORMAT

Indica el formato en que será devuelto el mapa. El valor es cualquier tipo MIME que se encuentre dentro de los elementos<Request><GetMap><Format> de la metadata de servicio.

2.2.2.1.8 WIDTH, HEIGHT

Indican el ancho y alto en pixels de mapa. Si la relación de aspecto (ancho:alto) determinada por estos parámetros no coincide con la determinada por BBOX, el mapa será deformado (utilizando pixels rectangulares en lugar de cuadrados) para que el BBOX ocupe exactamente ese tamaño. Los valores de WIDTH y HEIGHT están limitados por los valores <MaxWidth> y <MaxHeight> devueltos por GetCapabilities. Si una capa está marcada con atributos fixedWith y fixedSize, solo podrán utilizarse esos valores de ancho y alto.

2.2.2.1.9 TRANSPARENT

Indica si el fondo del mapa debe ser transparente o no. Los valores posibles son "TRUE" o "FALSE". El fondo del mapa lo componen los píxels que no representan ninguna característica geográfica. El formato de imagen especificado en el parámetro FORMAT podría no soportar transparencia. Si se especifica el valor "FALSE", el fondo se pinta del color especificado en el parámetro BGCOLOR.

2.2.2.1.10 BGCOLOR

Indica el color a utilizarse en el fondo del mapa. El valor del parámetro es una valor hexadecimal de la forma 0xRRGGBB en donde se utilizan dos caracteres para cada canal (rojo, verde, azul), en el rango de 00 a FF (0-255 en decimal).

2.2.2.1.11 EXCEPTIONS

Indica el formato en el que se recibirán las excepciones. El valor por defecto es "XML".

2.2.2.1.12 TIME

En capas que poseen la dimensión del tiempo, permite obtener los datos para un determinado valor de la variable tiempo.

2.2.2.1.13 ELEVATION

En capas que poseen la dimensión de elevación, permite obtener los datos para un determinado valor de la variable elevación.

2.2.2.2 Respuesta

La respuesta a la solicitud GetMap es un mapa en el formato de imagen especificado, formado por las capas solicitadas y con todas las propiedades y restricciones determinadas por los parámetros de la solicitud.

2.2.3 GetFeatureInfo

La operación GetFeatureInfo está soportada para aquellas capas cuyo atributo queryable tenga valor "1" (definido directamente o heredado). El caso de uso canónico de esta funcionalidad es cuando el usuario está viendo un mapa (obtenido mediante GetMap) y elige un punto del que desea obtener más información. Dado que WMS es un protocolo sin estado (stateless) que no guarda información de la sesión, además de los parámetros específicos de esta operación es necesario incluir parámetros de la operación GetMap (BBOX, CRS, WIDTH, HEIGHT).

2.2.3.1 Solicitud

2.2.3.1.1 QUERY_LAYERS

Este parámetro indica (mediante un lista de nombres separada por comas) las capas geográficas que serán consideradas para devolver la información de las características que contengan el (o estén próximas al) punto especificado mediante (I,J). Estas capas deben ser un subconjunto de las capas especificadas en el parámetro LAYERS de GetMap.

2.2.3.1.2 INFO_FORMAT

El parámetro INFO_FORMAT indica el formato en que se generará la respuesta para devolver la información de la característica. Los valores válidos son los tipos MIME que se obtienen mediate el método GetCapabilities en los elementos <Request><FeatureInfo><Format>, por ejemplo, text/xml.

2.2.3.1.3 I,J

Los parámetros I y J son las coordenadas de pantalla (medidas en pixels) que identifican el punto de interés en el mapa. Estas coordenadas van desde 0 hasta el valor de WIDTH y HEIGHT del mapa correspondiente.

2.2.3.2 Respuesta

La respuesta contiene las características que el servidor considera que están relacionas con el punto dado. El criterio de relacionamiento queda a discreción de cada implementación. Por ejemplo, en una capa de polígonos, normalmente se devolvería el plolígono que contega al punto. En una capa de líneas, la(s) línea(s) cuya(s) distancia(s) al punto sea mínima, etc. El formato de la respuesta queda determinado por el parámetro INFO_FORMAT de la solicitud.

3 Web Feature Service (WFS)

3.1 Introducción

WFS define una interfaz para especificar las operaciones de altas, bajas y modificaciones (ABM) de las entidades geográficas (geographical features) codificadas en el lenguaje GML [7], que es un esquema de XML para representar información geográfica. El estándar define algunas operaciones como obligatorias y otras como opcionales. A continuación se detallan las principales operaciones (algunas obligatorias y otras opcionales) de este protocolo. Sólo se omite la operación LockFeature, que es la operación opcional que permite bloquear una característica de un WFS transaccional. Operaciones.

3.1.1 DescribeFeatureType

La función de DescribeFeatureType es generar una descripción de esquema de los tipos de características presentes. Este esquema define como las instancias de una característica deben ser codificadas en una entrada (mediantes un Insert o un Update) o cómo serán generadas en una salida (en respuesta a un GetFeature o un GetGmlObject). Si el contenido del elemento DescribeFeatureType está vacío, el servidor deberá generar la descripción de todos sus tipos.

3.1.1.1 Solicitud

El siguiente fragmento del esquema XML define las solicitudes de este tipo:

```
<xsd:element name=”DescribeFeatureType” type=”wfs:DescribeFeatureTypeType”/>
<xsd:complexType name=”DescribeFeatureTypeType”>
<xsd:complexContent>
<xsd:extension base=”wfs:BaseRequestType”>
<xsd:sequence>
<xsd:element name=”TypeName” type=”xsd:QName”
minOccurs=”0” maxOccurs=”unbounded”/>
</xsd:sequence>
<xsd:attribute name=”outputFormat”
type=”xsd:string” use=”optional”
default=”text/xml; subtype=gml/3.1.1”/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

3.1.1.1.1

3.1.1.1.2 TypeName

El elemento TypeName especifica el nombre de un tipo de característica del que se quiere obtener la descripción.

3.1.1.1.3 outputFormat

El atributo outputFormat indica el tipo MIME del lenguaje de descripción del esquema que se utilizarán en la respuesta. El valor por defecto es text/xml;subtype=gml/3.1.1, que indica el esquema de aplicación GML3. Además de éste, todos los tipos MIME válidos se obtienen a través del método GetCapabilities.

Es importante destacar que la definición del esquema de características es responsabilidad de cada implementación WFS particular, siendo las siguientes las únicas restricciones:

- La geometría de las características debe estar expresada en GML (gml.xsd).
- El sistema de referencia coordenado (CRS) debe ser expresado en GML.
- El esquema de características debe ser consistente con el modelo de características de OGC. Por ejemplo, dentro de este modelo, se considera que cada elemento contenidos inmediatamente por el elemento raíz de una característica es una propiedad de esa característica (en otra interpretación, inconsistente con el modelo mencionado, podría considerarse que el elemento interior es una especialización o sub-tipo de la característica que lo contiene, por ejemplo).

3.1.1.2 Respuesta

La respuesta de esta operación, es la descripción de esquema de las características en el formato solicitado. Si el formato es GML3, será un esquema de aplicación GML con un subesquema por cada tipo de característica solicitado.

Dado que un esquema XML solo puede describir elementos que pertenecen a un mismo espacio de nombres, no es posible describir los esquemas de tipos de elementos que pertenecen a diferentes espacios. En este caso, la respuesta es un esquema XML en donde se utilizan elementos <import>.

Por ejemplo, dada la siguiente solicitud:

```
<?xml version=”1.0” ?>
<DescribeFeatureType
version=”1.1.0”
service=”WFS”
xmlns=”http://www.opengis.net/wfs”
xmlns:ns01=”http://www.server01.com/ns01”
xmlns:ns02=”http://www.server02.com/ns02”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:schemaLocation=”http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd”>
<TypeName>ns01:TreesA_1M</TypeName>
<TypeName>ns02:RoadL_1M</TypeName>
</DescribeFeatureType>
```

La respuesta podría tener esta forma:

```
<?xml version=”1.0” ?>
<schema
targetNamespace=”http://www.someserver.com/myns”
xmlns:myns=http://www.someserver.com/myns
xmlns=”http://www.w3.org/2001/XMLSchema”
elementFormDefault=”qualified”
attributeFormDefault=”unqualified”>
<import namespace=”http://www.server01.com/ns01”
schemaLocation=”http://www.myserver.com/wfs.cgi?
request=DescribeFeatureType&amp;typeName=ns01:TreesA_1M”/>
<import namespace=”http://www.server02.com/ns02”
schemaLocation=”http://www.yourserver.com/wfs.cgi?
request=DescribeFeatureType&amp;typeName=ns02:RoadL_1M”/>
</schema>
```

3.1.2 GetFeature

3.1.2.1 Solicitud

El siguiente fragmento del esquema XML define las solicitudes de este tipo:

```
<xsd:element name=”GetFeature” type=”wfs:GetFeatureType”/>
<xsd:complexType name=”GetFeatureType”>
<xsd:complexContent>
<xsd:extension base=”wfs:BaseRequestType”>
<xsd:sequence>
<xsd:element ref=”wfs:Query” maxOccurs=”unbounded”/>
</xsd:sequence>
<xsd:attribute name=”resultType”
type=”wfs:ResultTypeType” use=”optional”
Consejo de Educación Técnico Profesional Servicios Geográficos
- 16 -
default=”results”/>
<xsd:attribute name=”outputFormat”
type=”xsd:string” use=”optional”
default=”text/xml; subtype=3.1.1”/>
<xsd:attribute name=”maxFeatures”
type=”xsd:positiveInteger” use=”optional”/>
<xsd:attribute name=”traverseXlinkDepth”
type=”xsd:string” use=”optional”/>
<xsd:attribute name=”traverseXlinkExpiry”
type=”xsd:positiveInteger”
use=”optional”/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name=”ResultTypeType”>
<xsd:restriction base=”xsd:string”>
<xsd:enumeration value=”results”/>
<xsd:enumeration value=”hits”/>
</xsd:restriction>
</xsd:simpleType>
<xsd:element name=”Query” type=”wfs:QueryType”/>
<xsd:complexType name=”QueryType”>
<xsd:sequence>
<xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
<xsd:element ref=”wfs:PropertyName”/>
<xsd:element ref=”ogc:Function”/>
</xsd:choice>
<xsd:element ref=”ogc:Filter” minOccurs=”0” maxOccurs=”1”/>
<xsd:element ref=”ogc:SortBy” minOccurs=”0” maxOccurs=”1”/>
</xsd:sequence>
<xsd:attribute name=”handle”
type=”xsd:string” use=”optional”/>
<xsd:attribute name=”typeName”
type=”wfs:TypeNameListType” use=”required”/>
<xsd:attribute name=”featureVersion”
type=”xsd:string” use=”optional”/>
<xsd:attribute name=”srsName” type=”xsd:anyURI” use=”optional”/>
</xsd:complexType>
<xsd:simpleType name=”Base_TypeNameListType”>
<xsd:list itemType=”QName”/>
</xsd:simpleType>
```

```
<xsd:simpleType name=”TypeNameListType”>
<xsd:restriction base=”wfs:Base_TypeNameListType”>
<xsd:pattern value=”((\w:)?\w(=\w)?){1,}”/>
   </xsd:restriction>
   </xsd:simpleType>
```

3.1.2.1.1 Query

El elemento <Query> define qué tipos de características se quiere consultar, qué propiedades de los mismos se desea obtener y qué restricciones (espaciales y no-espaciales) se deben aplicar para realizar la selección. Los resultados de todas las consultas especificadas en la solicitud son concatenadas para producir el resultado (un conjunto de características). A continuación se describen los principales atributos de este elemento.

- **typeName**

El atributo typeName es utilizado para indicar el nombre de una o más instancias de un tipo de característica. Al especificar una lista de nombres separados por coma, se interpreta como una operación de JOIN entre las instancias de estos tipos.

Por ejemplo:

```
typeName=”ns1:Ciudad=A,ns1:Ciudad=B,ns2:CiudadCostera”
```

especifica que se realizará un JOIN entre tres tipos, a los que se les definen los alias A, B y C. Dato que A y B son alias del mismo tipo (Ciudad en el espacio de nombres ns1), se realiza un SELF-JOIN en el tipo Ciudad.

- **PropertyName**

El elemento <PropertyName> tiene como contenido el nombre de un elemento (calificado por su espacio de nombres) que representa un propiedad de una característica (ej. ns1:direccion). Estos elementos se incluyen para que las características devueltas por la consulta incluyan estas propiedades. En el caso de las propiedades que están representados por elementos obligatorias en el esquema de definición de la característica, no es necesario solicitarlas explícitamente mediante elementos <PropertyName>, ya que esas propiedades vienen necesariamente siempre que se solicita esa característica para que el documento XML sea válido contra su esquema.

- **Filter**

El elemento <Filter> se utiliza para imponer restricciones en una consulta.

- **srsName**

El atributo srsName indica el SRS (Spatial Reference System) en que se solicita que se devuelvan los datos.

3.1.2.1.2 outputFormat.

El atributo outputFormat indica el tipo MIME del lenguaje de descripción del esquema que se utilizarán en la respuesta. El valor por defecto es text/xml;subtype=gml/3.1.1 que indica que se generará un documento GML3 que puede ser validado contra el esquema de aplicación GML3 generado como respuesta de la operación DescribeFeatureType (ver 3.1.1). Además de éste, todos los tipos MIME válidos se obtienen a través del método GetCapabilities.

3.1.2.1.3 resultType

El atributo resultType puede tomar dos valores que establecen qué tipo de respuesta se espera del servicio. Si el valor es Results el servicio deberá devolver la descripción de todas las características que cumplan con las consultas de la solicitud; éste es el mismo comportamiento que se obtiene si omite este atributo. Si el valor es Hits el servicio deberá devolver únicamente el número de características que cumplan con las consultas de la solicitud.

3.1.2.1.4 maxFeatures

El atributo maxFeatures se utiliza para limitar el número de características solicitadas en forma explícita (via GetFeature/Query/@typeName).

3.1.2.2 Respuesta

La respuesta de esta operación está definida por este fragmento del esquema XML:

```
<xsd:element name=”FeatureCollection”
type=”wfs:FeatureCollectionType”
substitutionGroup=”gml:_FeatureCollection”/>
<xsd:complexType name=”FeatureCollectionType”>
<xsd:complexContent>
<xsd:extension base=”gml:AbstractFeatureCollectionType”>
<xsd:attribute name=”lockId” type=”xsd:string” use=”optional”/>
<xsd:attribute name=”timeStamp” type=”xsd:dateTime” use=”optional”/>
<xsd:attribute name=”numberOfFeatures”
type=”xsd:nonNegativeInteger”
use=”optional”/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

El elemento <FeatureCollection> es el elemento raíz de la respuesta. En el caso que resultType en la solicitud tenga el valor results, el contenido de este elemento serán todas las características recuperadas. En el caso que se hits, el contenido será vacío, pero se llenerán los atributos timeStamp y numberOfFeatures.

3.1.2.3 Ejemplo

El siguiente ejemplo muestra la utilización de la operación GetFeature para obtener las instancias de las características Calle y Via que se encuentre dentro de una cierta región. Para esto se utiliza el operador Within como filtro, especificando las cordenadas de los puntos superior izquierdo e inferior derecho de la región.

```
<?xml version="1.0" ?>
<GetFeature
version="1.1.0"
service="WFS"
handle="Example Query"
xmlns="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:gml="http://www.opengis.net/gml"
xmlns:myns="http://www.someserver.com/myns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd">
<Query typeName="myns:Calle">
<wfs:PropertyName>myns:path</wfs:PropertyName>
<wfs:PropertyName>myns:lanes</wfs:PropertyName>
<wfs:PropertyName>myns:surfaceType</wfs:PropertyName>
<ogc:Filter>
<ogc:Within>
<ogc:PropertyName>myns:path</ogc:PropertyName>
<gml:Envelope srsName="EPSG:63266405">
<gml:lowerCorner>50 40</gml:lowerCorner>
<gml:upperCorner>100 60</gml:upperCorner>
</gml:Envelope>
</ogc:Within>
</ogc:Filter>
</Query>
<Query typeName="myns:Via">
<wfs:PropertyName>myns:track</wfs:PropertyName>
<wfs:PropertyName>myns:gauge</wfs:PropertyName>
<ogc:Filter>
<ogc:Within>
<ogc:PropertyName>myns:track</ogc:PropertyName>
<gml:Envelope srsName="…">
<gml:lowerCorner>50 40</gml:lowerCorner>
<gml:upperCorner>100 60</gml:upperCorner>
</gml:Envelope>
</ogc:Within>
</ogc:Filter>
</Query>
</GetFeature>
```



```
<?xml version="1.0" ?>
<wfs:FeatureCollection
xmlns="http://www.someserver.com/myns"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd
http://www.someserver.com/myns ROADSRAILS.xsd">
<gml:boundedBy>
<gml:Envelope srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
<gml:lowerCorner>0 0</gml:lowerCorner>
<gml:upperCorner>180 360</gml:upperCorner>
</gml:Envelope>
</gml:boundedBy>
<gml:featureMember>
<Calle gml:id="Calle.100">
<path>

<gml:LineString gid="1"
srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
<gml:posList>10 10 10 11 10 12 10 13</gml:posList>
</gml:LineString>
</path>
<surfaceType>ASPHALT</surfaceType>
<nLanes>4</nLanes>
</Calle>
</gml:featureMember>
<gml:featureMember>
<Calle gml:id="Calle.105">
<path>
<gml:LineString gid="2"
srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
<gml:posList>10 10 10 11 10 12</gml:posList>
</gml:LineString>
</path>
<surfaceType>GRAVEL</surfaceType>
<nLanes>2</nLanes>
</Calle>
</gml:featureMember>
<gml:featureMember>
<Via gml:id="Via.119">
<track>
<gml:LineString gid="n"
srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
<gml:posList>15 10 16 11 17 12</gml:posList>
```

```
</gml:LineString>
</track>
<gauge>24</gauge>
</Via>
</gml:featureMember>
</wfs:FeatureCollection>
```

3.1.3 GetGmlObject

Esta operación permite obtener cualquier objeto GML a través de su identificador único (gml:id). Los objetos pueden ser características, geometrías, topologías, etc. Si el elemento solicitado posee XLinks a otros elementos, estos otros elementos se obtienen mediante solicitudes GetGmlObject recursivas, las que posiblemente se envíen desde el WFS actual a otro WFS remoto. Para saber a dónde enviar la solicitud recursiva, se utiliza el valor del atributo xlink:href, de donde se obtiene la URL del WFS remoto y el identificador único del elemento referenciado.

