# ARQUITECTURA Y DISEÑO DE SISTEMAS

# ESTILOS Y PATRONES DE ARQUITECTURA – PARTE 1

**ELSA ESTEVEZ**
UNIVERSIDAD NACIONAL DEL SUR
DEPARTAMENTO DE CIENCIAS E INGENIERIA DE LA COMPUTACION

# CONTENIDO

# GENERALIDADES

## ARQUITECTURA DE SOFTWARE ESTUDIA

- Cómo un sistema se puede descomponer de la mejor forma posible?
- Cuáles son sus componentes?
- Cómo se comunican dichos componentes?
- Cómo fluye la información?
- Cómo los elementos de un sistema pueden evolucionar independientemente?
- Cómo se puede describir todo esto a través de notaciones formales e informales?

# COMO SURGE

A medida que los ingenieros de software fueron construyendo diferentes sistemas a lo largo de muchos dominios de aplicación, observaron que, bajo determinadas circunstancias, ciertas decisiones de diseño regularmente resultan en soluciones con propiedades superiores: más elegantes, efectivas, eficientes, confiables, evolucionables y escalables.

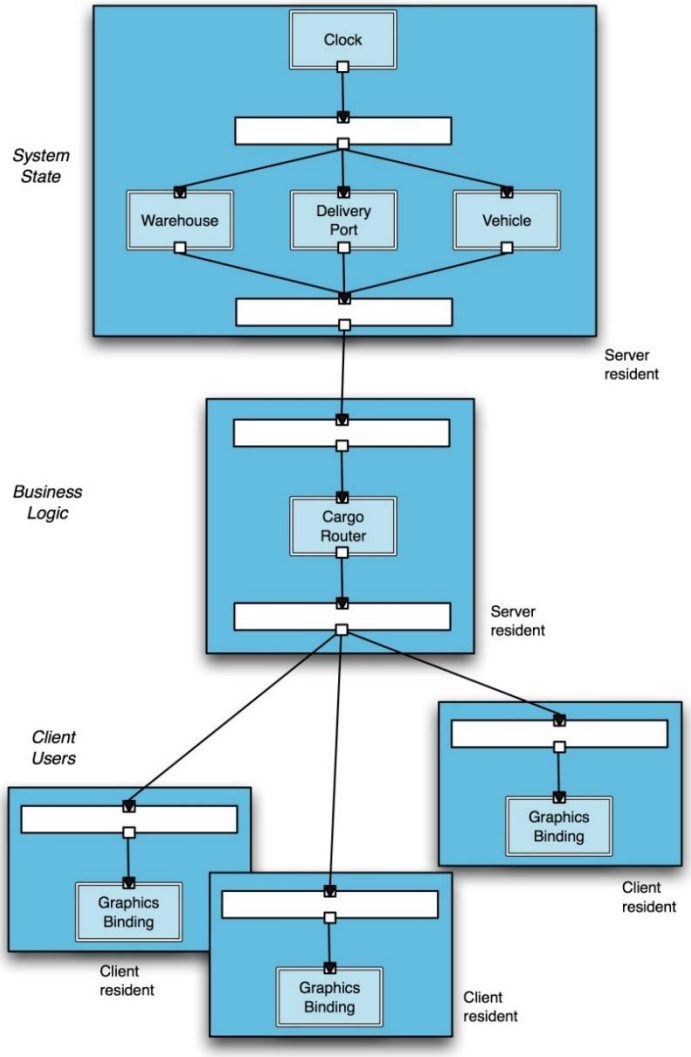
Esta experiencia y conocimiento común se generalizó y fue adoptando distintas formas dependiendo del contexto de aplicación con el fin de ser reutilizado: arquitecturas de software específicas de un dominio, estilos arquitectónicos y patrones arquitectónicos.

# ELEMENTOS DE LA ARQUITECTURA

La arquitectura de software de un sistema por lo general no es (y no debería ser) monolítica y uniforme.

Por el contrario debe resultar de la composición e interacción de diferentes elementos:

- Procesamiento
- Datos, también conocida como información o estado
- Interacción entre los anteriores



# COMPONENTES

En la arquitectura de un sistema se referencia como componente de software a los elementos que encapsulan procesamiento y datos

## DEFINICIÓN

Un componente de software es una entidad arquitectónica que:

1) encapsula un subconjunto de funcionalidad y / o datos del sistema,
2) restringe el acceso a ese subconjunto a una interfaz definida explícitamente
3) tiene definidas dependencias explícitas sobre el contexto de ejecución requerido

Los componentes suelen proporcionar los servicios específicos de la aplicación.

# EJEMPLOS DE COMPONENTES

## Componentes específicos de la aplicación

- Ejemplos: bancarias, almacenes, vehículos

## Componentes de reutilización limitada

- Ejemplos: servidores web, los relojes, las conexiones

## Componentes reutilizables

- Ejemplos: bibliotecas de componentes GUI, librerías matemáticas

# CONECTORES

En sistemas muy complejos, la interacción puede llegar a ser más difícil e importante que la funcionalidad de los componentes

## DEFINICIÓN

Un conector es una pieza de la arquitectura que se ocupa de llevar adelante y regular interacciones entre los componentes.

- En muchos sistemas, los conectores son por llamadas a procedimientos o accesos a datos compartidos
- Los conectores típicamente proporcionan facilidades de interacción que son independientes de las aplicación
- Se pueden describir independiente de los componentes.

# EJEMPLO DE CONECTORES

- Llamadas a procedimientos
- Memoria compartida
- Pasaje de mensajes
- Streaming
- Conectores de distribución (distribution connectors) – encapsulan “application programming interfaces (API’s)” que habilitan la interacción de componentes distribuidos
- Conectores de adaptación (Wrapper/adaptor connectors)

# CONTENIDO

1 CONCEPTOS Y ELEMENTOS

2 ESTILOS Y PATRONES

3 CLASIFICACIÓN - ESTILOS SIMPLES

- ESTILOS INFLUENCIADOS POR LENGUAJES DE PROGRAMACIÓN
- ESTILOS POR CAPAS
- ESTILO DE FLUJO DE DATOS

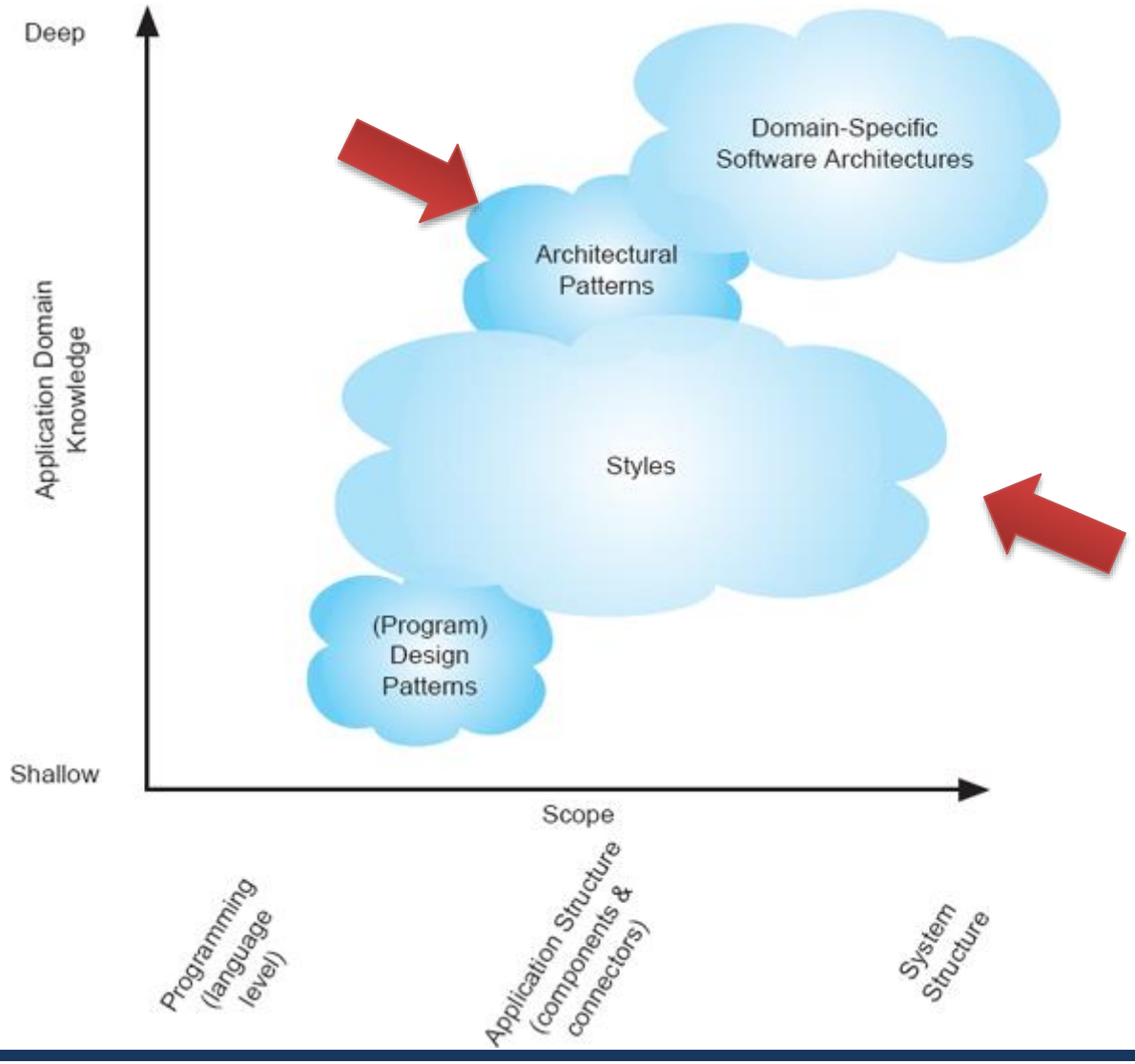
# ESTILOS Y PATRONES ARQUITECTONICOS

Un estilo arquitectónico es un colección de decisiones de diseño arquitectónico que:

1. son aplicables a un contexto de desarrollo,
2. dado un sistema particular, restringe las decisiones de diseño de arquitectura sobre dicho sistema, y
3. garantiza ciertas calidades del sistema resultante

Un patrón arquitectónico es una colección de decisiones de diseño arquitectónico que son aplicables a *problemas de diseño recurrentes*, y que están parametrizados para tener en cuenta los diferentes contextos de desarrollo de software en el que surge el problema.

# DEFINICIONES




- Límites difusos
- Guía de alto nivel
- El eje del “alcance” no es tan lineal como parece
- Lo que un arquitecto puede llamar “estilo arquitectónico”, otro puede llamarlo “patrón arquitectónico”

# PATRONES DE DISEÑO

- La comunidad OO ha estado explorando el uso de patrones de diseño para describir abstracciones recurrentes en el desarrollo de software basado en objetos
- Generalmente en el marco de la Orientación a Objetos
- Se concentran en soluciones de diseño comunes aplicables a nivel de clases, procedimientos y estructuras de datos.
- Expresan esquemas para definir estructuras de diseño y sus relaciones a partir de los cuales se puede construir un componente de software.
- Catálogos más conocidos:
  - ✓ GoF: Design Patterns: Elements of Reusable Object-Oriented Software (Gamma, Helm, Johnson, Vlissides)
  - ✓ GRASP: General Responsibility Assignment Software Patterns (Craig Larman)

# ARQUITECTURA DE SOFTWARE DE DOMINIO ESPECIFICO (DSSA)

- Reúne conocimiento sustancial, adquirido a partir de una extensa experiencia, sobre cómo estructurar aplicaciones completas dentro de un dominio particular.
- Desde el punto de vista operacional, podría definirse como la combinación de:
  - ✓ La arquitectura de referencia para un dominio de aplicación
  - ✓ Una librería de componentes de software para dicha arquitectura, conteniendo componentes reusables basados en la experiencia en el dominio
  - ✓ Un método para elegir y configurar los componentes para que trabajen dentro de una instancia de la arquitectura de referencia.
- Están íntimamente relacionadas con las arquitecturas para una familia de productos (recordar el caso de estudio de los TV Philipps)
- Están especializadas para un dominio particular y sólo son valiosas para aplicaciones dentro de dicho dominio

# ESTILOS ARQUITECTÓNICOS

## DEFINICIÓN

Un estilo arquitectónico es una colección de decisiones de diseño arquitectónicas que tiene un nombre específico y que:

- es aplicable a un contexto de desarrollo dado
- restringe decisiones de diseño arquitectónicas que son específicas a un sistema particular dentro de aquel contexto
- elicita cualidades beneficiosas en cada sistema resultante

Taylor, Medvidovic and Dashofy

- caracterizan lecciones aprendidas en el diseño de sistemas de software
- proveen decisiones de diseño generales que restringen y pueden necesitar ser refinadas en decisiones de diseño adicionales y más específicas para que sean aplicadas en un sistema.