3.1.3.1 Solicitud

El siguiente fragmento de esquema XML define las solicitudes de este tipo:

```
<xsd:element name=”GetGmlObject” type=”wfs:GetGmlObjectType”/>
<xsd:complexType name=”GetGmlObjectType”>
<xsd:complexContent>
<xsd:extension base=”wfs:BaseRequestType”>
<xsd:sequence>
<xsd:element ref=”ogc:GmlObjectId”/>
</xsd:sequence>
<xsd:attribute name=”outputFormat”
type=”xsd:string” use=”optional” default=”GML3”/>
<xsd:attribute name=”traverseXlinkDepth”
type=”xsd:string” use=”required”/>
<xsd:attribute name=”traverseXlinkExpiry”
type=”xsd:positiveInteger”
use=”optional”/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

3.1.3.1.1 GmlObjetcId

Consejo de Educación Técnico Profesional Servicios Geográficos.

El elemento <GmlObjetId> se utiliza para especificar el identificador único (el gml:id) del elemento que se quiere obtener. Todos los objetos GML poseen un identificador gml:id como atributo.

3.1.3.1.2 outputFormat

El atributo outputFormat indica el tipo MIME del formato que se utilizará en la respuesta. El valor por defecto es text/xml;subtype=gml/3.1.1 que indica que se generará un documento. Además de éste, todos los tipos MIME válidos se obtienen a través del método GetCapabilities.

3.1.3.1.3 traverseXlinkDepth

El atributo traverseXlinkDepth indica la profundidad de búsqueda de elementos referenciados mediante XLinks. Por ejemplo, si este atributo tiene el valor “1”, solo el elemento solicitado será devuelto sin agregar elementos anidados que sean vinculados por XLinks. En cambio, si tiene valor “2”, se tratará de resolver el primer nivel de anidamiento de elementos, enviando solicitudes GetGmlObject recursivas pero con un nivel menos de profundidad (“1”). Además de valores enteros positivos, se utiliza el literal “*” para indicar una profundidad de búsqueda ilimitada.

3.1.3.1.4 traverseXlinkExpiry

El atributo traverseXlinkExpiry, especificado en minutos, indica el tiempo máximo que un WFS que ha recibido un GetGmlObject debe esperar por la respuesta de otro WFS al que ha enviado otro GetGmlObject para poder resolver un elemento referenciado mediante XLinks.

3.1.3.2 Respuesta

La respuesta a una solicitud GetGmlObject es un elemento GML devuelto como un fragmento de documento XML. Esto se diferencia de la respuesta de solicitud GetFeature que es un documento XML completo.

El contenido de la respuesta es afectado por los valores de las propiedades traverseXlinkDepth y traverseXlinkExpiry, así como de la capacidad del WFS actual de resolver elementos vinculados mediante XLinks enviando solicitudes GetGmlObject a otros WFS.

Los Xlinks que son resueltos se mantienen en la respuesta en su forma original dentro de comentarios XML (encerrados entre <!-- y -->).

3.1.3.3 Ejemplo

En el siguiente ejemplo, se solicita el elemento con identificador único “t1” y con una profundidad “1”.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<wfs:GetGmlObject
xmlns:wfs=”http://www.opengis.net/wfs”
xmlns:ogc=”http://www.opengis.net/ogc”
xmlns:gml=”http://www.opengis.net/gml”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
```

```
xsi:schemaLocation="http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd"
service="WFS"
version="1.1.0"
outputFormat="text/xml; subtype=gml/3.1.1"
traverseXlinkDepth="1"
traverseXlinkExpiry="1">
<ogc:GmlObjectId gml:id="c1"/>
</wfs:GetGmlObject>
```

La respuesta que se obtiene es:

```
<Ciudad gml:id="c1">
<gml:id="c1">
<gml:name>CiudadGotica</gml:name>
<gml:directedNode orientation="+" xlink:href="#n1"/>
</Ciudad>
```

Se puede observar en la respuesta que el elemento <Ciudad> posee un elemento anidado <gml:directedNode> que no está resuelto, es decir, no está completo sino que posee un XLink a "#n1". Para poder resolver todos los elementos en forma recursiva, haríamos la misma solicitud pero con el valor traverseXlinkDepth="*".

```
<Ciudad gml:id="c1">
<gml:name>Bedford</gml:name>
<gml:directedNode orientation="+">
<!-- xlink:href="#n1" -->
<gml:Node gml:id="n1">
<gml:pointProperty>
<!-- xlink:href="http://www.ciudadgotica.gov/gps.gml#townHall" -->
<gml:Point gml:id="townHall" srsName="…">
<gml:pos>147 234</gml:pos>
</gml:Point>
</gml:pointProperty>
</gml:Node>
</gml:directedNode>
</Ciudad>
```

En la nueva respuesta, dos XLinks fueron resueltos, uno de profundidad 2 y el otro de profundidad 3 respecto a la solicitud originaria. Para el primer elemento (<gml:directedNode>), se realizó una solicitud GetGmlObject al WFS local con el gml:id="n1". Para el segundo elemento (<gml:Point>), se envió una solicitud GetGmlObject al WFS remoto alojado en la URL http://www.ciudadgotica.gov/gps.gml con el gml:id="townHall".

3.1.4 Transaction

3.1.4.1 Solicitud

El siguiente fragmento de esquema XML define las solicitudes de este tipo:

```
<xsd:element name="Transaction" type="wfs:TransactionType"/>
<xsd:complexType name="TransactionType">
<xsd:complexContent>
<xsd:extension base="ows:GetCapabilitiesType">
<xsd:sequence>
<xsd:element ref="wfs:LockId" minOccurs="0"/>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element ref="wfs:Insert"/>
<xsd:element ref="wfs:Update"/>
<xsd:element ref="wfs:Delete"/>
<xsd:element ref="wfs:Native"/>
</xsd:choice>
</xsd:sequence>
<xsd:attribute name="releaseAction"
type="wfs:AllSomeType" use="optional"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="LockId" type="xsd:string"/>
<xsd:element name="Insert" type="wfs:InsertElementType"/>
<xsd:complexType name="InsertElementType">
<xsd:choice>
<xsd:element ref="gml:_FeatureCollection" />
<xsd:sequence>
<xsd:element ref="gml:_Feature" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:choice>
<xsd:attribute name="idgen"
type="wfs:IdentifierGenerationOptionType"
use="optional" default="GenerateNew"/>
<xsd:attribute name="handle" type="xsd:string" use="optional"/>
<xsd:attribute name="inputFormat" type="xsd:string"
use="optional" default="text/xml; subversion=gml/3.1.1"/>
<xsd:attribute name="srsName" type="xsd:anyURI" use="optional"/>
</xsd:complexType>
<xsd:simpleType name="IdentifierGenerationOptionType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="UseExisting"/>
<xsd:enumeration value="ReplaceDuplicate"/>
```

```
<xsd:enumeration value="GenerateNew"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:element name="Update" type="wfs:UpdateElementType"/>
<xsd:complexType name="UpdateElementType">
<xsd:sequence>
<xsd:element ref="wfs:Property" maxOccurs="unbounded"/>
<xsd:element ref="ogc:Filter" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="handle" type="xsd:string" use="optional"/>
<xsd:attribute name="typeName" type="xsd:QName" use="required"/>
<xsd:attribute name="inputFormat" type="xsd:string"
use="optional" default="text/xml; subversion=gml/3.1.1"/>
<xsd:attribute name="srsName" type="xsd:anyURI" use="optional"/>
</xsd:complexType>
<xsd:element name="Property" type="wfs:PropertyType"/>
<xsd:complexType name="PropertyType">
<xsd:sequence>
<xsd:element name="Name" type="xsd:QName"/>
<xsd:element name="Value" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<xsd:element name="Delete" type="wfs:DeleteElementType"/>
<xsd:complexType name="DeleteElementType">
<xsd:sequence>
<xsd:element ref="ogc:Filter" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="handle" type="xsd:string" use="optional"/>
<xsd:attribute name="typeName" type="xsd:QName" use="required"/>
</xsd:complexType>
<xsd:element name="Native" type="wfs:NativeType"/>
<xsd:complexType name="NativeType">
<xsd:attribute name="vendorId" type="xsd:string" use="required"/>
<xsd:attribute name="safeToIgnore" type="xsd:boolean" use="required"/>
</xsd:complexType>
```

3.1.4.1.1 Transaction

El elemento <Transaction> contiene una secuencia de elementos <Insert>, <Update> y <Delete>, que define las operaciones que se ejecutarán sobre los datos y el orden correspondiente. Nótese que las operaciones de Update y Delete pueden ejecutarse sobre características creadas mediante Insert de la misma transacción.

Al finalizar la transacción, el WFS aplicará el procedimiento que corresponda al sistema en el que se almacenan los datos. Por ejemplo, en un RDBMS, se aplicará un commit para finalizar la transacción o un rollback para abortar los cambios si ocurre algún error.

3.1.4.1.2 Insert

Las características que son insertadas por esta operación vienen especificadas como una secuencia de elementos <gml:_Feature>, es decir, elementos de características en formato GML.

El atributo idgen define el método para asociar identificadores a las nuevas características. Los valores que puede tomar son:

- GenerateNew: El WFS generará identificadores únicos para las nuevas características.
- UseExisting: El WFS utilizará los identificadores provistos en la solicitud por los atributos gml:id de cada característica. Si se encuentra algún duplicado, se generará una excepción.
- ReplaceDuplicate: El WFS utilizará los identificadores provistos en la solicitud por los atributos gml:id de cada característica. Si se encuentra algún duplicado, generará un identificador único para la nueva característica.

En el siguiente ejemplo se utiliza la operación <Insert> para crear dos nuevas características en una capa de departamentos.

```
<?xml version="1.0"?>
<wfs:Transaction
version="1.1.0"
service="WFS"
xmlns="http://www.someserver.com/myns"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.someserver.com/myns
http://www.someserver.com/wfs/cwwfs.cgi?
request=describefeaturetype&amp;typename=InWaterA_1M.xsd
http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd">
<wfs:Insert idgen="UseExisting">
<Departamento gml:id="Depto1">
<wkbGeom>
<gml:Polygon gml:id="P1"
srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
<gml:exterior>
<gml:LinearRing>
<gml:posList>-98.54 24.26 ...</gml:posList>
</gml:LinearRing>
</gml:exterior>
</gml:Polygon>
</wkbGeom>
<id>150</id>
<f_code>ABCDE</f_code>
<hyc>152</hyc>
<tileId>250</tileId>
```

```
<facId>111</facId>
</ Departamento >
< Departamento gml:id=”Depto2”>
<wkbGeom>
<gml:Polygon gml:id=”P2”
srsName=”http://www.opengis.net/gml/srs/epsg.xml#63266405”>
<gml:exterior>
<gml:LinearRing>
<gml:posList>-99.99 22.22 ...</gml:posList>
</gml:LinearRing>
</gml:exterior>
</gml:Polygon>
</wkbGeom>
<id>111</id>
<f_code>FGHIJ</f_code>
<hyc>222</hyc>
<tileId>333</tileId>
<facId>444</facId>
</Departamento>
</wfs:Insert>
```

3.1.4.1.3 Update

Un elemento <Update> contiene uno o más elementos <Property> que especifican el nombre de una propiedad (<Name>) y el valor de reemplazo para la misma (<Value>). El tipo de característica al que pertenece la propiedad se declara mediante el atributo typeName.

Consejo de Educación Técnico Profesional Servicios Geográficos

El alcance de la operación, es decir, las instancias dentro del tipo que serán afectadas por la actualización se restringe mediante el elemento <Filer>.

En el siguiente ejemplo se actualiza una propiedad, la población, de una instancia del tipo de característica Ciudad. Dentro del filtro se utiliza un elemento <GmlObjectId> para indicar el gml:id que identifica la ciudad que se quiere actualizar.

```
<?xml version=”1.0” ?>
<wfs:Transaction
version=”1.1.0”
service=”WFS”
xmlns=”http://www.someserver.com/myns”
xmlns:ogc=”http://www.opengis.net/ogc”
xmlns:wfs=”http://www.opengis.net/wfs”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:schemaLocation=”http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd”>
<wfs:Update typeName=”Ciudad”>
<wfs:Property>
<wfs:Name>poblacion</wfs:Name>
<wfs:Value>4070000</wfs:Value>
</wfs:Property>
<ogc:Filter>
<ogc:GmlObjectId gml:id=”CiudadGotica.10131”/>
</ogc:Filter>
</wfs:Update>
</wfs:Transaction
```

3.1.4.1.4 Delete

El elemento <Delete> permite indicar que una o más instancias de una característica deben se borradas. Para esto se utiliza el elemento <Filter> de la misma forma que en la operación <Update>.

3.1.4.1.4.1 Ejemplo

En el siguiente ejemplo se muestra como se puede utilizar esta operación para eliminar las características del tipo Ciudad que estén dentro de un polígono (elementos <ogc:Within> y <gml:Polygon>).

```
<?xml version=”1.0” ?>
<wfs:Transaction
version=”1.1.0”
service=”WFS”
xmlns=”http://www.someserver.com/myns”
xmlns:wfs=”http://www.opengis.net/wfs”
xmlns:gml=”http://www.opengis.net/gml”
xmlns:ogc=”http://www.opengis.net/ogc”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:schemaLocation=”http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd”>
<wfs:Delete typeName=”Ciudad”>
<ogc:Filter>
<ogc:Within>
<ogc:PropertyName>wkbGeom</ogc:PropertyName>
<gml:Polygon gid=”pp9”
srsName=”http://www.opengis.net/gml/srs/epsg.xml#63266405”>
<gml:exterior>
<gml:LinearRing>
<gml:posList>-95.7 38.1 -97.8 38.2 ...</gml:posList>
</gml:LinearRing>
</gml:exterior>
</gml:Polygon>
</ogc:Within>
</ogc:Filter>
</wfs:Delete>
</wfs:Transaction>
```

3.1.4.2 Respuesta

El siguiente fragmento de esquema XML define el formato de la respuesta a la operación Transaction.

```
<xsd:element name=”TransactionResponse”
type=”wfs:TransactionResponseType”/>
<xsd:complexType name=”TransactionResponseType”>
<xsd:sequence>
<xsd:element name=”TransactionSummary”
type=”wfs:TransactionSummaryType”/>
<xsd:element name=”TransactionResults”
type=”wfs:TransactionResultsType” minOccurs=”0”/>
<xsd:element name=”InsertResults”
type=”wfs:InsertResultsType” minOccurs=”0”/>
</xsd:sequence>
<xsd:attribute name=”version”
type=”xsd:string” use=”required” fixed=”1.1.0”/>
</xsd:complexType>
<xsd:complexType name=”TransactionSummaryType”>
<xsd:sequence>
<xsd:element name=”totalInserted”
type=”xsd:nonNegativeInteger”
minOccurs=”0”/>
<xsd:element name=”totalUpdated”
type=”xsd:nonNegativeInteger”
minOccurs=”0”/>
<xsd:element name=”totalDeleted”
type=”xsd:nonNegativeInteger”
minOccurs=”0”/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name=”TransactionResultsType”>
<xsd:sequence>
<xsd:element name=”Action” type=”wfs:ActionType”
minOccurs=”0” maxOccurs=”unbounded”/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name=”ActionType”>
<xsd:sequence>
<xsd:element name=”Message” type=”xsd:string”
minOccurs=”0” maxOccurs=”1”/>
</xsd:sequence>
<xsd:attribute name=”locator” type=”xsd:string” use=”required”/>
<xsd:attribute name=”code” type=”xsd:string” use=”optional”/>
</xsd:complexType>
<xsd:complexType name=”InsertResultsType”>
<xsd:sequence>
<xsd:element name=”Feature”
type=”wfs:InsertedFeatureType”
maxOccurs=”unbounded”/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name=”InsertedFeatureType”>
<xsd:sequence>
<xsd:element ref=”ogc:FeatureId” maxOccurs=”unbounded”/>
</xsd:sequence>
<xsd:attribute name=”handle” type=”xsd:string” use=”optional”/>
</xsd:complexType>
```

Dentro de los elementos de la respuesta, el único obligatorio es el <TransactionSummary>, que indica el número de características creadas, modificadas y eleminadas en la transacción.