# ESTILOS ARQUITECTÓNICOS - PROPIEDADES

Un vocabulario para los elementos de diseño

- Tipos de componentes y conectores
- Por ejemplo: clases, invocaciones, “pipes”, clientes, etc.

Reglas de composición

- Un estilo tiene restricciones topológicas que determinan cómo se puede hacer la composición de los elementos

  Por ejemplo: los elementos de un “layer” se pueden comunicarse sólo con los del “layer” inferior

Semántica para los elementos - elementos con significado bien conocido

# ESTILOS ARQUITECTÓNICOS - BENEFICIOS

- reúso de diseños
- soluciones maduras aplicadas a problemas nuevos
- una parte importante del código que implementa la arquitectura puede pasarse de un sistema a otro
- comunicación más efectiva
- portabilidad de soluciones

# ESTILOS ARQUITECTÓNICOS – EJEMPLO CLIENTE-SERVIDOR

El siguiente conjunto de decisiones de diseño asegura la provisión efectiva de servicio a múltiples usuarios del sistema en un ambiente distribuido

Separar físicamente los componentes de software usados para requerir servicios de aquellos usados para proveer servicios, con el objetivo de proveer una distribución y escalamiento apropiados (tanto en la cantidad de proveedores de servicio como de solicitantes de servicios)

Hacer que los proveedores no tengan conocimiento sobre la identidad de los solicitantes para permitir que los sirvan transparentemente a muchos (posiblemente cambiantes) solicitantes

Aislar a los solicitantes unos de otros para permitir que sean agregados o eliminados independientemente. Hacer que los solicitantes sólo dependan de los proveedores de servicio

Permitir que múltiples proveedores de servicios se creen de forma dinámica para sacar carga a los proveedores existentes si la demanda de servicios aumenta por encima de un determinado umbral

# PATRONES ARQUITECTÓNICOS

## DEFINICIÓN

Un patrón arquitectónico es una colección de decisiones de diseño arquitectónicas que tiene un nombre específico y que son aplicables a problemas de diseño recurrentes, y son parametrizadas para tener en cuenta diferentes contextos de desarrollo de software en los cuales el problema aparece.

Taylor, Medvidovic and Dashofy

- provee un conjunto de decisiones específicas de diseño que han sido identificadas como efectivas para organizar ciertas clases de sistemas de software o, más típicamente, subsistemas específicos.
- estas decisiones de diseño pueden pensarse como “configurables”, ya que necesitan ser instanciadas con los componentes y conectores particulares a una aplicación.

# ESTILOS Y PATRONES - DIFERENCIAS

| | ESTILO | PATRÓN |
|---|---|---|
| Alcance | Aplican a un contexto de desarrollo:<br>○ “sistemas altamente distribuidos”,<br>○ “sistemas intensivos en GUI” | Aplican a problemas de diseño específicos:<br>○ El estado del sistema debe presentarse de múltiples formas<br>○ La lógica de negocio debe estar separada del acceso a datos |
| Abstracción | Son muy abstractos para producir un diseño concreto del sistema. | Son fragmentos arquitectónicos parametrizados que pueden ser pensados como una pieza concreta de diseño. |
| Relación | Un sistema diseñado de acuerdo a las reglas de un único estilo puede involucrar el uso de múltiples patrones | Un único patrón puede ser aplicado a sistemas diseñados de acuerdo a los lineamientos de múltiples estilos |

# CONTENIDO

1 CONCEPTOS Y ELEMENTOS

2 ESTILOS Y PATRONES

**3 CLASIFICACIóN - ESTILOS SIMPLES**

- ESTILOS INFLUENCIADOS POR LENGUAJES DE PROGRAMACIóN
- ESTILOS POR CAPAS
- ESTILO DE FLUJO DE DATOS

# UNA PROPUESTA

## ESTILOS SIMPLES

**Estilos influenciados por lenguajes tradicionales**
- Programa principal y subrutinas
- Orientación a objetos

**Por capas**
- Maquinas virtuales
- Cliente-servidor

**Estilos de Flujo de Datos**
- Batch sequencial
- Pipe and Filters

**Memoria compartida**
- Blackboard
- Basado en reglas

**Intérprete**
- Intepreter
- Mobile Code

**Invocación Implícita**
- Publish-Subscriber
- Event Based

**Peer to Peer**

## ESTILOS COMPLEJOS

- C2
- Objetos distribuidos

Ref: *Software Architecture, Foundations Theory and Practice*

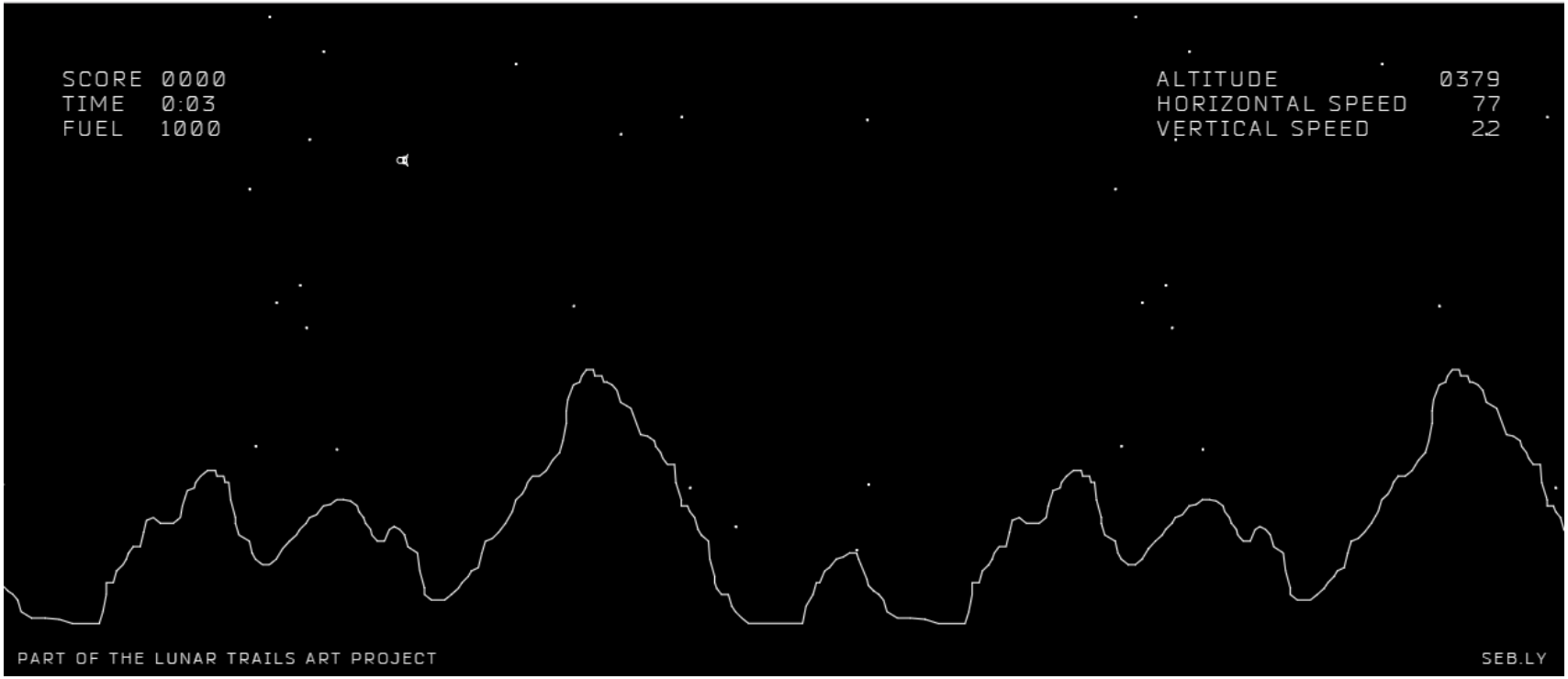
# EJEMPLO: LUNAR LANDER

Lunar Lander es un videojuego desarrollado por Atari, Inc. en 1979. El juego trata sobre pilotear un módulo lunar y alunizarlo de manera segura en la luna.

Fecha de lanzamiento al mercado: agosto de 1979

Géneros: Videojuego de simulación de vehículos

**Plataformas:** Arcade, Microsoft Windows, Xbox 360, Windows Phone”



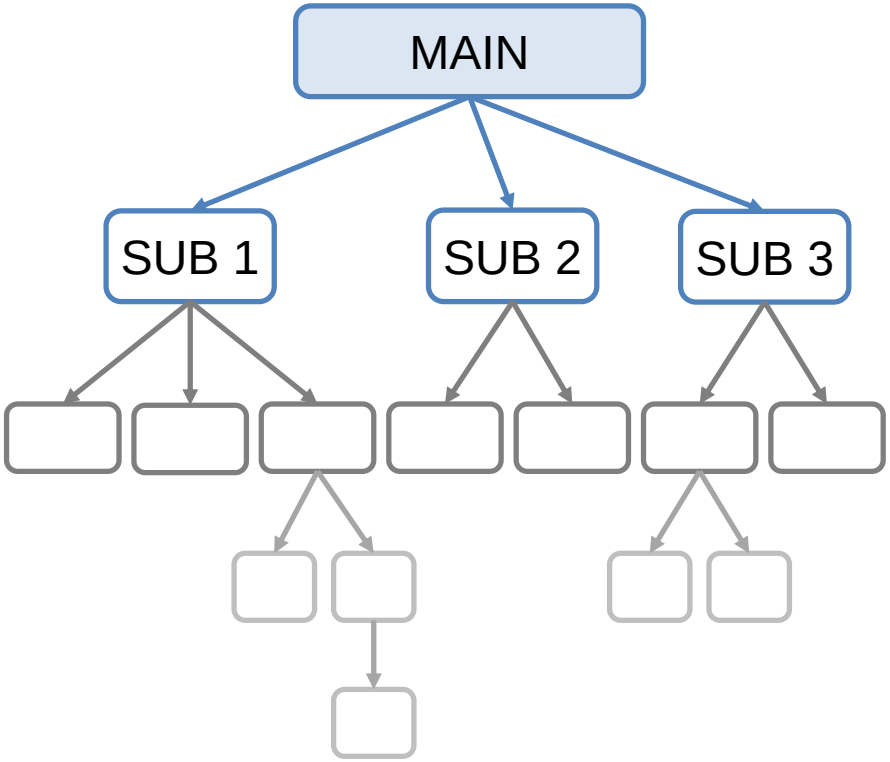
# LUNAR LANDER – DESCRIPCIÓN

- Juego popular en los años 70
- Referencia: http://moonlander.seb.ly/
- Reglas
  - ✓ El piloto controla el descenso de un lunar lander estilo Apollo
  - ✓ El piloto controla la aceleración del descenso
  - ✓ El combustible es limitado
  - ✓ La altitud inicial y la velocidad están predefinidas
  - ✓ Si aterriza con una velocidad de descenso < 5, gana aunque no tenga combustible

# ESTILOS INFLUENCIADOS POR EL LENGUAJE

Programa principal y subrutinas

- Influenciado por la Programación Estructurada
- Envío de mensaje / respuesta
- Se cede el control y se espera la respuesta