3.1.5 GetCapabilities

Como todo OWS, WFS posee una operación GetCapabilities que le permite devolver la metadata del servicio para poder comunicarle al cliente cuáles son sus capacidades específicas.

A continuación se muestra el formato XML de la solicitud y la respuesta de esta operación en el caso que se utilice el método HTTP POST. Si se utiliza HTTP GET, la solicitud debe codificarse como KVP enviando la lista de parámetros de la misma manera que se describió para el protocolo WMS (ver 2.2.1).

3.1.5.1 Solicitud

El siguiente fragmento de esquema XML define las solicitudes de este tipo.

```
<xsd:element name=”GetCapabilities” type=”wfs:GetCapabilitiesType”/>
<xsd:complexType name=”GetCapabilitiesType”>
<xsd:complexContent>
<xsd:extension base=”ows:GetCapabilitiesType”>
<xsd:attribute name=”service” type=”ows:ServiceType”
use=”optional” default=”WFS”/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

3.1.5.2 Respuesta

El siguiente fragmento de esquema XML define parcialmente las respuestas de este tipo.

```
<xsd:element name=”WFS_Capabilites”
type=”wfs:WFS_CapabilitiesType”
substitutionGroup=”ows:Capabilites”/>
<xsd:complexType name=”WFS_CapabilitiesType”>
<xsd:complexContent>
<xsd:extension base=”ows:CapabilitiesBaseType”>
<xsd:sequence>
<xsd:element ref=”wfs:FeatureTypeList” minOccurs=”0”/>
<xsd:element ref=”wfs:ServesGMLObjectTypeList” minOccurs=”0”/>
<xsd:element ref=”wfs:SupportsGMLObjectTypeList”/>
<xsd:element ref=”ows:Filter_Capabilities”/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

El contenido de la respuesta de esta operación se divide en las siguientes secciones:

- Service Identification: Provee información que identifica al servicio como un WFS.
- Service Provider: Provee información que identifica a la organización que opera el WFS.
- Operation Metadata: Provee metadata sobre cada una de las operaciones que provee este WFS, incluyendo los parámetros y restricciones de cada una.
- Lista de FeatureType: Declara la lista de tipos de características que están disponibles en el WFS. Para cada tipo, se provee información adicional, como el SRS por defecto, los otros SRS soportados, etc.
- Lista de ServesGMLObjectType: Declara la lista de tipos de objetos GML que no son características (no son derivados de gml:AbstractFeatureType) que están disponibles en el WFS y pueden obtenerse mediante la operación GetGmlObject.
- Lista de SupportsGMLObjectType: Declara la lista de tipos de objetos GML que el WFS podría servir si estuviera configurado para servir datos descriptos por un esquema de aplicación que usara esos tipos directamente (tipos no-abstractos) o definiera otros tipos derivados de ellos.
- Filter: Define los tipos de filtros que el WFS soporta para restringir el alcance de las operaciones. Si no presente, solo se soporta una conjunto mínimo de filtros básicos.

4 Referencias

[1] Open Geospatial Consortium (OGC) (Junio 2011)
http://www.opengeospatial.org/
[2] Hypertext Transer Protocol (HTTP) (Junio 2011)
http://www.ietf.org/rfc/rfc2616.txt
[3] Extensible Markup Language (XML) (Junio 2011)
http://www.w3.org/XML/
[4] Multipurpose Internet Mail Extensions (MIME) (Junio 2011)
http://tools.ietf.org/html/rfc2045
[5] KML (Junio 2011)
http://www.opengeospatial.org/standards/kml/
[6] Web Feature Service (WFS) (Junio 2011)
http://www.opengeospatial.org/standards/wfs
[7] Geographic Markup Language (GML) (Junio 2011)
http://www.opengeospatial.org/standards/gml
[8] Web Map Service (WMS) (Junio 2011)
http://www.opengeospatial.org/standards/wms

Visualización de mapas

1. Introducción

El resultado final de muchas operaciones geográficas es un mapa, los mismos son datos que almacenan y comunican información geográfica. Dichos mapas representan datos espaciales y son los que contienen las ubicaciones y formas de características cartográficas. La cartografía es la ciencia que se encarga del estudio y de la elaboración de los mapas geográficos y territoriales.

Estos datos son también llamados datos cartográficos digitales y son necesarios para realizar mapas y para estudiar relaciones espaciales.

Los datos espaciales incluyen puntos que representan bancos, hospitales, escuelas, etc., y líneas que representan calles, ríos, rutas, etc.

Dado que los mapas son un conjunto de datos, los mismos pueden ser visualizados y editados y, para esta tarea, son necesarios los visualizadores de mapas. Esta herramienta es un software que se encarga de la interacción con el usuario ofreciendo una interfaz fácil de usar.

2. Visualizadores

En esta sección se presentan ejemplos de visualizadores de escritorio y visualizadores Web. Dentro de estos últimos se diferencian los visualizadores que son sitios web en sí mismos (como Yahoo, Google Maps y Bing Maps) y, por otro lado, aquellos que son sitios desarrollados a medida, que son aplicaciones que utilizan un simple Browser o navegador para acceder a servicios geomáticos desde cualquier ubicación con conexión a Internet.

Además, se comentan las principales funcionalidades de cada uno.

2.1 Visualizadores de escritorio

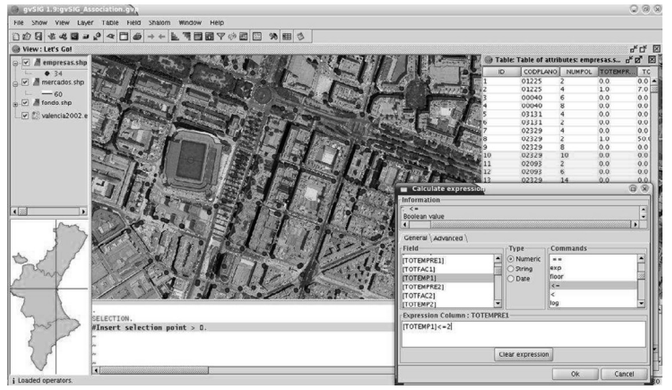
gvSIG[1]

gvSIG Desktop permite visualizar y editar información geográfica. Es capaz de acceder al formato vectorial y a rasters, tanto locales como remotos, integra estándares Open Geospatial Consortium [2] (OGC) y cuenta con un amplio número de herramientas para trabajar con información geográfica (consulta, creación de mapas, geoprocesamiento, redes, etc.).

La OGS define estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica y de la World Wide Web.

De las funcionalidades de gvSIG se destacan las que se presentan a continuación:

• Vectorial: Acceso a formatos vectoriales, acceso a bases de datos, navegación, consulta, selección, análisis y geoprocesamiento; edición gráfica y alfanumérica, simbología, etiquetado, diseñador de planos, conversión de datos a otros formatos y sistemas de proyección, relaciones entre tablas, estadísticas, normalización, etc.. En la figura 1 se muestra un ejemplo de dicha sesión de trabajo.



Raster: acceso a formatos raster, tabla de color y gradientes, recorte de datos, exportación de capas, procesamiento por píxel, histogramas, geolocalización, reproyección de raster, georreferenciación, vectorización automática, definición de áreas de interés, fusión de imágenes, etc. En la figura 2 se presenta una imagen con estas funcionalidades.

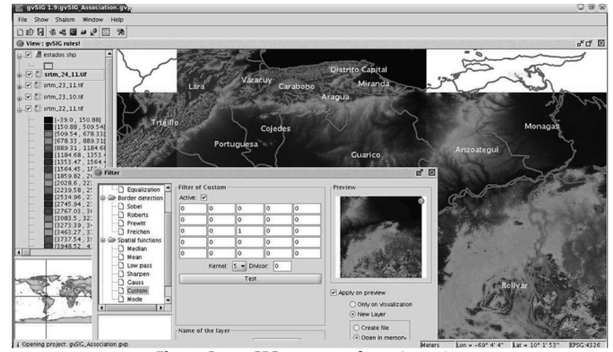


Figura 2 – gvSIG, acceso a formato raster

3D: permite tener una vista 3D plana y 3D esférica. Tiene capas de elevación, capas vectoriales con alturas, capas 3D, posibilidad de rasterizar o visualizar como primitivas gráficas las capas vectoriales; simbología 3D, georreferenciación y edición de objetos 3D, selección, información, búsqueda geográfica por nombre (gazeeteer), etc. La imagen de la figura 3 presenta un ejemplo.

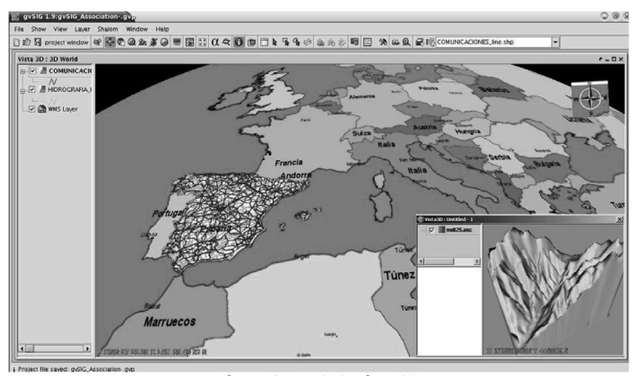


Figura 3 – gvSIG, vista 3D

QuantumGIS[3]

QuantumGIS o QGIS ofrece muchas funcionalidades SIG comunes. Se pueden ver y superponer datos vectoriales; da la posibilidad de componer mapas y de explorar datos espaciales. A su vez, es posible crear, editar, gestionar y exportar mapas vectoriales a varios formatos. Los rasters pueden ser importados a GRASS para poder ser editados y exportados a otros formatos. En la figura 4 se muestra la ventana principal de QSIG con datos de ejemplo y algunas de las principales funcionalidades de QGIS se listan a continuación:

- Soporte ráster y vectorial.
- Soporte para PostgreSQL con tablas espaciales utilizando PostGIS.
- Integración con GRASS, incluída visualización, edición y análisis.
- Digitalización GRASS y OGR/Shapefile.
- Diseño de Mapas.
- Soporte OGC.
- Edición / Visualización / Busqueda de atributos.
- Cambio de simbología vectorial y raster, etc.

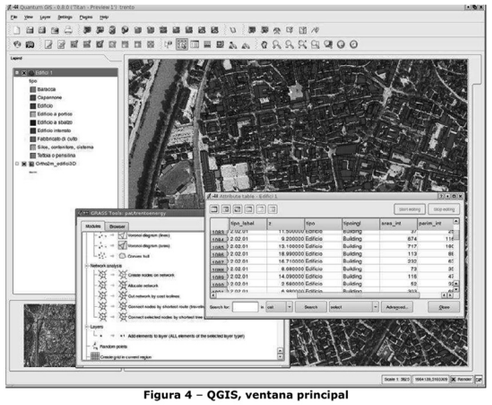


Figura 4 – QGIS, ventana principal

ArcGIS Desktop[4]

Es un conjunto de productos software que corre en computadoras de escritorio estándar.

Se utiliza para crear, importar, editar, consultar, hacer mapas, analizar y publicar información geográfica. Hay cuatro productos en la colección de ArcGIS Desktop y cada uno trae un nivel de funcionalidad creciente. Los mismos se describen a continuación:

• ArcReader: es un visor gratuito para mapas que usan los otros productos de ArcGIS Desktop. Este puede visualizar e imprimir todos los mapas y tipos de datos. También tiene algunas herramientas simples para explorar y consultar mapas. En la figura 5 se presenta una imagen de dicha herramienta.

• ArcView: En la figura 6 se muestra la imagen de la herramienta ArcView, la misma se centra en el uso adecuado de los datos, mapeos y análisis.

• ArcEditor: Agrega la funcionalidad de edición geográfica y creación de datos. En la figura 7 se presenta este software

• ArcInfo: En la figura 8 se muestra la herramienta de nivel superior y aquí se incluye geoprocesamiento avanzado.

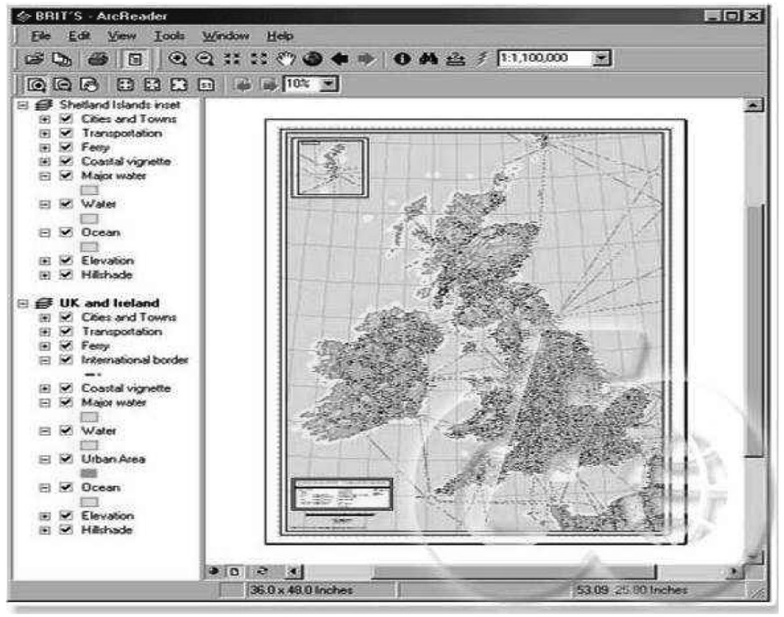


Figura 5 - ArcReader

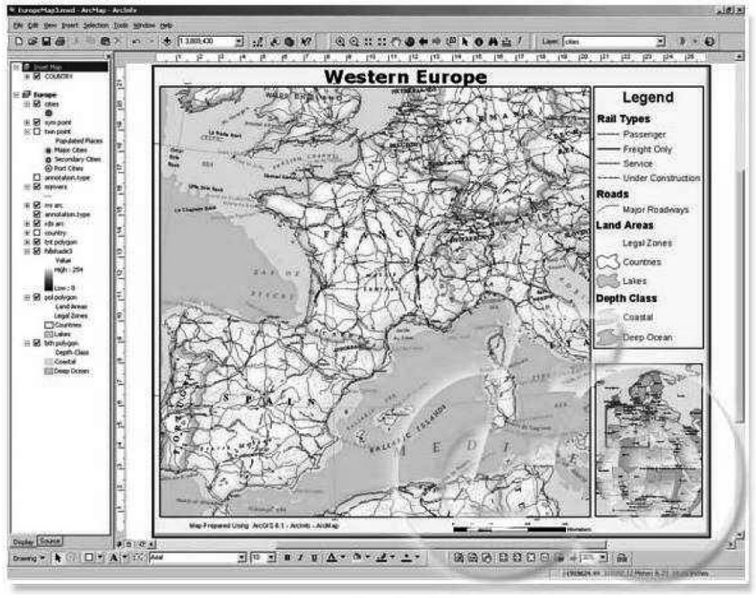


Figura 6 – ArcView

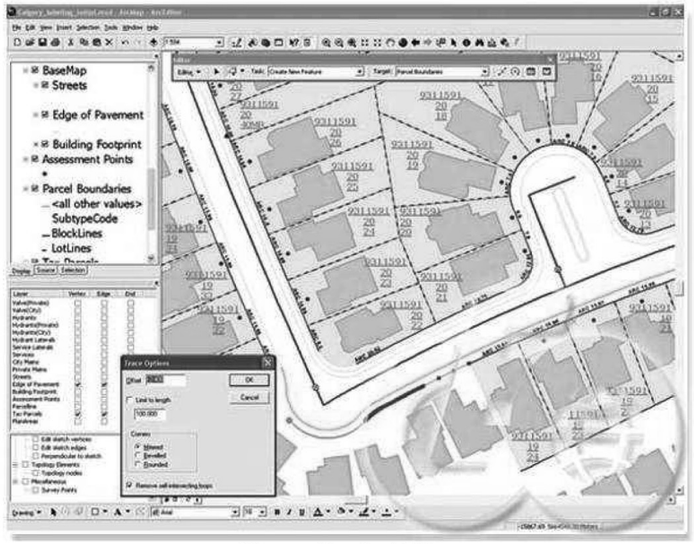


Figura 7 – ArcEditor

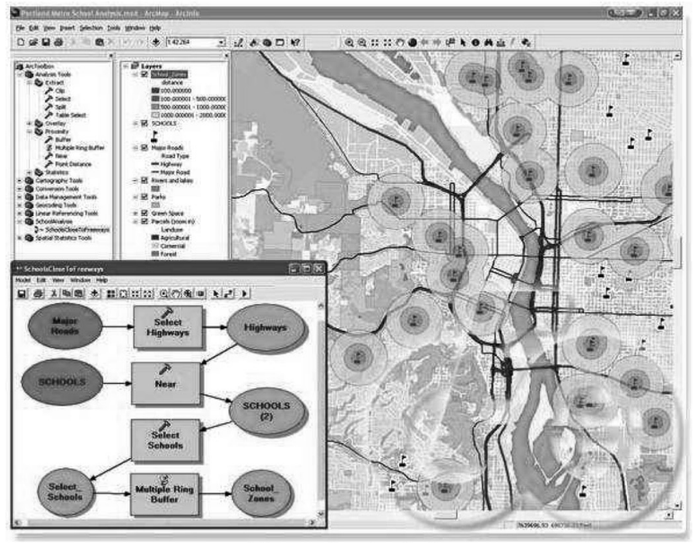


Figura 8 – ArcInfo

Emerillon[5]

Es un visualizador de mapas de escritorio para Gnome. El proyecto Gnome es una comunidad que hace software libre y es el entorno de escritorio más populares para GNU/ Linux y sistemas operativos de tipo UNIX.