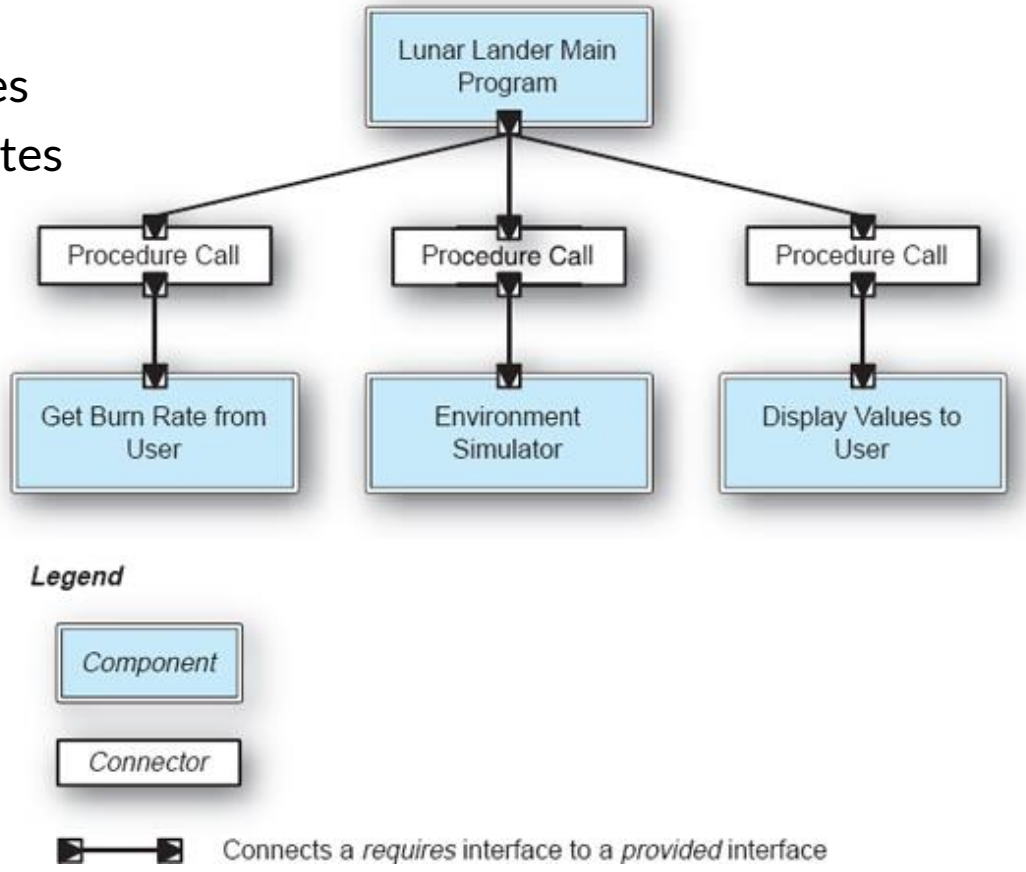


# 1-PROGRAMA PRINCIPAL Y SUBRUTINAS (PPS) - ANÁLISIS

| | |
|---|---|
| DESCRIPCIÓN | Descomposición basada en la separación de pasos de procesamiento funcionales |
| COMPONENTES | Programa principal y subrutinas |
| CONECTORES | Llamadas a procedimientos / funciones |
| ELEMENTOS DE DATOS | Valores de entrada/salida de las subrutinas |
| TOPOLOGÍA | La organización estática de los componentes es jerárquica.<br>La estructura es un grafo dirigido |
| RESTRICCIONES ADICIONALES | Ninguna |
| CUALIDADES | Modularidad: Las subrutinas pueden ser reemplazadas con diferentes implementación mientras su interfaz no sea afectada |
| USOS TÍPICOS | Pequeños programas. Uso pedagógico. |
| PRECAUCIONES | ▪ Generalmente falla al escalar a grandes aplicaciones<br>▪ Inadecuada atención a estructuras de datos<br>▪ Requiere bastante esfuerzo para introducir nuevos requerimientos: “impredecible”<br>▪ Reuso limitado de funciones y procedimientos |

# DISEÑO

- Diseño basado en descomposición funcional
- Programa principal
  - ✓ Muestra saludos e instrucciones
  - ✓ Cicla llamando a los componentes



# 2- ORIENTACION A OBJETOS

Promueve el uso de las principales características de la OO

- Ocultamiento de información
- Encapsulamiento
- Herencia / Polimorfismo

**Ocultamiento de Información -** Las decisiones de diseño que es probable que cambien, son ocultadas en un módulo o conjunto de pequeños módulos