Búsqueda por localidad, manejo de zoom, lista de favoritos, vista del transporte público, vista del terreno, de mapas y de rutas son las principales funcionalidades de Emerillon, en la figura 9 se presenta una imagen del visualizador.

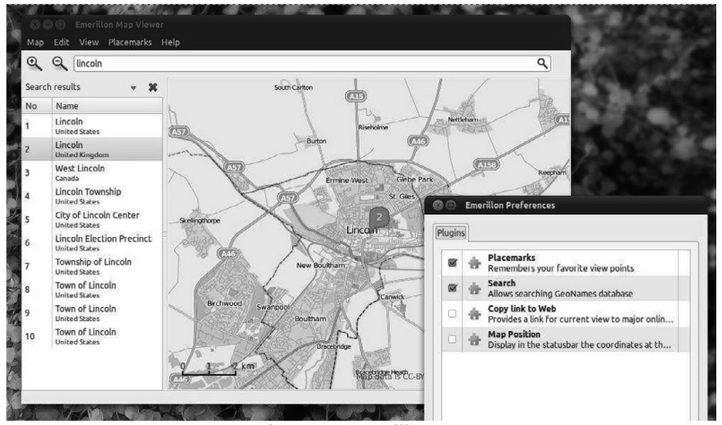


Figura 9 - Emerillon

Google Earth[6]

Google Earth es una aplicación de escritorio que combina una base cartográfica de imágenes aéreas y de satélite de alta resolución de las zonas más pobladas del mundo, con un buscador de puntos de interés (POI) y direcciones, permitiendo vistas en 3D mediante la proyección de las capas de imágenes sobre un modelo digital del terreno.

Google ofrecen una versión gratuita y dos versiones de pago. La versión gratuita de Google Earth tiene, entre sus funcionalidades, la posibilidad de ir desde y hacia cualquier dirección o lugar, de realizar búsquedas de interés como son escuelas, parques, restaurantes, hoteles, hospitales, etc. También ofrece las direcciones del tráfico y rutas.

Por otro lado, la vista puede ser inclinada y rotada para ver en 3D el terreno y los edificios.

Las búsquedas y favoritos pueden ser guardadas y compartidas y, a su vez, se pueden añadir anotaciones geo-referenciadas. En la figura 10 se observa una vista del Cañón de Colorado en las pantallas de Google Earth.

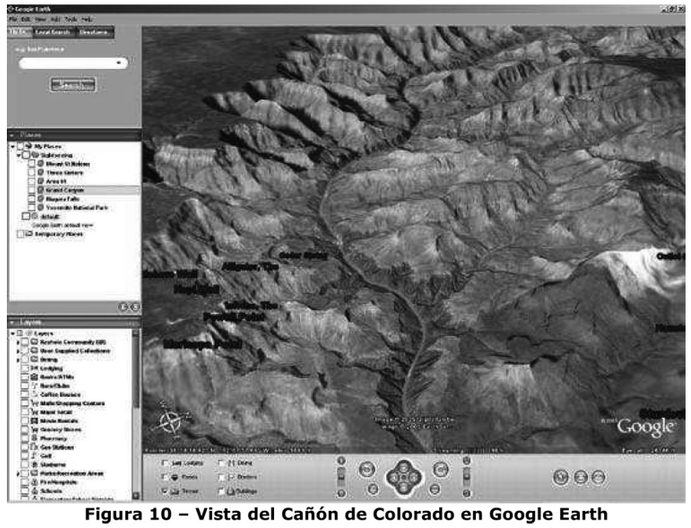


Figura 10 – Vista del Cañón de Colorado en Google Earth

2.2 Visualizadores Web

Google Maps[7]

Google Earth permite utilizar al mismo tiempo la aplicación Google Maps que es su versión Web más sencilla. En la figura 11 se muestra una imagen del sitio Web de google Maps. La base de información cartográfica e imágenes que utiliza Google Earth es prácticamente la misma que se puede visualizar en Google Maps, aunque se observa que las imágenes están proyectadass de forma distinta, es decir, con distinta proyección cartográfica.

Google Maps acepta únicamente longitud y latitud, no posee geocoder. Un geocoder es una herramienta que proporciona la ubicación exacta o aproximada de algún dato geográfico. Además, en una ventana de información es posible agregar texto HTML o XML.

Figura 11 – Sitio Web de google map

Yahoo[8]

Las funcionalidades de Yahoo Maps son: integración de elementos al realizar búsquedas, página de impresión, selección de tipos de rutas y elementos destacados de los mapas; permite agregar texto HTML en una pequeña ventana de información y proporciona su propio geocoder. Sus características son muy parecidas a Google Maps y Bing Maps.

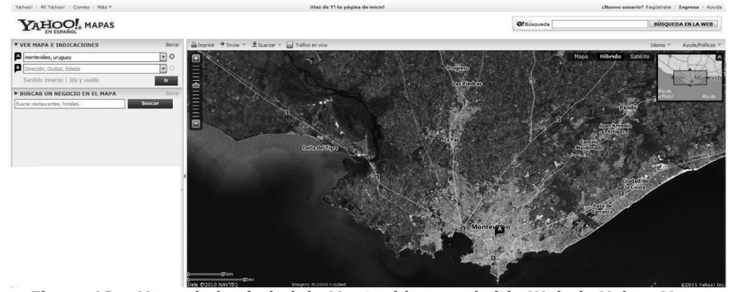


Figura 12 – Mapa de la ciudad de Montevideo en el sitio Web de Yahoo Maps

Bing Microsoft[9]

Bing Maps es un servicio de mapas Web, parte del motor de búsqueda Bing de Microsoft.

Es un sitio Web que ofrece un mapa de la Tierra navegable en tres dimensiones, incluyendo monumentos, edificios y parajes naturales. Entre sus funcionalidades presenta visión alterna 2D y 3D. Además, ofrece un modo de viaje a vista de pájaro y está integrado con el buscador Bing.

Por otro lado, ofrece el servicio Street side, que presenta una vista completa y continua de las calles, pero sólo está disponible para las grandes ciudades de Estados Unidos. Por lo que, una de las desventajas de Bing Maps es que carece de información completa de muchos países. En la figura 13 se muestra una imagen del mapa de la ciudad de Montevideo.

Figura 13 – Localización de la ciudad de Montevideo en Bing Maps

Las herramientas que se presentan a continuación son productos personalizables, son desarrollos a media.

OpenstreetMap[10]

OpenStreetMap (OSM) es un proyecto colaborativo para crear mapas libres y editables.

Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles, ortofotografías y otras fuentes libres. Esta cartografía, tanto las imágenes creadas como los datos vectoriales, son almacenados en su base de datos.

Los datos en bruto que los colaboradores capturan con sus dispositivos GPS sirven como guía para dibujar las nuevas vías. Estos datos suelen cargarse desde el equipo local del usuario o bien solicitando al servidor de OSM que descargue aquellas trazas de la zona que van a ser editadas y que otros usuarios han subido previamente a OpenStreetMap. Los datos brutos son de libre acceso para el desarrollo de otras aplicaciones. El usuario debe registrarse de

manera gratuita mediante una dirección de correo para poder realizar ediciones pero, si solo se desea consultar información, dicho registro no es necesario. A continuación se muestra el mapa de Uruguay en una vista de OSM.

Figura 14 – Vista del mapa de Uruguay con OpenStreetMaps

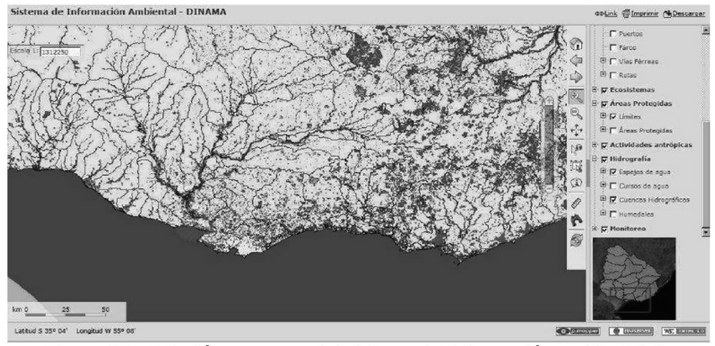


Figura 15 – Interfáz p.mapper del sistema de Información ambiental de DINAMA

i3Geo[13]

i3Geo es un software para internet basado en un conjunto de software libres, principalmente Mapserver [12]. Ofrece datos geográficos que pueden ser consultados utilizando herramientas de navegación, generación de análisis, etc. I3Geo incorpora funcionalidades que facilitan el acesso remoto a los datos, permitiendo el establecimento de redes cooperativas. En la figura 16 se adjunta una imagen del sitio i3Geo.

Se busca difundir el uso del geoprocesamiento como instrumento técnico-científico e implementar una interfáz genérica para acceder a los datos geográficos existentes en instituciones públicas, privadas o no gubernamentales, por esto puede ser utilizado e incorporado por cualquier institución interesada, sin costos.

Figura 16 – Vista del sitio i3Geo

3. OpenLayers

OpenLayers [14] es un cliente visualizador de mapas ligero basado en JavaScript. Ofrece un API para acceder a diferentes fuentes de información cartográfica en la red:

- Protocolo Web Map Services (WMS, permite acceder a diferentes servidores de cartografía digital utilizando un lenguaje común).
- Mapas comerciales (tipo Google Maps, Bing Maps, Yahoo Maps).
- Protocolo Web Features Services (WFS, ofrece una interfaz de comunicación que permite interactuar con los mapas servidos por el estándar WMS como, por ejemplo, editar la imagen que nos ofrece el servicio WMS o analizar la imagen siguiendo criterios geográficos).
- Distintos formatos vectoriales
- Mapas de OpenStreetMap, etc.

Una interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

OpenLayers hace que sea fácil colocar un mapa dinámico en cualquier página Web. Se puede mostrar cuadros de mapas de cualquier fuente y se ha desarrollado para promover el uso de la información geográfica de todo tipo.

Tiene por objeto separar las herramientas de mapas de los datos de mapas, de forma que todas las herramientas pueden funcionar en todas las fuentes de datos.

Ejemplos de uso de OpenLayers pueden ser encontrados en [15], algunos como los que se presentan a continuación pueden ser destacados:

- demostración de uso de capas Bing,
- uso de OpenLayers usando un servidor ArcGIS,
- demostración de la versión 3 del API de Google Maps, etc.

Como se observa, varios de los visualizadores antes presentados utilizan OpenLayers. En las figuras que se ofrecen a continuación se presenta cómo es posible personalizar un mapa utilizando OpenLayers.

En la figura 17 se muestra cómo editar y crear puntos, líneas y polígonos. En este caso se dibujó un punto en la ciudad de Montevideo.

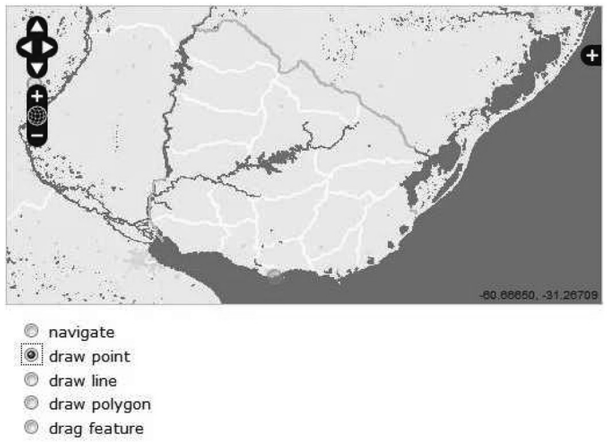


Figura 17 – Edición y creación de puntos, líneas y polígonos con OpenLayers.

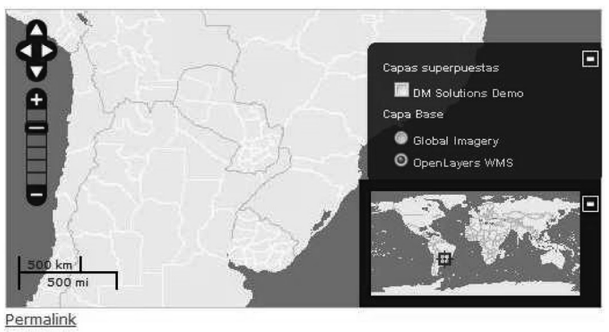


Figura 18 – Ejemplos de controles de mapas con OpenLayers.

En la figura 18 se observa, en un mapa de referencia, dónde se hizo el zoom sobre el mapa de trabajo. Además, se puede ver que en las opciones de capa base está seleccionado el protocolo de WMS y es posible la superposición de capas.

Las clases básicas de OpenLayers son las que se describen a continuación:

• Map.js: Es el objeto central de OpenLayers y contiene a todos los demás.

• Layer.js: Cada capa hereda los métodos básicos de esta capa.

• Control.js: Controles del Mapa. Tienen una relación "1 a 1" con los handlers y generalmente son los elementos que suelen personalizarse.

• Handler.js: Son los manejadores de eventos. Están asociados a los eventos típicos de la web como es, por ejemplo, el "Click".

4 Bibliografía

http://www.gvsig.com/
http://www.opengeospatial.org/
http://www.qgis.org/
http://www.geoinfo-int.com/htmls/prod_arcgis_desk.html.
http://www.webcoz.com/install-emerillon-desktop-map-viewer-for-gnome/
http://google-earth.uptodown.com/
http://maps.google.com/
http://espanol.maps.yahoo.com/
http://www.bing.com/maps/#
http://www.openstreetmap.org/
http://www.pmapper.net/
http://mapserver.gis.umn.edu/
http://mapas.mma.gov.br/i3geo/aplicmap/openlayers.htm?ba0ac7ff2d4740bd21e68eb2592d509e
http://www.openlayers.org/
http://openlayers.org/dev/examples/

Metadatos y calidad de la información geográfica

1. Introducción

La primer pregunta que surge es qué son los metadatos. Los metadatos son información acerca de los datos [1] [2] [3] o, como suele decirse, son los datos de los datos; en definitiva, diferentes formas que se refieren a la descripción de los datos.

Los metadatos son creados para comunicar, por lo tanto deben compartirse y es conveniente que se haga a través de un estándar. El más aceptado en cuanto a información geoespacial es el del Comité Federal de Datos Geográficos de los Estados Unidos (Federal Geographic Data Committee o su sigla FGDC) [4]. También está muy avanzado el tratamiento del estándar de International Standard Organization (ISO TC/211) [5]. La importancia de los estándares radica en que éstos se han definido para determinar qué información debe documentarse de las bases de datos; proveen una terminología común y un conjunto de definiciones para la documentación de los datos geoespaciales. Un ejemplo de estándar de metadatos de información geográfica se encuentra disponible en [2].

Una de las principales ventajas de los metadatos es la organización y mantenimiento de un catálogo de datos de una organización o sistema y su interacción entre estas entidades, ya que podrían manejar independientemente diferentes estructuras pero comunicarse entendiendo el mismo estándar de metadatos. Los metadatos describen el contenido, la calidad, la condición y otras características de los datos. En el caso de los Sistemas Geográficos es importante mencionar que el contenido de los estándares para metadatos geo-espaciales del Comité Federal de Datos Geográficos (FGDC) de los Estados Unidos [4] fue diseñado para documentar un conjunto de datos geo-espaciales.

Los estándares para metadatos documentan las características o propiedades de los datos.

Los principales usos de los metadatos son:

• Ayudar a las empresas a organizar y dar valor agregado a su inversión en datos geo-referenciados.

• Proveer información sobre las bases de datos de las que dispone las empresas, de forma tal que se puedan formar catálogos de datos, repositorios de datos y proveer información ágil a potenciales comercializadores de datos.

• Proveer información que permita procesar los archivos de una fuente externa al usuario.

• Proveer una guía para los usuarios de los datos en cuanto a su resolución espacial, sistema de coordenadas, datum y calidad.

2. Diseño y organización de los Metadatos

Los metadatos, según FGDC [4] están constituidos por datos o elementos agrupados en siete secciones principales y tres de apoyo, algunas de ellas son de carácter obligatorio. En la figura 1 se muestra dicha distribución.

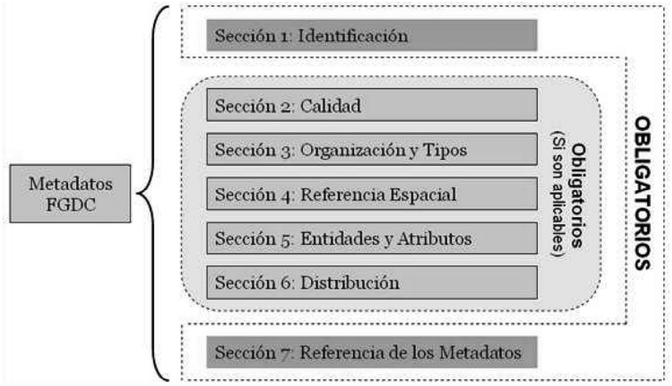


Figura 1.- Organización del estándar de Metadatos de la FGDC

Cada una de las secciones del estándar dispone de elementos obligatorios, otros elementos obligatorios para el caso en el que sean aplicables, y elementos opcionales.