*Parnas, 1972*

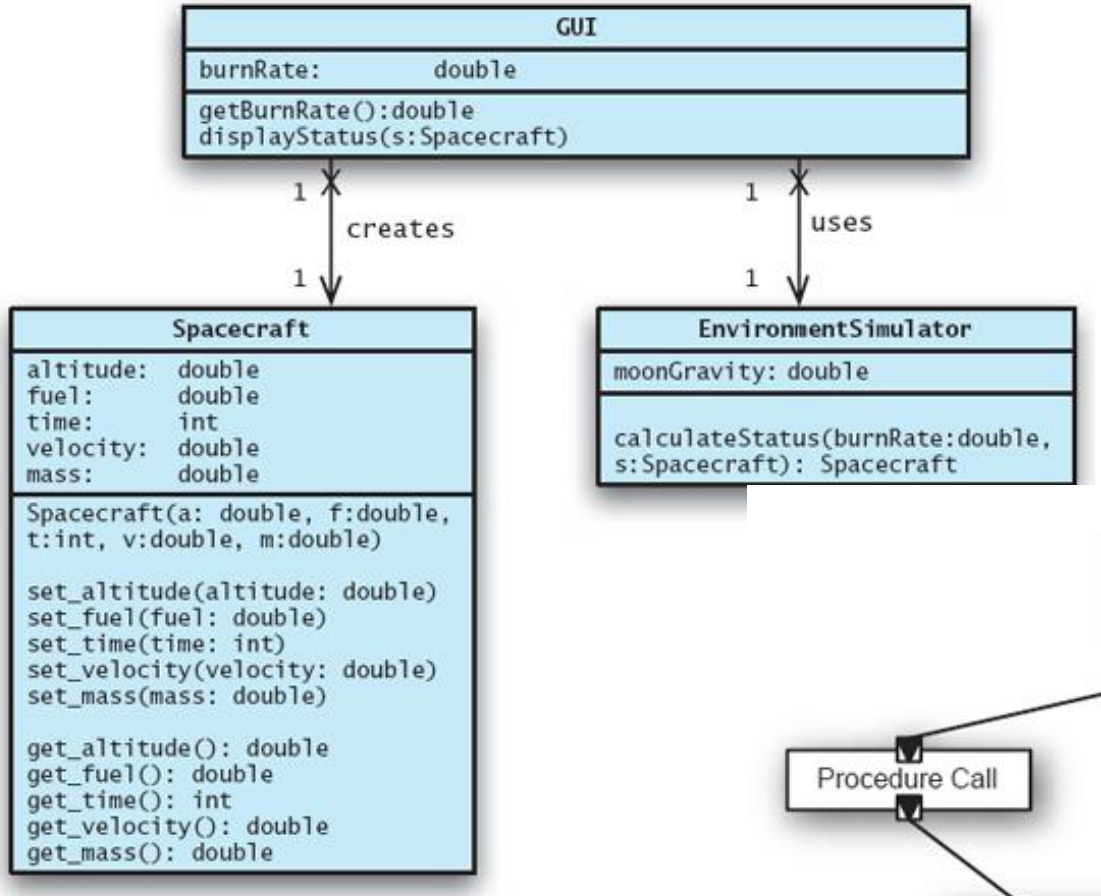
# ORIENTACIÓN A OBJETOS – ANÁLISIS 1

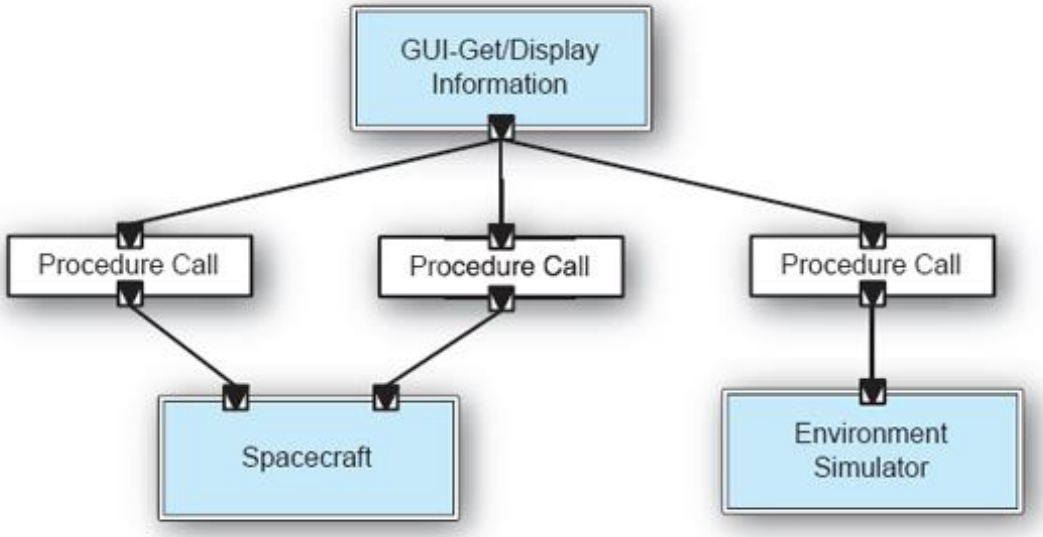
| | |
|---|---|
| DESCRIPCIÓN | Estado y funciones (sobre dicho estado) encapsulados en objetos. Los objetos deben ser instanciados antes que sus métodos sean invocados. |
| COMPONENTES | Objetos (datos y operaciones asociadas) |
| CONECTORES | Invocación a métodos (llamadas a procedimientos para manipular el estado) |
| ELEMENTOS DE DATOS | Parámetros de los métodos |
| TOPOLOGÍA | Puede variar. Los componentes pueden “compartir” datos y funciones a través de jerarquías de herencia |
| RESTRICCIONES ADICIONALES | ▪ Generalmente, memoria compartida (para soportar el uso de punteros).<br>▪ Único thread (por defecto)<br>▪ Conocer la identidad de los objetos |

# ORIENTACIÓN A OBJETOS – ANÁLISIS 2

| | |
|---|---|
| CUALIDADES | ▪ Integridad de las operaciones sobre los datos (están co-localizadas)<br>▪ Abstracción: Ocultamiento de los detalles de implementación<br>▪ Reuso a gran escala<br>▪ Correspondencia con objetos del dominio |
| USOS TÍPICOS | ▪ Correlación entre las entidades en el mundo real y las entidades en la app.<br>▪ Apps que tienen estructuras de datos complejas y dinámicas. |
| PRECAUCIONES | ▪ El uso en aplicaciones distribuidas requiere acceso remoto a los objetos.<br>▪ Relativa ineficiencia para apps de alta performance<br>▪ Carencia de principios de diseño resulta en apps altamente complejas.<br>▪ No todo es un objeto |

# DISEÑO OO


GUI
burnRate: double
getBurnRate():double
displayStatus(s:Spacecraft)
1
creates
1
1
uses
1
Spacecraft
altitude: double
fuel: double
time: int
velocity: double
mass: double
Spacecraft(a: double, f:double, t:int, v:double, m:double)
set_altitude(altitude: double)
set_fuel(fuel: double)
set_time(time: int)
set_velocity(velocity: double)
set_mass(mass: double)
get_altitude(): double
get_fuel(): double
get_time(): int
get_velocity(): double
get_mass(): double
EnvironmentSimulator
moonGravity: double
calculateStatus(burnRate:double, s:Spacecraft): Spacecraft


GUI-Get/Display Information
Procedure Call
Procedure Call
Procedure Call
Spacecraft
Environment Simulator

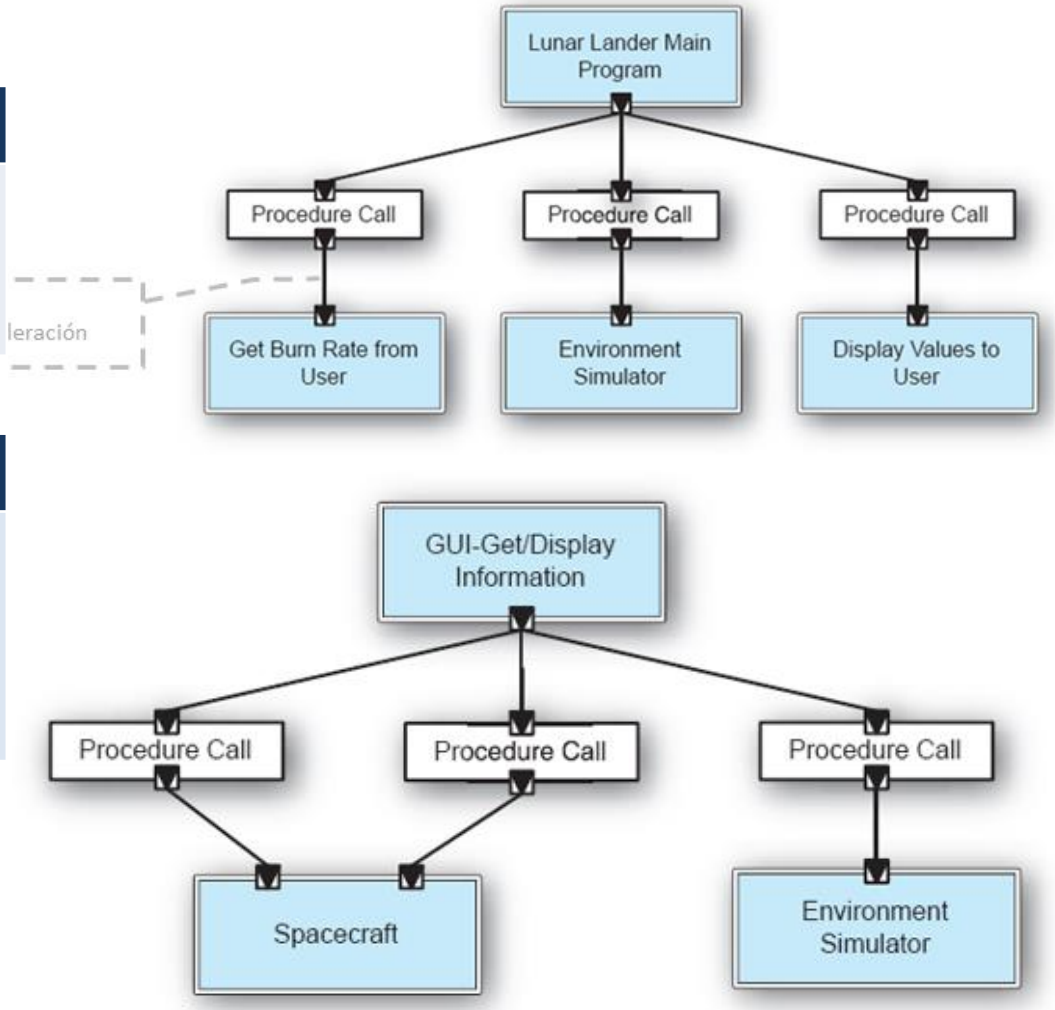
# COMPARACIÓN DE DISEÑOS

## SIMILITUDES

- Conectores - llamadas a procedimientos
- Datos – argumentos

## DIFERENCIAS

- Encapsulamiento (UI, Spacecraft, Environment Simulator)
- PPS: input y output separadas
- OO: input y output juntas



# UNA PROPUESTA - REVISIÓN

## ESTILOS SIMPLES

Estilos influenciados por lenguajes tradicionales
- Programa principal y subrutinas
- Orientación a objetos

✓

Por capas
- Maquinas virtuales
- Cliente-servidor

Estilos de Flujo de Datos
- Batch sequencial
- Pipe and Filters

Memoria compartida
- Blackboard
- Basado en reglas

Intérprete
- Intepreter
- Mobile Code

Invocación Implícita
- Publish-Subscriber
- Event Based

Peer to Peer

## ESTILOS COMPLEJOS

- C2
- Objetos distribuidos

Ref: *Software Architecture, Foundations Theory and Practice*

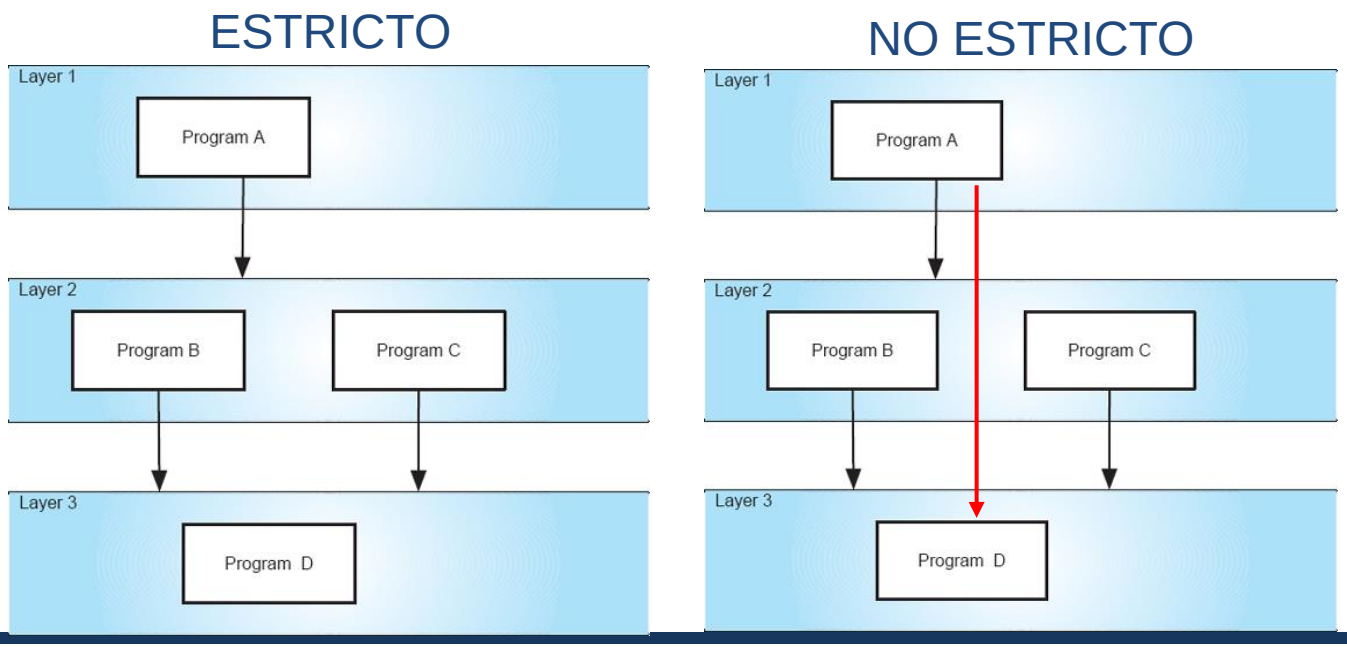
# ESTILO DE CAPAS

- Considera la arquitectura como un sistema en capas ordenadas. Un programa ubicado en un nivel o capa puede obtener los servicios de un nivel inferior.
- Es un estilo simple y de uso clásico para desarrollos en lenguajes de programación como Java o C.