Cuando no se cuenta con un dato que es obligatorio se debe incluir un contenido que aclare su inexistencia. Los elementos obligatorios, cuando son aplicables, se deben especificar si los datos cuentan con las características a describir. Los elementos opcionales se incorporan si el proveedor de la información lo desea. Para cada una de las secciones que se observa en la imagen anterior se presenta una breve descripción:

• La sección 1 se refiere a la "Identificación" de la información. Trata información acerca de la identificación de los metadatos, como puede ser su propósito y descripción, los tiempos de publicación y de su actualización, la frecuencia de mantenimiento y el estado de avance. También los datos de posicionamiento espacial, las personas de contactos y las palabras claves, es decir los términos que describen y permiten ubicar representativamente al metadato.

• La sección 2 corresponde a la "Calidad". Básicamente almacena información sobre la precisión y la consistencia lógica de los datos, también los métodos y tiempos de captura o creación.

• La sección 3 se refiere a la "Organización y Tipos". Considera la información sobre la referencia espacial y los objetos vectoriales o rasters que la conforman.

• La sección 4 corresponde a la "Referencia Espacial". Se refiere al Datum y a la definición del sistema de coordenadas, ya sean geográficas, planas o locales.

• La sección 5 toma en cuenta las "Entidades y Atributos". Detalla en forma repetitiva la definición de cada campo de las estructuras de datos asociadas.

• La sección 6 corresponde la "Distribución" que describe el medio y modo en que se presenta y distribuye la información, ello incluye formato, disponibilidad, ubicación, accesibilidad y precio, entre otras características.

• La sección 7 corresponde a la "Referencia de los metadatos". Esta incluye una breve descripción del metadato en sí, y no de la información que describe el metadato, por este motivo, junto a la sección 1, esta sección es obligatoria.

• Incluye la fecha del metadato y de su revisión, nombre, versión y persona de contacto del metadato, además de cierta información del uso y restricciones de seguridad.

• Toda la información que recoge un metadato se almacena en un simple archivo de texto cumpliendo con todas las definiciones del estándar.

3. Herramientas

Existen varias herramientas o asistentes (wizard) que permiten generar metadatos, entre ellas MetaLite [6], Tkme [7], CorpsMet95 [8], Catmdedit [9] y ArcCatalog [10]. Los editores tienen la intención de simplificar el proceso de creación de metadatos que se ajustan a la norma.

MetaLite [6] es una de las más utilizadas, es gratuita, incluye el idioma español y funciona sobre plataformas Windows. Permite crear y validar metadatos respetando un conjunto de datos mínimos del estándar FGDC [4], abarcando las secciones 1, 3, 6 y 7, incluso genera una base de datos (archivo mdb) almacenando todos los metadatos creados.

Por otro lado, Tkme [7] comparte gran parte de su código con su progenitor, Xtme.

Ambos, Tkme y Xtme, están estrechamente relacionados con mp, un compilador de metadatos formales, cuyo propósito es verificar que la estructura sintáctica de un archivo que contiene metadatos formales se ajusta al estándar FGDC [4] y para volver a expresar los metadatos en varios formatos útiles. Tkme se puede construir para los sistemas Unix, si se desea. Tkme está diseñado específicamente como un puerto de Xtme de Microsoft Windows 95, 98, NT y 2000.

CorpsMet95 [8] es una herramienta de creación de metadatos originalmente desarrollado para el Cuerpo de Ingenieros del Ejército de los EE.UU. (USACE). Inicialmente se trataba de una versión del producto comercial Metagen32. Desde la versión inicial ha sido actualizada por USACE. Al igual que tkme, la interfaz de esta herramienta ofrece al usuario varios paneles.

Otra herramienta de edición de metadatos es Catmdedit [9], que facilita la documentación de los recursos, con especial énfasis en la descripción de los recursos de información geográfica. Se trata de una iniciativa del Instituto Geográfico Nacional de España (IGN) [11], que es el resultado de la colaboración científica y técnica entre IGN y el Grupo de Sistemas de Información Avanzados (IAAA) [12] de la Universidad de Zaragoza con el apoyo técnico de GeoSpatiumLab (GSL) [13].

Dos editores de metadatos se proporcionan con ArcCatalog [14]. Un editor permite crear documentos siguiendo el estándar para Metadatos Digitales Geoespaciales de la FGDC [4]. El otro editor permite documentar datos según la norma ISO 19115 [14], metadatos de información geográfica, que sólo es compatible con los elementos de metadatos básicos definidos por esa norma.

4. Calidad de la Información Geográfica

Como se observó en la figura 1, la sección 2 del estándar de Metadatos de la FGDC [4] corresponde a la Calidad. Esta sección se refiere a la calidad de los datos de acuerdo a la precisión y la consistencia lógica de los mismos. Ya en el momento de pensar en la descripción de los datos, metadatos, se está considerando la calidad de la información geográfica.

En [15] se menciona que los datos geográficos son datos de propósito general que tienen un ciclo de generación y uso muy diferente a los datos tradicionales de negocio de las empresas. Son generados por organizaciones especializadas que los brindan a las organizaciones que los usarán para diferentes aplicaciones. Por esto, es muy importante conocer la calidad de los datos que se pueden obtener y poder evaluar si son adecuados para el uso que se les quiere dar. En este mismo trabajo se subraya que, por la forma tradicional de generación de los datos geográficos, el propio proceso establece parámetros de control del proceso. De este modo, los productores de datos geográficos realizan controles internos sobre la calidad de los datos en relación a la especificación del producto. Algunos de los parámetros son, por ejemplo: escala, extensión (cobertura), nivel de detalle. Por ejemplo, en un mapa de escala mundial las ciudades serán puntos, mientras que en un mapa de escala nacional las ciudades pueden ser representadas por polígonos que muestren su extensión e, incluso, en un mapa de una ciudad se consideran otros datos y los límites de la ciudad.

En los mapas tradicionales (en papel), en la misma hoja de impresión se anexaban metadatos sobre el mapa (escala, fecha de edición, leyenda, etc.). Ahora, con los datos geográficos digitales, puede suceder que se cuente con un conjunto de datos sin sus metadatos. Esto hace más complejo el poder evaluar si el conjunto de datos es apropiado para el uso que se le quiere dar. Sumado a esto, cada vez se cuenta con mayor cantidad de datos geográficos generados a demanda para un proyecto o dominio que pueden, o no, ser apropiados para otro proyecto.

4.1 Normas de Calidad

En el trabajo realizado en [15] se presentan un conjunto de normas de Calidad, y menciona que la comunidad de generadores de datos geográficos ha propuesto estándares que especifican desde la forma de representar, intercambiar y publicar, hasta cómo manipular la información geográfica. Estos estándares se han ido normalizando a través de la familia de normas ISO 19100 [16]. Dentro de esta familia se definen las siguientes normas sobre calidad:

- ISO 19113 – Principios de la Calidad.
- ISO 19114 – Procedimientos de Evaluación de la Calidad.
- ISO 19138 – Medidas de la Calidad.

Estas normas buscan estandarizar los aspectos de identificación, evaluación y descripción de la calidad de los datos geográficos. De este modo, se pueden comparar productos, evitar informaciones ambiguas y facilitar la elección y el uso de los datos geográficos.

Estas normas, junto con las de metadatos (ISO 19115), facilitan la comunicación entre productores y consumidores de datos geográficos.

Para describir la calidad de los datos geográficos se distinguen dos tipos de atributos: cualitativos y cuantitativos. Los datos cualitativos son solamente descriptivos e incluyen, por ejemplo: la historia de los datos (el linaje), los casos de uso para los que fueron recopilados y el propósito para el que fue generado el conjunto de datos. Aquí también se considera la especificación que se tomó en cuenta para la generación de los datos geográficos (de acuerdo a la norma ISO 19131 – Geographic Information – Data Product Specifications). Los datos cuantitativos son los que se pueden medir de acuerdo a las medidas establecidas en la norma ISO 19138 y que se implementan en base a las normas ISO 28593 e ISO 3159 que tratan de muestreo y procesos estadísticos en general. En la Figura 2 se muestra el ciclo de evaluación de la calidad de los datos geográficos y los puntos dónde intervienen las normas. Aquí se asume que los datos están almacenados en una Base de Datos Geográfica – BDG. También se asume que se está trabajando con un conjunto de datos geográficos que ya fueron generados y están listos para su evaluación de calidad. No consideramos los controles correctivos que se puedan haber aplicado en las diferentes etapas del proceso de generación.

Para definir el Modelo de Calidad, las normas establecen cinco grandes Elementos de la calidad para los datos geográficos. Estos son:

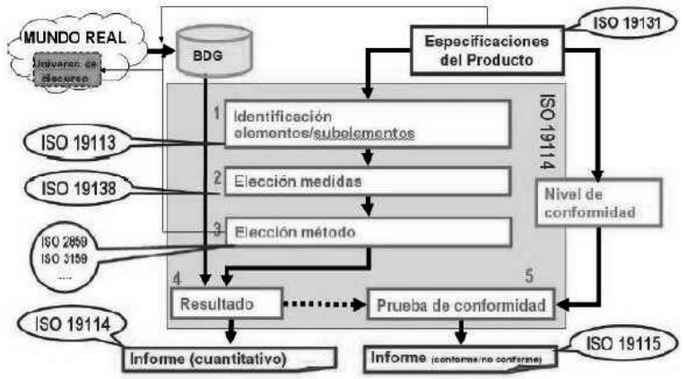


Figura 2.- Relación entre los procesos y las normas de Calidad de la Información Geográfica

1. Compleción (traducción oficial española para Completitud): describe los errores de omisión/comisión en los elementos, atributos y relaciones.
2. Consistencia Lógica: adherencia a las reglas lógicas del modelo, de la estructura de datos, de los atributos y las relaciones.
3. Exactitud Posicional: exactitud alcanzada en la componente posicional de los datos.
4. Exactitud Temporal: exactitud alcanzada en la componente temporal de los datos.
5. Exactitud Temática: exactitud de los atributos y de la corrección de las clasificaciones de los elementos y sus relaciones.

Para cada Elemento se definen Subelementos que ayudan a definir más precisamente lo que se desea medir. Estos subelementos se presentan la tabla 1 vinculados al Elemento que los contiene.

La norma propone luego un conjunto de descriptores para cada subelemento de forma de estandarizar su documentación:

Elemento	Subelementos
Compleción	Omisión – Comisión
Consistencia Lógica	Conceptual – de Dominio – de Formato – Topológica
Exactitud Posicional	Absoluta – Relativa
Exactitud Temporal	Exactitud de la Medida – Validez Temporal – Consistencia Temporal
Exactitud Temática	Corrección de: la clasificación, atributos cualitativos, atributos cuantitativos.

Tabla 1.- Subelementos definidos para cada Elemento

- Ámbito: alcance de aplicación del subelemento, puede ser una sección de un conjunto de datos.
- Medida: definición del tipo de prueba a realizar y sus parámetros.
- Procedimiento: metodología para ejecutar la medición.
- Resultado: resultado para la medida, puede ser un valor, conjunto de valores o su evaluación frente a un umbral determinado por los requerimientos de calidad.
- Tipo de Valor: se asocia al tipo de resultado: numérico, booleano, graduado.
- Unidad del Valor: unidad del tipo de valor. Por ejemplo, para una medida de precisión posicional se puede dar en metros, kilómetros u otra unidad.
- Fecha: fecha de realización de la medida. Es particularmente importante para las evaluaciones relativas a la temporalidad.

Esto brinda un modelo genérico y extensible (ya que la norma permite definir nuevos elementos y subelementos) para la medición y el reporte de la calidad de datos geográficos.

Normas sobre Metadatos y Calidad de Información Grográfica pueden ser consultadas en [17] y [18].

5. Bibliografía

http://www.iiap.org.pe/
http://www.agrimensoreschubut.org.ar/Ptsig/metadatos.htm
http://www.sigfam.com.ar/content/view/102/2/
http://www.fgdc.gov/
http://www.isotc211.org
http://edcnts11.cr.usgs.gov/metalite/
http://geology.usgs.gov/tools/metadata/tools/doc/tkme.html
http://www.sco.wisc.edu/wisclinc/metatool/cormet95.htm
http://catmdedit.sourceforge.net/
http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=Editing_metadata
http://www.ign.es/ign/main/index.do
http://webdiis.unizar.es/~zarazaga/workPage/docencia/ingSoft1/index.html
http://www.geoslab.com/
http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020
Ing. Sosa, Raquel. "Calidad de Datos Geográficos". Curso: Calidad de Datos
http://www.fing.edu.uy/inco/cursos/caldatos/).
http://www.eurogeographics.org/documents/Guidelines_ISO19100_Quality.pdf
http://www.mappinginteractivo.com/plantilla-ante.asp?id_articulo=1457
http://www.mappinginteractivo.com/prin-ante2.asp?id_periodo=137

9. Taller de Formación.NET

Ing. Gustavo Guimerans
A/C. Nicolás Sampietro
A/C. Emiliano Martínez

Laboratorio de Integración de Sistemas
Introducción a .NET Framework

Agenda

¿ Qué es .NET Framework?
- Componentes Fundamentales
- Funcionamiento Interno del CLR
- Bibliotecas Principales
- Características de .NET
- Herramientas de Desarrollo .NET
- Lab. Punto de venta

¿Qué es .NET Framework?

Tecnología de desarrollo de aplicaciones empresariales, compuesta de:
- Entorno de Ejecución (Runtime)
- Bibliotecas de Funcionalidad (Class Library)
- Lenguajes de Programación
- Visual Basic
- C#
- F#
- C++
- Compiladores
- Herramientas de Desarrollo (IDE VS2010 & Tools)
- Guías de Arquitectura

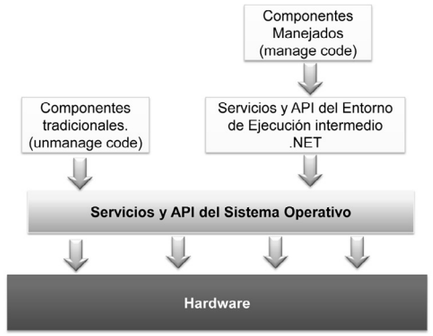
Plataforma de Ejecución Intermedia

Agenda

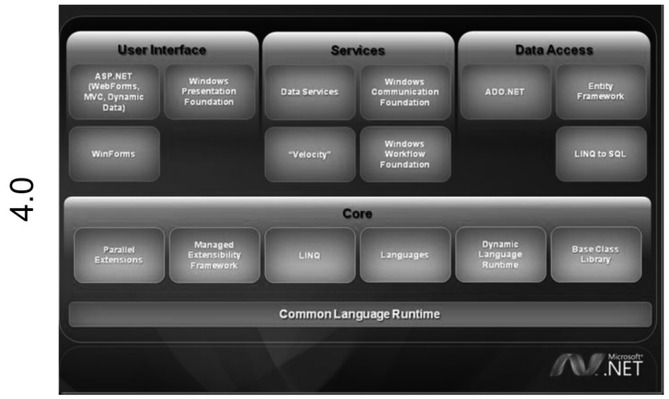
¿Qué es .NET Framework?
- Componentes Fundamentales
- Arquitectura

- Common Language Runtime (CLR)
- Microsoft Intermediate Language (MSIL)
- Assemblies
- .NET Class Library
- Common Language Specification (CLS)

CLR - Arquitecturas de Ejecución de Aplicaciones



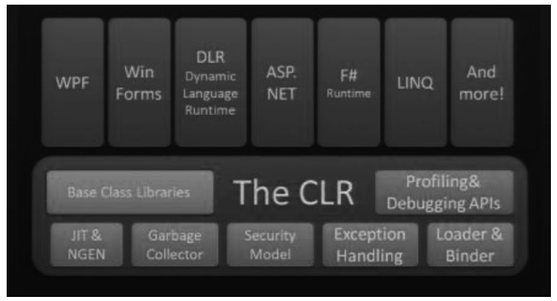
Arquitectura del .NET Framework



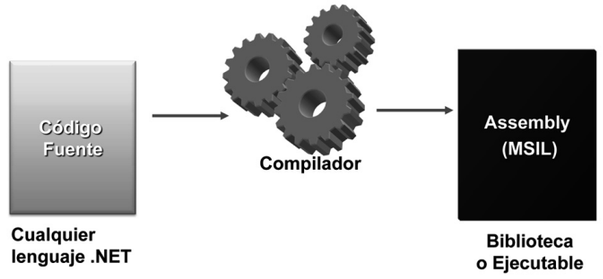
CLR – Common Language Runtime

- El CLR es el motor de ejecución (runtime) de .NET.
- Características.
- Compilación Just-In-Time (JIT).
- Gestión automática de memoria (Garbage Collector).
- Gestión de errores consistente (Excepciones).
- Ejecución basada en componentes (Assemblies).
- Gestión de Seguridad.
- Multithreading.

CLR – Componentes Internos



CLR – Proceso de Compilación

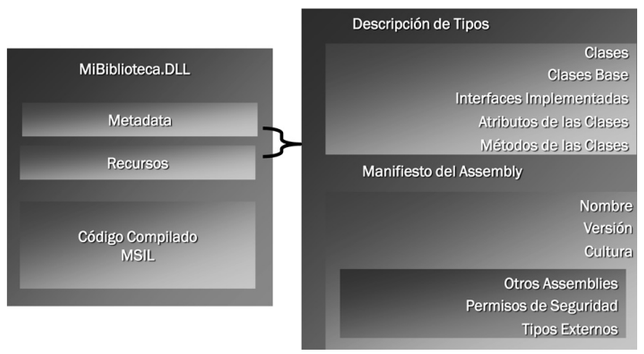


CLR - MSIL

```
.method private hidebysig static void Main(string[] args) cil
      managed {
.entrypoint
maxstack 8
L_0000: ldstr "Hola Mundo"
L_0005: call void [mscorlib]System.Console::WriteLine(string)
L_000a: ret
}
```

¿Qué es un "Assembly"?

- Un Assembly es la unidad mínima de ejecución, distribución, instalación y versionado de aplicaciones .NET



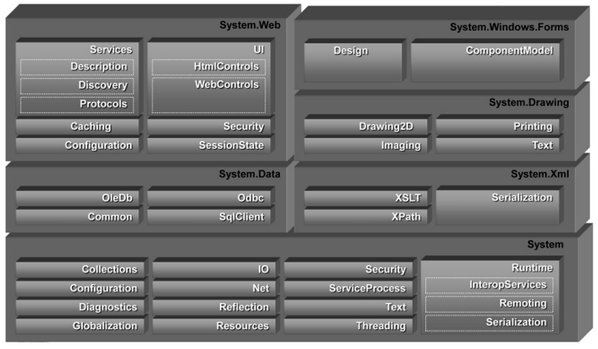
Assemblies - Aplicaciones .NET

- Uno o más Assemblies (.dll, .exe).
- Al ejecutar una aplicación, ¿cómo ubico los assemblies necesarios?
- El Class Loader busca en el directorio local (preferido).
- Global Assembly Cache (GAC).
- Diferentes aplicaciones pueden usar diferentes versiones.
- Actualizaciones más simples.
- Desinstalación más simple.

NET Framework Class Library

- Conjunto de Tipos básicos (clases, interfaces, etc.) que vienen incluidos en el .NET Framework.
- Los tipos están organizados en jerarquías lógicas de nombres, denominados NAMESPACES.
- Los tipos son INDEPENDIENTES del lenguaje de desarrollo.
- Es extensible y totalmente orientada a objetos.

.NET Framework Class Library

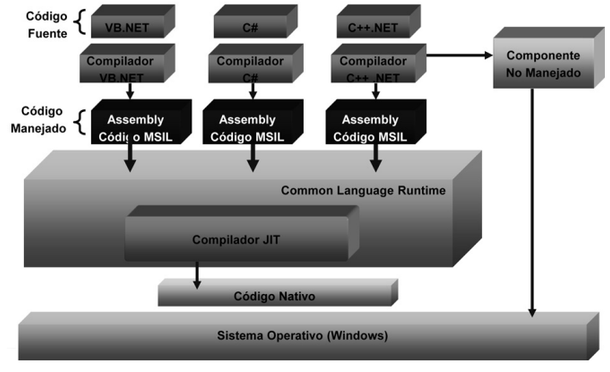


Agenda

¿ Qué es .NET Framework?

- Componentes Fundamentales
- **Funcionamiento Interno del CLR**
- Modelo de Ejecución
- Common Type System

Modelo de Ejecución del CLR



CTS (Common Type System)

- Define un conjunto común de "tipos" de datos orientados a objetos.
- Todo lenguaje de programación .NET debe implementar los tipos definidos por el CTS.
- Todo tipo hereda directa o indirectamente del tipo System.Object.
- Define Tipos de VALOR y de REFERENCIA.

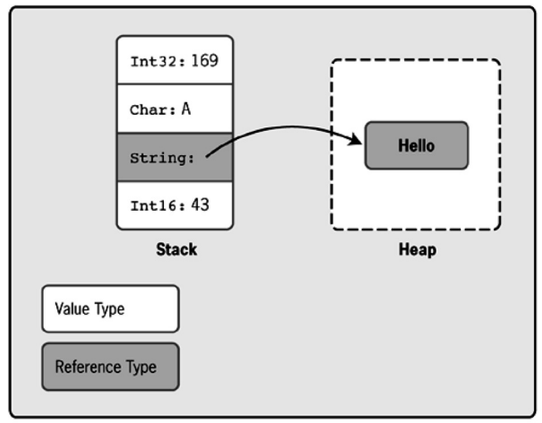
La Memoria y los Tipos de Datos

El CLR administra dos segmentos de memoria: Stack (Pila) y Heap (Montón).

- El Stack es liberado automáticamente y el Heap es administrado por el GC (Garbage Collector).

La Memoria y los Tipos de Datos.

- Los tipos VALOR se almacenan en el Stack.
- Los tipos REFERENCIA se almacenan en el Heap.



Agenda

¿ Qué es .NET Framework?
- Componentes Fundamentales
- Funcionamiento Interno del CLR
- Bibliotecas Principales
- Características de .NET

Características de .NET

Entorno de Ejecución robusto y seguro:
- Gestión automática de memoria.
- Manejo de excepciones.

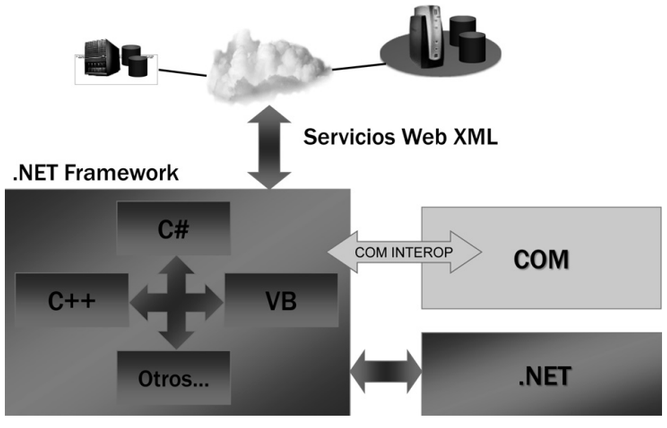
Independiente del lenguaje de programación.
- Libertad en la elección del lenguaje (o mixto).
- Herramientas de desarrollo compartidas.

Interoperabilidad con código existente:
- Unmanage code. COM.

Simplifica la instalación y administración de las aplicaciones:
- GAC, múltiples versiones.

Extensible:
- Las clases pueden ser extendidas usando herencia.
- Herencia entre distintos lenguajes.

Interoperabilidad



¿ Qué es .NET Framework?

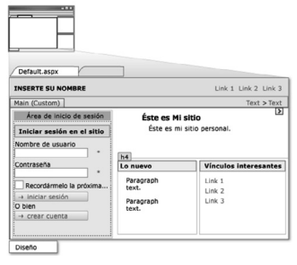
- Componentes fundamentales
- Funcionamiento interno
- Bibliotecas Principales
- Ventajas de .NET

Herramientas de Desarrollo .NET
- Visual Studio 2010
- SQL Server 2010 Express
- SQL Server Managment Studio.
- IIS7

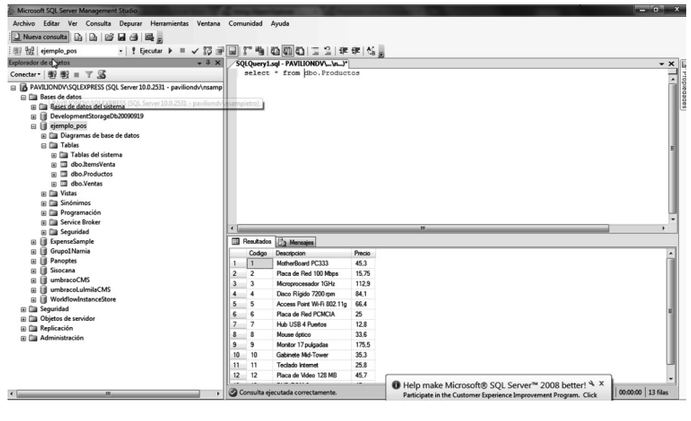
Microsoft Visual Studio 2010

- “Solución” es un contenedor de proyectos.
- “Proyecto” es un contenedor de: archivos fuente, conexiones a base, recursos etc.

- Diferentes tipos de proyectos, (VS Projects Templates).
- Area editor o diseñador depende del tipo de archivo que se esta usando.



SQL Server Management Studio



• SQL Server Management Studio es un entorno integrado para obtener acceso a todos los componentes de SQL Server, configurarlos, administrarlos y desarrollarlos.

• Con SQL Server Management Studio, el programador y el administrador de bases de datos pueden desarrollar o administrar cualquier componente del Motor de base de datos.

Agenda

¿ Qué es .NET Framework?

- Componentes Fundamentales
- Funcionamiento Interno del CLR
- Bibliotecas Principales
- Características de .NET
- Herramientas de Desarrollo .NET
- Lab. Punto de venta

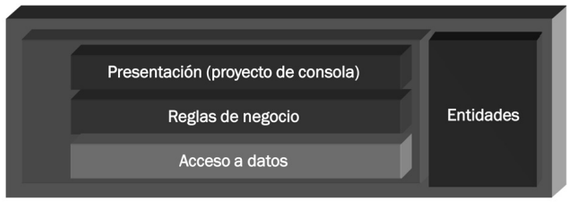
Lab 0

```
/* HolaMundo.cs */

public class HolaMundo{
  public static void Main() {
  //Imprimir hola mundo
  System.Console.WriteLine("Hola !");
  }
}
```

Invocamos al compilador C# con:
>csc Holamundo.cs

Lab. Punto de venta

• La aplicación "Punto de venta" posee una arquitectura en tres capas (presentación, reglas de negocios y acceso a datos).



Funcionalmente, la aplicación permite:

- Listar una serie de productos disponibles para vender.
- Iniciar una nueva venta.
- Agregar un ítem a la venta (se agrega un producto al carrito de compras).
- Listar los productos contenidos en el carrito.
- Confirmar la venta (obteniéndose el total a pagar).
- Cancelar la venta (vaciando el carrito de compras).
- Salir (cancelando las ventas no confirmadas).

Lab. POS. Presentación

- Assembly .EXE, proyecto de consola.
- Main(), punto de entrada a la aplicación.
- Utiliza el patrón command, para las opciones del menú.
- Este proyecto contiene, App.config, (Cadenas de conexión, variables etc.).

Presentación: (Proyecto de consola)

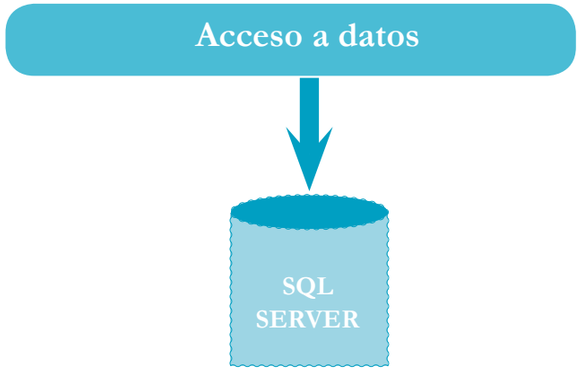
Lab. POS. Reglas de negocio

- Assembly .dll, librería.
- Clases que representan Entidades (Venta producto, ItemVenta).
- Clases que representan maestros de las entidades. (CatalogoProductos, CatalogoVentas).

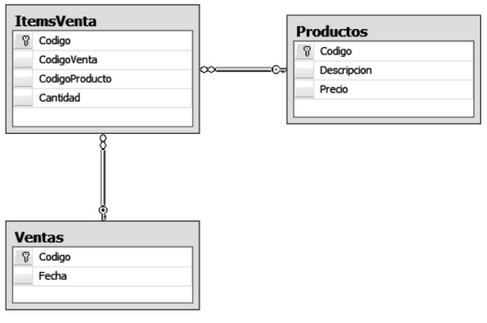
Reglas de negocio

Lab. POS. Acceso a datos

- Assembly .dll, librería.
- Contiene una clase que encapsula y simplifica el acceso a datos con ADO.NET. (BaseDatos).
- Clase que representa un error (BaseDatosexception).



Lab. Punto de venta



Bibliografía

MSDN .NET Framework 4
ohttp://msdn.microsoft.com/es-es/library/w0x726c2.aspx
DCE
ohttp://mslatam.com/latam/msdn/comunidad/dce2005/
SQL Server Managment Studio
ohttp://msdn.microsoft.com/en-us/library/ms174173.aspx

13. Integración de equipos multidisciplinarios: Agrotecnologías

Multi-agent systems applied to land use and social changes in Rio de la Plata basin (South America)

Corral, Jorge, Facultad de Ingeniería, Uruguay
Arbeletche, Pedro, Facultad de Agronomia, Uruguay
Burges, Julio César, INTA-EEABalcarce, Argentina
Morales, Hermes, Instituto Plan Agropecuario, Uruguay
Continanza, Guadalupe, INTA-EEA Balcarce, Argentina
Couderc, Jorge, INTA-EEA Bordenave, Argentina
Courdin, Virginia, Facultad de Agronomia, Uruguay
Bommel, Pierre, CIRAD-Univ. Brasilia, Brazil

Abstract

Dynamics in agrarian systems of Uruguay and Argentina present some positive aspects as well as other potentially devastating. Traditional producers have a production strategy based on looking for a balance between cattle and agriculture production, alternating pasture and crops. A new actor: investment-fund-managers rent the land for agriculture production, and from our team's discussions emerged that they follow a strategy similar to that of financial capital: decide what to do in terms of the expected net profit. Economical, ecological and social consequences could be expected. Modeling and simulating with Multi-Agent-Systems was used for exploring the system's evolution with the objective of improving our understanding of the agrarian system and to contribute to the envision of possible effects on land use caused by changes in product prices and/or policy changes. The model considers the soil resource as having productive potential and assumes that each traditional producer annually decides whether to change or not its production activity over 25% of its land units or to rent to investment fund managers. A six year database with historical production activity revenues and product's prices was used for the simulation, where each simulated year randomly chooses from this database. The first results of these simulations generate questions about the dynamics of the natural resources, challenge the survival of traditional farmers and anticipate landscape changes associated to economic, ecological and social changes. A strong variability was observed from year to year in respect to land use. Results show that if the current price structure is maintained as well as the relation between net profits of agriculture and cattle, then a tendency to expand Investment Fund Managers' lands will occur. We conclude that the newly arrived Investment Fund Managers tend to induce a rent activity in traditional producers as well as a substantial increase in agricultural activities (decreasing cattle activities). The historical cattle-agricultural model well known in Uruguay and Argentina has been challenged by market-imposed conditions.The simulation shows that social effects should also be foreseen.

Keywords: land use change; simulation models (modelling); farm(farmers) strategy(ies).

Introduction

Recently observed changes in agrarian systems of the Rio de la Plata Basin (MERCOSUR, South America), with important similarities between the different regions, should be given some thought in the sense of possible economical, ecological and social consequences, which could be anticipated from these changes in the farming sector due to the explosive introduction of soybean crops and a new push of the forest industry. The increase in the soybean production is common to all countries in Rio de la Plata Basin since it is mainly produced by large companies and had the effect of increasing the rent value, and therefore the values of properties that could put pressure on traditional cattle producers to incorporate more intensive practices. The evolution in the structure of farms has lead to a new kind of actor (agricultural investment fund managers -IFM from now on- that rent land) which together with all these changes raise questions about which could be the positive and negative aspects, what kind of actors will still be present in the near future and if it possible to state if agriculture is going through a concentration process similar to those observed in other sectors of the economy? This soybean expansion process takes over new lands now devoted to agriculture or leaving behind traditional activities. For example, in Uruguay between 2000 and 2006 the total agricultural area increased in 17% due to the expansion of soybean crops which have multiplied by 25 (in area) in just 5 years. Among soybean producers, 6% have control of 40% of the sowed area; while among the whole agricultural area, 1% of the producers have control of 45% of the sowed area.

This expansion in the production takes place in an agrarian structure characterized by an increasingly economic concentration, which affects thousands of producers, especially small ones since in just 5 years 45% of them were no longer agricultural producers. (Arbeletche et al, 2006) This expansion process is also characterized by the denationalization of the agricultural production, the coming of a soybean complex related to a monopolistic offer of inputs (especially seeds and machinery), an almost monopolistic export demand, and a set of technologies driven by a few foreign companies.

As in Argentina, Brazil and Paraguay, this expansion is not the result of a planned productive one, based on social and economic development objectives. Instead it is the result of capital advance (mainly financial capital) due to: new conditions result of the disappearance of legal regulations that existed until the 90's; technological changes related with direct sowing and transgenic crops; and finally the increasing demand of agricultural products by the international market.

Also, aspects such as biodiversity, soil fertility conservation and, in general, the capacity of ecosystems to satisfy human needs are all related to land use, and therefore, related to economic or socio-political disturbances (Paruelo et al. 2006).

In order to understand and predict land use change effects historical reconstructions should be made that identify the essential factors and develop models that help us explore future scenarios. These models should show these dynamics at different levels, including the global scale (Lambin et al. 2006).

The current agricultural situation with these unprecedented changes requires us to imagine and examine actions that could leverage the positive aspects and mitigate the negative ones. In this context, this work refers to a Uruguayan region and can be seen as a case study with relatively abundant information, where a methodology can be adjusted and later be used in similar circumstances, or in comparison to other regions of the Rio de la Plata Basin. Displace

Objectives

The objectives of this work are:

• Identify and model the strategies followed by the different kinds of farm producers present in Uruguay, in order to analyze and understand their long term consequences.

• Develop a model to allow the simulation of the evolution of the different kinds of producers and land uses.

• Perform these exploratory and prospective simulations using a Multi-Agent System constructed by our multidisciplinary team over the Cormas simulation platform (Cormas 2006).

This first approach focuses on analyzing possible evolutions in land use (on lands perfectly suitable for crops) as well as the evolution in the different kinds of producers. This approach is supported by various ongoing works from different team members, such as the construction of a typology of producers' behavior and their corresponding organization of production activities, among others.

This work is part of the TRANS programme (Transformation de l'élevage et dynamiques des espaces de la Agence Nationale de Recherche - France).

Methods and materials

The general characterization of the agrarian dynamics was created from secondary information, national and international statistics and from other documents that allow a comparison with other regions as well as with other historical information.

The typology used was developed from the General Farming Census (year 2000) and the farming polls (years 2002 to 2005) of the Economic Research head office of the Livestock, Agriculture and Fishing Ministry of Uruguay (Arbeletche et al. 2006). The classification was done using the Cluster Analysis methods from the (Sparks) algorithms contained in the SPSS software (version 10).

The methodology that was followed consists of three stages:

Stage I: A multidisciplinary team was formed with researchers, teachers and extensionists from Uruguay and Argentina that complement their competences in understanding a phenomenon that is present in various areas of the Rio de la Plata Basin, with their similarities and differences.

Stage II: In order to make the model (understood as the construction of an image that highlights those aspects of interest for the modeler, ruling out others) we used UML (Unified Modeling Language, Fowler 2003). The use of a common modeling language (such as UML) enables many people to communicate with each other with little ambiguities (Krutchen 2003) and allows us to understand, analyze, communicate and improve a given situation or reality. UML defines a model as a set of diagrams, accepting from the beginning that no single diagram can represent an entire system. UML proposes 13 different kinds of diagrams: six of

these are for describing the structure of a system (being the Class Diagram the most widely used) and seven are for describing the dynamic of a system (being the Activity Diagram and the Sequence Diagram the most widely used) (Fowler 2003).

Stage III: We define simulation as the computer implementation of a model that allows for exploring its evolution as well as proving the coherence and consistency of its construction. In order to simulate a model where bio-physical and social subsystems interact, the model should:

1. Take into account the dynamic present in decision-making. For that it should incorporate qualitative information in the form of decision rules;
2. Show the dynamic of this interaction, and
3. Include heterogeneous components with quantitative and qualitative dynamics.

This simulation, using Multi-Agent Systems (MAS from now on) is supported by object oriented programming and it is getting more and more attention as a tool, especially adapted for these kind of analysis. MAS appears as a tool especially adapted when trying to simulate the functioning and evolution of systems composed by heterogeneous agents interacting among themselves that are influenced by their location in space, in situations that can or cannot be of equilibrium (Bonabeau, 2002; Weiss, 1999; Bousquet, 2006; Janssen 2002). When trying to study systems that include human behavior, prospective simulation take distance from the normative approach that has been common in many sciences, and this difference is important enough as to classify it as a "new kind of science" (Bradbury 2006).

Axelrod (1997) proposes that agent-based simulation constitutes a third way of acquiring knowledge, different from the usual deduction and induction methods. The potential of MAS to study the dynamics of natural resources that interact with society has been identified more than a decade ago (Bousquet, 2006; Janssen 2002). Their capacity to simulate social systems (Gilbert & Terna 1999) and its interaction with heterogeneous elements such as those that dynamically characterize ecological systems, place them as an adequate tool for such situations (Parker et al. 2002; Moran & Orstrom 2005).

The actual way of modeling depends on the right judgment of the team that is involved (Ericsson & Penker 2000), and the task of defining the level of abstraction is quite sensitive since the main capacity of the modeler consists of choosing what to include and what to exclude from the model, keeping in mind the objectives (Schmuller 2004, Holland 1998).

According to Le Page & Bommel (2006), a MAS is a set of agents with the ability to act and communicate; with perception, communication, production, consumption and data transformation within an environment; a topological space; a whole that contains agents and objects; a resource for communication and action; passive objects such as resources, organization plans, or ways coordinate represented by the set of rules and relations.

Each agent presents a collective behavior, consequence of its perceptions, representations and interactions with the environment and with other agents, and it communicates with these, it has a perception of them as well as of the environment, and it perceives and acts over objects (Janssen 2002; Weiss 1999). In equation-based models, agents are frequently (and sometimes even implicitly) assumed as representing an average behavior, so these approaches cannot take into account the interactions between agents or their heterogeneity.

Results

The different kinds of producers

Beginning from the typology created by Arbeletche and Carballo (2006) two subsets of producers were identified: first, traditional producers that integrate into their production system crop and pasture rotations, as well as cattle production; and second, newly arrived farmers (IFMs) that base their productive systems in continuous crops over rented land. Within the first subset of traditional producers, a second classification was done identifying three kinds of traditional producers according to their amount and combination of farm resources (land, work and capital): family producers (also called "small producers"), medium size entrepreneurs (also called "medium producers") and full size entrepreneurs (also called "big producers").

Description of the simulation model

The model shows the interactions generated through the use of the land (agriculture and cattle) and the ownership of the land (rented or owned). The model looks forward to generate knowledge about these aspects and understand the relation among traditional producers and IFMs. The strategy of the latter consists of renting land (plots) in order to intensively and continuously produce soy bean crops. The model also considers that traditional producers (including small, medium and big) who have a history of combining cattle and crops, are profit-sensitive, meaning that they will try to practice whatever activity is more profitable.

The model simulates the behavior of both, traditional producers and IFMs and assumes that the three kinds of traditional producers behave in the same way (that means that their rules are the same) but they differentiate in the amount of resources they manage (that means the number of plots they own and/or operate). Therefore traditional producers risk their properties, while IFMs manage capital (other people's money).

In our model, IFMs rent (and eventually release) plots as long as traditional producers are willing to put some of their plots to rent (and eventually recover them). This means that the initiative of whether to rent or not is taken by the traditional producers according to their decision rules, represented by a UML Activity Diagram (which will be later presented).

The main assumptions of the model are the following:

- Traditional producers can give up for rent one or more of their plots,
- They can only give their plots for rent to the IFM,
- The rented plots are always used for continuous crops (the only activity of IFMs) at the very moment they are rented, and it can or cannot coincide with its previous use,
- Traditional producers can buy and sell plots between themselves, as well as give up for rent to the IFMs, which can only rent plots to traditional producers (therefore IFMs cannot buy land).

Model's structure

Figure 1 presents the UML Class Diagram (Fowler 2003) with the (simplified) structure of the model. Class Diagrams graphically show the main components of the model (classes)

and their associations (relations between classes). Each component (class) contains its name (first section), its attributes (second section) and its operations (third section) and during the simulation run, each individual element will be an instance of some class (for example, producers A, B and C will all be instances of the class Traditional). The relations among classes are typically an association (represented by an open-ended arrow) or a specification (represented by a close-ended arrow).

The right-hand side of the diagram represents the resources (the Plot class) and their use (classes LandUse, Cattle, SoyBean and Empty). Each plot (land unit) has just one use at a time. Each plot (100 hectares) can be either rented (to the IFM) or sold (to other traditional producers). When they are not exploited their state is empty. Exploitation then consists of choosing between cattle and soybean. Each productive activity has its own cost and price that evolve according to the market (this evolution will be discussed later).

The left-hand side of the diagram represents the agents present in the model: the classes InvestmentFundManager and Traditional, both subclasses of Producer. Any producer can manage a set of N plots (represented by the relation Producer –manages Plot) but only traditional producers can own a set of N plots (represented by the relation Traditional –owns Plot). According to the number of plots owned by a traditional producer we further classify them in "small", "medium" and "big" producers (represented by the attribute type of the Traditional class). Only traditional producers can buy and sell land (represented by the operations buyPlot() and sellPlot()) since this is a strategy that IFMs do not have (they systematically choose to sow continuous crops –soybean– over rented plots using high technology which gives them a productivity 30% higher than traditional producers).

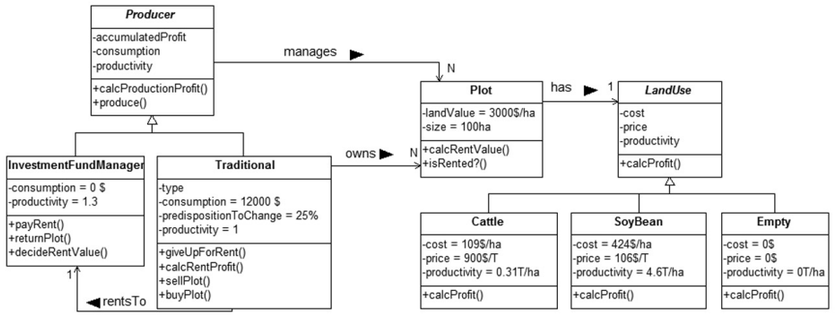


FIGURE 1: CLASS DIAGRAM

Traditional producers can use their plots in the following ways (represented by the relation Plot has→ LandUse):

- continuous crops (every year with double crop, covering 100% of the surface with a winter crop and 80% with a second summer crop),
- cattle fattening,
- rent the plot to the IFM, or
- leave it empty.

The decision of what activity to perform depends on which activity is the most profitable (gross margin) for the traditional producer. The current version of the model assumes that all three kinds of traditional producers are in the same conditions to perform any of these activities.

Models rules

Figure 2 presents the UML Activity Diagram (Fowler 2003) with the (simplified) strategy of traditional producers (remember that only these kinds of producers are proactive, while IFMs react to the actions of these). The Activity Diagram graphically shows the activities that a traditional producer can perform during a certain year. The diagram has a beginning (represented as a solid circle) and an end (represented as a circle & dot) and in between there are tasks or individual activities (represented as rounded rectangles). These tasks or activities are connected with arrows that represent the flow within the diagram and this flow can be controlled by decisions (represented as rhombus). Each decision has a set of outgoing flows, each one with a guard (represented as a boolean yes/no expression within) and only one of these guards can be true at a time (therefore choosing that outgoing flow).

The diagram could be divided in three sections: left, center and right. The left-hand side corresponds to the situation in which the traditional producer presents good levels of accumulated profit that year (remember that the diagram shows the decision rules for one year and is "executed" each year of the simulation period). These "good levels" are defined as having enough accumulated profit for (at least) the next two years (that means that he/she can produce and consume for at least two more years). Under this situation the traditional producer could try to regain its rented plots (if this activity is not more profitable than cattle or soybean). Eventually the traditional producer could even buy one (or more) plots. The right-hand side of the diagram corresponds to the opposite situation: the accumulated profit is not enough for the next year, so the traditional producer is faced to, firstly, try to give up plots for rent, and secondly, try to sell one by one their plots, until either of two things happen: they can upfront the next year, or declare bankruptcy (and leave the simulation). The center part of the diagram corresponds to the intermediate situation: the traditional producer has enough money to face only the next year. In all cases (except of course when declaring bankruptcy) the traditional producer plans the activities for next year (considering renting an activity) and according to market prices (which can evolve and will be discussed later) they will produce (or rent) up to 25% of their plots. This percentage (which can actually change, but for clarity reasons was fixed in 25% in the diagram) is represented as the predispositionToChange attribute of the Traditional class. This attribute is necessary since if not present, traditional producers could change 100% of their managed plots from one year to the other.

IFMs' activities and rules are much simpler, since each year they produce soybean crops over all the plots they have rent. Even though these kind of producers are not proactive (meaning that traditional producers are the ones that offer their plots for rent) they can decline a rent offering if the price of the rent is too high (meaning that they would have no profit with such high rents). They can also return rented plots in that situation. The simulation model also allows IFMs to determine the price of rent (this is a parameter that can be turned on or off and it will be discussed later when showing the simulation results for different input parameters).

Simulation initialization

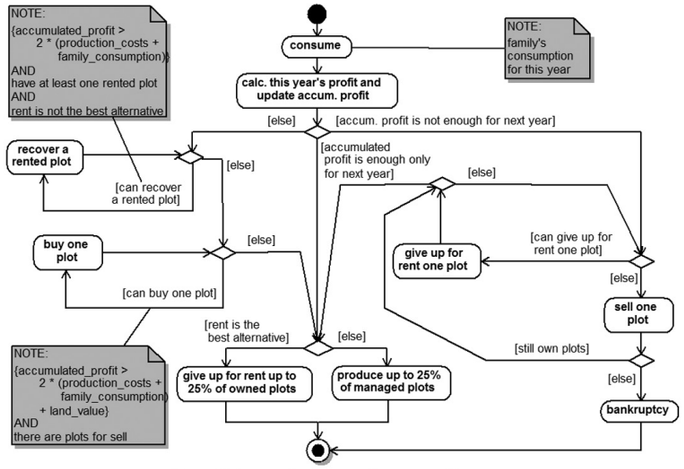


FIGURE 2: ACTIVITY DIAGRAM FOR TRADITIONAL PRODUCERS

The simulation assigns to each kind of traditional producer (small, medium & big) a certain amount of land units (plots) with a randomly selected land use that can be cattle fatening or continuous crop. Each plot (according to its use) has a certain production level that corresponds to average values of Uruguay. Each plot can be bought, sell or rented at market values. All values of products have been average market values for the last five years, and are the same for all producers. Rent and property values are the same as market values.

All plot yields are the same for all traditional producers.

At the beginning of the simulation (time step zero) each traditional producer is given a randomly selected activity, and while the simulation runs, they change this activity according to their decision rules (that is, according to that activity that is most profitable) due to the evolution of prices. Giving up for rent one or more plots is also considered an activity. The initial distribution of the number of each kind of traditional producers was taken from the results of the typology performed over traditional farmers corresponding to the year 2005. The capital of each traditional producer is initiated in zero and increases with each randomly assigned plot. When the simulation runs, each time the traditional producer buys, sells, rents or recovers a rented plot, this capital is updated. This capital is also updated when calculating the profits for each year due to the activity of each managed plot.

The current version of the model assumes that all traditional producers have the same annual cost of living (consumption) which is subtracted from the profits for that year.

4.2.4 Simulation

344'¡which are a subset of the combination of three parameters: the way the rent value is determined (this value can be: a) determined by the IFM as 1 $ higher than the best alternative –crop or cattle- or b) defined as 35% of the value earned by producing soybean); the evolution of soybean price (this evolution can be: a) sinusoidal or b) increasing sinusoidal, which starts with very low prices for soybean and ends with very high soybean prices) and the presence of the IFM which can be present or not in the simulation. If it is not present, there won't be rented plots, therefore, they won't be able to determine the rent value. The reason that makes the situation where the IFM determines the rent value is supported on the idea that they can pay traditional producers more (for their rented land) than if the traditional producer produces cattle or soybean by himself in his plots (evidently as long as the IFM continues to earn a positive net profit from his activity).

Case 1:

Rent Value: determined by the IFM as the best alternative plus one money unit: rent value = MAX(soybean_profit; cattle_profit) + 1$

Soybean Price Evolution: sinusoidal (ranging from historical min. and max. values).

Presence of IFM: yes (and determining the rent value as indicated above).

Note: the X-axis of both graphs have the same scale so they can be analyzed together.

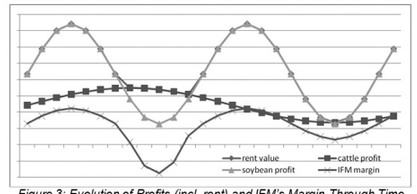


Figure 3: Evolution of Profits (incl. rent) and IFM's Margin Through Time

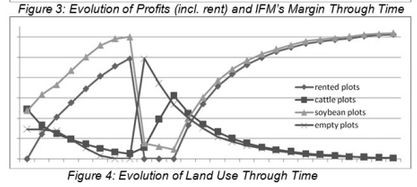


Figure 4: Evolution of Land Use Through Time

Case 2:

Rent Value: defined as 35% of the value of soybean.

Soybean Price Evolution: increasing sinusoidal (starting from very low prices up to very high soybean prices).

Presence of IFM: no (it is not present in the simulation, so no plots will be rented).

Note: the X-axis of all four graphs have the same scale so they can be analyzed together.

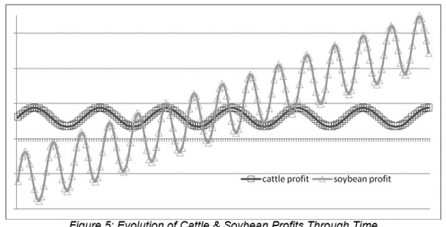


Figure 5: Evolution of Cattle & Soybean Profits Through Time

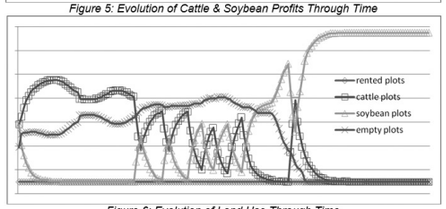


Figure 6: Evolution of Land Use Through Time

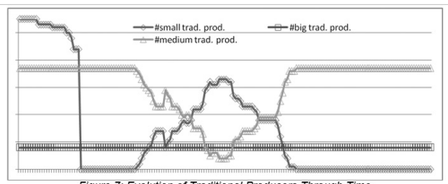


Figure 7: Evolution of Traditional Producers Through Time

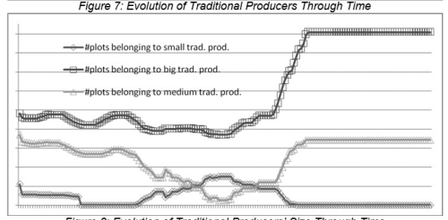


Figure 8: Evolution of Traditional Producers' Size Through Time

Cattle profit is always considered as evolving with a sinusoidal function with historical minimum and maximum values (normal distribution).

Case 3:

Rent Value: determined by the IFM as the best alternative plus one money unit: rent value = MAX(soybean_profit; cattle_profit) + 1$

Soybean Price Evolution: increasing sinusoidal (starting from very low prices up to very high soybean prices).

Presence of IFM: yes (and determining the rent value as indicated above).

Note: the X-axis of both graphs have the same scale so they can be analyzed together.

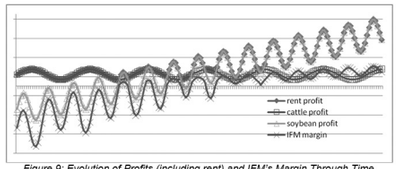


Figure 9: Evolution of Profits (including rent) and IFM's Margin Through Time

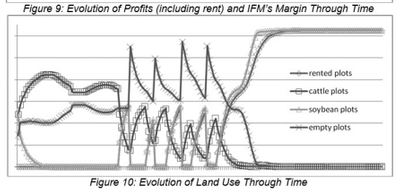


Figure 10: Evolution of Land Use Through Time

The results achieved so far are limited since the model is still under construction. We will extend it introducing variability in traditional producers' productivity, including lands of lower quality and less productive potential.

Conclusions

From the analysis and synthesis of all available data we can conclude that new ways of land use appeared in the Rio de la Plata Basin region, with a steady increase of continuous crops that was not present at the end of the past century. From the modeling and simulation we conclude that:

If the decisions of traditional producers are supported by the expected profit and with a normal price distribution (Case 1):

When soybean prices are good, the IFM rents all plots for agriculture. There are no changes in land property. There is no concentration of land concerning property, but there will be a concentration of land use. Traditional producers do not sell their plots, and they will tend to rent all of them.

If the IFM is willing to pay a fixed rent value in tones of product (Case 2):

It could be the case that for traditional producers is more profitable to produce soybean by themselves (if rent is less profitable) so no rent will occur.

In this case, the simulation showed that big traditional producers would buy land in order to grow more soybean crops, so land property as well as land use concentration would occur. Small traditional producers would tend to extinction (see Figures 7 and 8).

If the price of soybean increases (Case 3):

Even in the case that IFMs do not exist there still is a continuous crop usage. Also, a land property concentration will occur where at first small and medium producers would disappear.

In any case, cattle is moved out to non-farming areas of lower quality lands.

The preliminary simulations using MAS that we have done showed us that this tool has a good potential for exploring the evolution of these kinds of systems.

It should be considered that the results obtained are limited since we deal with a model under construction. However, the model suggests that changes on international prices and policies in countries like Uruguay and Argentina greatly determine that the best quality lands are mainly used for agriculture.

It would be of major importance to reevaluate land use dynamics and its causes, since the survival of small producers is in stake (which represents a large part of total producers and support the existence of multiple rural populations in the Rio de la Plata Basin region). On the other hand, these models allows us to anticipate and act in prevention, facing potential land use changes associated to economical, ecological and social changes.

References

Arbeletche, P., Carballo, C., 2006 Crecimiento agrícola y exclusión: el caso de la agricultura de secano en Uruguay in Proceedings VII Congreso de Alasru, Quito, Ecuador, november 6-10.

Arbeletche, P, Carballo, C., 2006 Sojización y concentración de la agricultura uruguaya in Proceedings del XXXIV Congreso de la Asociación Argentina de Economía Agrícola Córdoba, Argentina, october 18-20.

Bonabeau E., 2002 Agent-Based modeling: Methods and techniques for simulating human systems in Proceedings of the National Academy of Sciences of the USA 99: 7280-87.

Bousquet F., 2006 Multi-agent systems, companion modeling and land use change in Lambin E.F. & Geist H. (eds): Land-Use and Land-Cover Change. Local Processes and Global Impacts. Springer. Berlin Germany.

Bradbury R., 2006 Towards a new ontology of complexity science in Perez P. Batten D. (eds) Complex Science for a Complex World. ANU E Press. Camberra Australia.

Cormas, 2006 Ressources naturelles et simulations multi-agents. CIRAD. URL: http://cormas.cirad.fr

Ericsson H. E., Penker M., 2000 Business Modeling with UML. Business Patterns at Work. OMG Press. John Wiley & Sons, Inc. USA.

Fowler M., 2003. UML Distilled, Third Edition. A Brief Guide to the Standard Object Modeling Language. Addison Wesley. USA.

Gilbert N., Terna P., 1999, How to build and use agent-based models in social science, URL: http://web.econ.unito.it/terna/deposito/gil_ter.pdf/

Holland J.H. 1998. Emergence. From Chaos to Order. Basic Books.

Janssen M. (Ed.), 2002 Complexity and Ecosystem Management: The Theory and Practice of Multi-agent Approaches, Edward Elgar Publishers.

Kruchten P. (2003) The Rational Unified Process: An Introduction, Third edition. Addison Wesley. 302 pp.

Lambin E. F., Geist H., Rindfuss R. R., 2006, Introduction: local processes with global impacts, in Lambin E.F. & Geist H. (eds) Land-Use and Land-Cover Change. Local Processes and Global Impacts. Springer. Berlin Germany.

Lambin E. F.; Geist H. J.; Lepers E., 2003, Dynamics of land-use and land-cover change in Tropical Regions, Annu. Rev. Environ. Resour. 20:205-41.

Le Page C., Bommel P. 2006, A methodology to building agent-based simulations of common pool resources management: from a conceptual model designed with UML to its implementation in Cormas.

CORMAS, in Bousquet F.; Trébuil G.; Hardy B. (eds) Companion Modeling and Multi-

Agent Systems for Integrated Resource Management in Asia. Los Baños (Philippines): International Rice Research Institute. 327-350.

MGAP-DIEA, 2001, Censo General Agropecuario 2000, Montevideo, Uruguay.

Moran E. F., Orstrom E (eds), 2005), Seeing the forest and the trees: Human-environment interactions in forest ecosystems, MIT Press, Cambridge London.

Parker D.C. Berger Th. Manson M., 2001, Agent-Based Models of Land-Use and Land-Cover Change, Report and Review of an International Workshop October 4–7, 2001 Irvine, California, USA Edited by: Parker D.C., Berger T., Manson S.M. URL: http://www.indiana.edu/%7Eact/focus1/ABM_Report6.pdf

Paruelo J.M, Guerschman, J.P.; Piñeiro, G.; Jobbágy, E.G, Verón, S.R.; Baldi, G. y Baeza, S., 2006, Cambios en el uso de la tierra en Argentina y Uruguay: Marcos Conceptuales para su análisis, Agrociencia. Vol. X N° 2 pp. 47 – 61.

Schmuller J. 2004, Sams Teach Yourself UML in 24 hours, SAMS Publishing USA.

Weiss G. (ed), 1999, Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence, MIT. USA.

Web pages consulted: URL: http://www.mgap.gub.uy/diea and URL http://www.mgap.gub.uy/opypa

14. Ingeniería de Software

Ing. Alejandro Adorjan

Ingeniería de software: un enfoque en ingeniería de requerimientos

Introducción

La ingeniería de software es una disciplina que estudia la aplicación de la teoría, el conocimiento y la práctica de la construcción eficaz y eficiente de sistemas de software que satisfacen las necesidades de usuarios y clientes.

Un proyecto de ingeniería de software requiere desarrollar algunas competencias como ser: identificar los interesados en un proyecto, determinar sus necesidades, negociar un conjunto de especificaciones, planes y, fundamentalmente, establecer el alcance del proyecto y responder a los distintos cambios en el transcurso del mismo.

El objetivo del curso es brindar los conceptos teóricos y prácticos que permitan comprender y ejecutar los distintos procesos involucrados en el desarrollo de software e implementar una aplicación en el contexto de un proyecto de ingeniería. Si bien el área de ingeniería de software es muy amplia, y las metodologías y herramientas muy variadas, nos enfocaremos en la metodología tradicional.

En el contenido del curso se incluyen los siguientes temas: Introducción a la Ingeniería de Software, Procesos de Software, Gestión de Proyectos, Requerimientos de Software (donde se realiza un enfoque exhaustivo), Pruebas, Gestión de la Calidad (SQA) y Gestión de la Configuración (SCM).

Marco Teórico

A continuación se realizará una síntesis de una de las áreas más relevantes de la ingeniería de software que será tratada en el curso: la ingeniería de requerimientos.

Según el Software Engineering Body of Knowledge (Swebok), el área de conocimientos de requerimientos de software refiere al análisis, la especificación y la validación de los requisitos del software. Sommerville [2] expone que en el proceso de ingeniería de requerimientos se establecen: la obtención, análisis, validación y gestión de requerimientos. A su vez, refiere que la obtención y análisis de requerimientos es un proceso iterativo que puede ser representado como una espiral de actividades (Boehm [1]), las cuales incluyen actividades de elicitación, clasificación, negociación y documentación de requerimientos. Sommerville [2] explica que el proceso de gestión de requerimientos incluye la gestión de la planificación donde se analizan, a su vez, los posibles cambios de requerimientos y su impacto en el proyecto.

Es importante tener en cuenta la problemática de este proceso: el 45% de los errores detectados en los proyectos son por mala o poca especificación. Dichos errores, descubiertos en etapas tardías, son muy costosos (Boehm [1]).

A su vez, ciertos proyectos que se entregan fuera de tiempo con menor calidad son consecuencia de un input insuficiente por parte de los usuarios o por requerimientos incompletos o cambiantes (Standish Group).

Una referencia para la especificación de requerimientos está dada por la Asociación de Estándares de la IEEE, en su documento 830-1998 [5], el cual establece un formato estándar de la especificación de requerimientos. En este documento, conocido como ESRE (Documento de especificación de requerimientos), se deben establecer: el propósito, el alcance del sistema, las definiciones, acrónimos y abreviaciones que correspondan, las referencias y una visión del documento. A su vez, se describe la perspectiva del producto, las funciones del mismo, los requerimientos funcionales y no funcionales. En otra sección del mismo se establecen las características de los usuarios, las restricciones, suposiciones y dependencias del sistema; se documentan, si corresponden, las interfaces externas, los requisitos de rendimiento y las restricciones de diseño.

En el proceso de captura de requerimientos existen distintas fuentes de requerimientos y, a su vez, distintas técnicas de captura de los mismos. Algunas de las características que son deseables en la especificación de un requerimiento son: correctitud, no ambigüedad, completitud, consistencia y verificabilidad.

Es importante recordar que el principal objetivo de la ingeniería de requerimientos es comprender el problema, especificarlo adecuadamente definiendo una solución y validarlo con el cliente.

Ejemplo de Aplicación

A continuación se muestra un bosquejo de la documentación de especificación de requerimientos de software realizada por uno de los grupos del curso, siguiendo el formato sugerido por el estándar IEEE Std 830-1998 [5].

1. Introducción

El presente documento provee una descripción general del producto:

1.1 Propósito

Se pretende orientar el desarrollo de un producto de software…

1.2 Alcance

El desarrollo del producto permitirá registrar…

1.3 Definiciones, Acrónimos y Abreviaciones

Insumo: Conjunto de bienes empleados en…

1.4 Referencias

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

2. Descripción General

2.1 Actores

Administrativo: Es la persona encargada de realizar los registros.

Director: Es el usuario que tendrá el perfil para obtener los informes brindados por el sistema.

2.2 Funciones del producto

Las principales funciones del producto son:

Administración de usuarios.

Registro de alumnos.

Registro de alumnos para el servicio…

Registro de…

2.1.1 Requerimientos funcionales

RF1 Administración de Usuarios.

Descripción: El administrador del sistema podrá gestionar los usuarios (agregar, modificar, eliminar, buscar, listar).

Especificación: Caso de uso 3.1.

Prioridad: 1

RF2: Registro de Alumnos.

Descripción: El sistema deberá registrar los alumnos, ingresando los datos correspondientes a los mismos.

Especificación: Caso de uso 3.2

Prioridad: 1

2.1.2 Requerimientos no funcionales

RNF1 El sistema deberá ser codificado en… versión…

Descripción: El sistema deberá codificarse según la versión…

Prioridad: 1

3. Especificación de Casos de Uso

Caso de Uso 3.3: Registro de inscripción en el servicio

Actores: Usuario.

Precondición: El usuario debe estar registrado…

Sinopsis: El caso uso comienza cuando…

Referencia: RF3.

Caso de Uso	**3.3**
Nombre	Registro de inscripción en el servicio.
Descripción	Descripción del requerimiento funcional.
Prioridad	1
Estado	Codificado.
Actores	Usuario.
Precondiciones	El usuario deberá...
Entradas	
Flujo de Eventos	
Curso Normal	1. El usuario selecciona opción cambiar contraseña. 2. El sistema muestra en pantalla el ingreso de la actual contraseña y la nueva. 3. El usuario ingresa contraseña actual y nueva. 4. El sistema…
Curso Alternativo	3.1 Si la contraseña actual no coincide, muestra un mensaje de error. *Se enumera la acción / acciones que aplican al no cumplirse el curso normal.*
Post Condiciones	*Lista el estado posterior a la ejecución del caso de uso.*
Salidas	
Restricciones	*Lista de las restricciones del sistema, generalmente asociadas a los requerimientos no funcionales.*

Casos de uso relacionados	
Interfaz de Usuario	*Es una buena práctica establecer la interfaz de usuario asociada al caso de uso.*

4. Contexto de presentación del curso

El curso se dicta en modalidad semipresencial con tres instancias presenciales y actividades semanales orientadas al estudio guiado sobre las temáticas del curso en la plataforma educativa de UTU, a través del Campus Virtual.

Los libros de Sommerville [2] y Pressman [3] son parte de la bibliografía sugerida del curso. Si bien el enfoque es tradicional se recomiendan artículos y libros de las metodologías ágiles. Se realizan cuestionarios de los distintos temas planteados y se propone un trabajo final donde el tema es de libre elección de los alumnos, planteándose el desafío de la resolución de un producto de software y la gestión del mismo desde el análisis hasta su implementación.

5. Algunas preguntas planteadas a los participantes

A continuación se exponen algunas de las preguntas que pueden plantearse a los participantes del curso en los cuestionarios de evaluación:

¿Cuál fue la crisis del software?
¿Cuáles son los dos tipos fundamentales de producto de software?
¿Qué es la ingeniería de software?
¿Cuáles son las actividades fundamentales en los procesos de software?
¿Cuáles son los tres paradigmas en el desarrollo de software (en estas metodologías tradicionales)?
¿Por qué es importante el mantenimiento de software?
¿Cuáles son las actividades fundamentales que son comunes a todos los procesos de software?
¿Cuáles son las ventajas de utilizar el desarrollo incremental?
¿Cuáles son las principales actividades de ingeniería de requerimientos?
¿Cuáles son las etapas importantes en el proceso de pruebas?
¿Cuáles son las principales diferencias entre la gestión de proyectos de software y otros tipos de gestión de proyectos?
¿Cuál es la diferencia entre un hito y una entrega?
¿Qué se incluye en un plan de calidad y en un plan de validación?
¿Qué son los requisitos de un sistema?
¿Qué son los requisitos del usuario y los requisitos del sistema?
¿Cuál es la diferencia entre los requerimientos funcionales y los no funcionales?
¿Qué problemas pueden surgir cuando los requisitos están escritos en lenguaje natural?
¿Cuáles son las principales ventajas de utilizar un formato estándar para especificar los requerimientos?
¿Qué es un documento de especificación de requerimientos?
¿Qué es un caso de uso?
¿Cuáles son los dos objetivos complementarios del proceso de pruebas?
¿Qué herramientas de gestión de la configuración conoce?
¿Cuándo una prueba de defectos es exitosa?
¿Qué tipos de pruebas conoce?
¿Qué enfoques pueden ser utilizados en el diseño de casos de prueba?
¿Qué es una partición de equivalencia?
¿Qué se entiende por gestión de la configuración?

6. Conclusión

En este curso se presentan algunos de los temas más relevantes del área de ingeniería de software a partir de una especificación de requerimientos formal. Se enfatizan los conceptos teóricos y prácticos que permiten comprender y ejecutar los distintos procesos involucrados en el desarrollo de software.

7. Bibliografía

Boehm B, (1986). "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", "ACM", 11(4):14-24,

Sommerville, I. (2010). Software Engineering, (9th ed.) Addison-Wesley.

Pressman, Roger. (2010). Software Engineering: A Practitioner's Approach, (7th ed.). NY: McGraw-Hill.

Kotonya Gerald, Sommerville Ian. Requirements Engineering Processes and Techniques. John Wiley & Sons Ltd. ISBN 0-471-97208-8

IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications –Description.

8. Referencias Web

Se aconseja, para ampliar los conocimientos sobre este tema, visitar el sitio web de Ian Sommerville, autor de uno de los libros de referencia del curso:

http://www.softwareengineering-9.com/

Recomendamos, asimismo, el estudio del mismo autor para el tema Enfoque de requerimientos:

Procesos y Técnicas de requerimientos:
www.comp.lancs.ac.uk/computing/resources/re/
http://www.comp.lancs.ac.uk/computing/resources/re-gpg/