- Las variedades del estilo capas incluyen:
  - Maquinas virtuales: de amplio uso en arquitecturas de computadoras y sistemas operativos.
  - Cliente servidor: presente principalmente en aplicaciones de negocio

# 3 - MÁQUINA VIRTUAL

- Una capa ofrece un conjunto de servicios (“interface” o API). Dichos servicios pueden ser accedidos por programas que residen en la capa superior.
- Encapsula la implementación de los servicios de la capa
- Dos modos
  - ✓ Estricto
  - ✓ No estricto

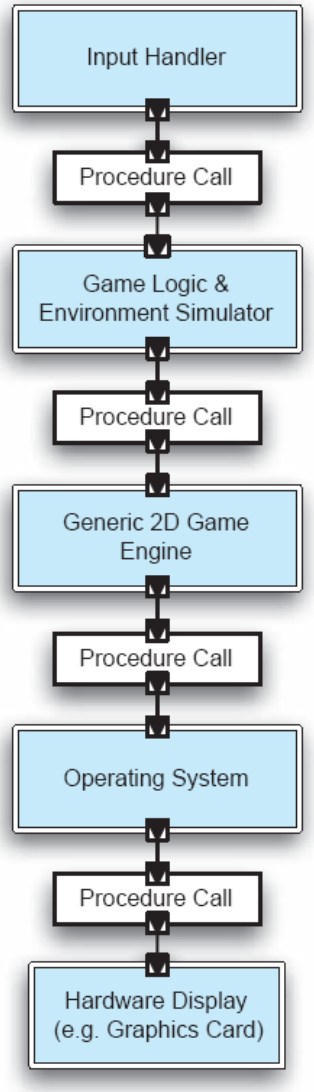


# MÁQUINA VIRTUAL - ANÁLISIS

| | |
|---|---|
| DESCRIPCIÓN | ▪ Secuencia ordenada de capas. Cada capa (virtual machine) ofrece servicios a subcomponentes de la capa encima de ella. |
| COMPONENTES | ▪ Capas, generalmente conteniendo subcomponentes |
| CONECTORES | ▪ Llamadas a procedimientos |
| ELEMENTOS DE DATOS | ▪ Parámetros pasados entre las capas |
| TOPOLOGÍA | ▪ Lineal |
| RESTRICCIONES ADICIONALES | ▪ Ninguna |
| CUALIDADES | ▪ Dependencias claras y acotadas<br>▪ Encapsulamiento<br>▪ Basado en niveles de abstracción<br>▪ Reusabilidad, Portabilidad |
| USOS TÍPICOS | ▪ Sistemas Operativos<br>▪ Stacks de protocolos de red<br>▪ Aplicaciones empresariales |
| PRECAUCIONES | ▪ Performance<br>▪ No siempre se puede encontrar la abstracción correcta<br>▪ Una maquina virtual con varios niveles puede resultar ineficiente |

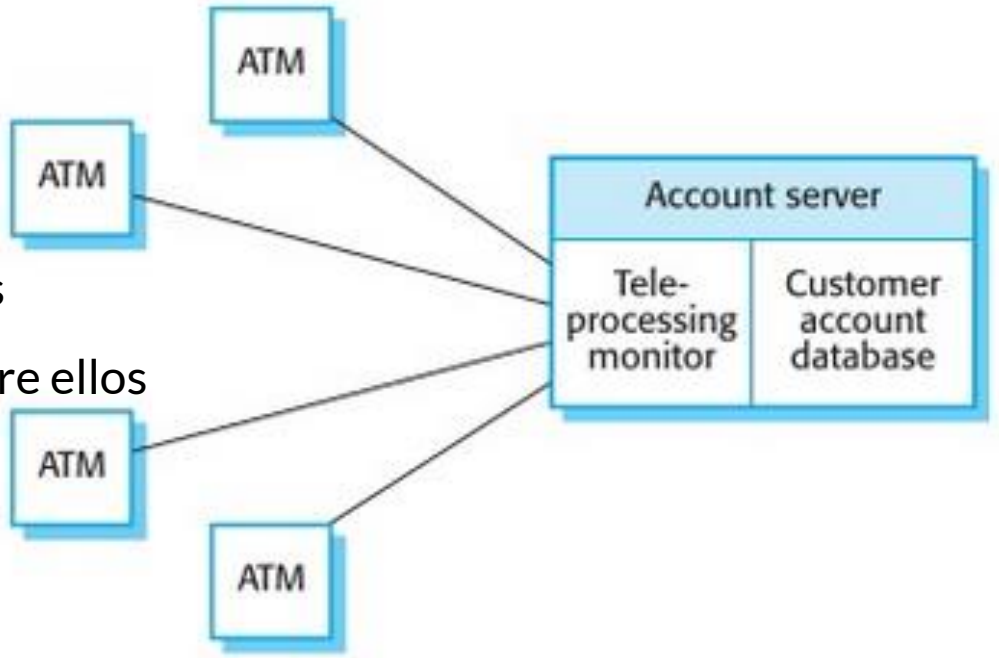
# MÁQUINA VIRTUAL - DISEÑO

| CAPA | FUNCIONALIDAD |
|---|---|
| 1 | Recibe la entrada del usuario |
| 2 | Implementa el simulador del ambiente y la lógica del juego |
| 3 | Motor de juegos de gráficos en dos dimensiones genérico. Capaz para ser usada en cualquier juego 2D |
| 4 | Sistema Operativos proveyendo soporte de GUI |
| 5 | Firmware o hardware |



# 4 - CLIENTE SERVIDOR

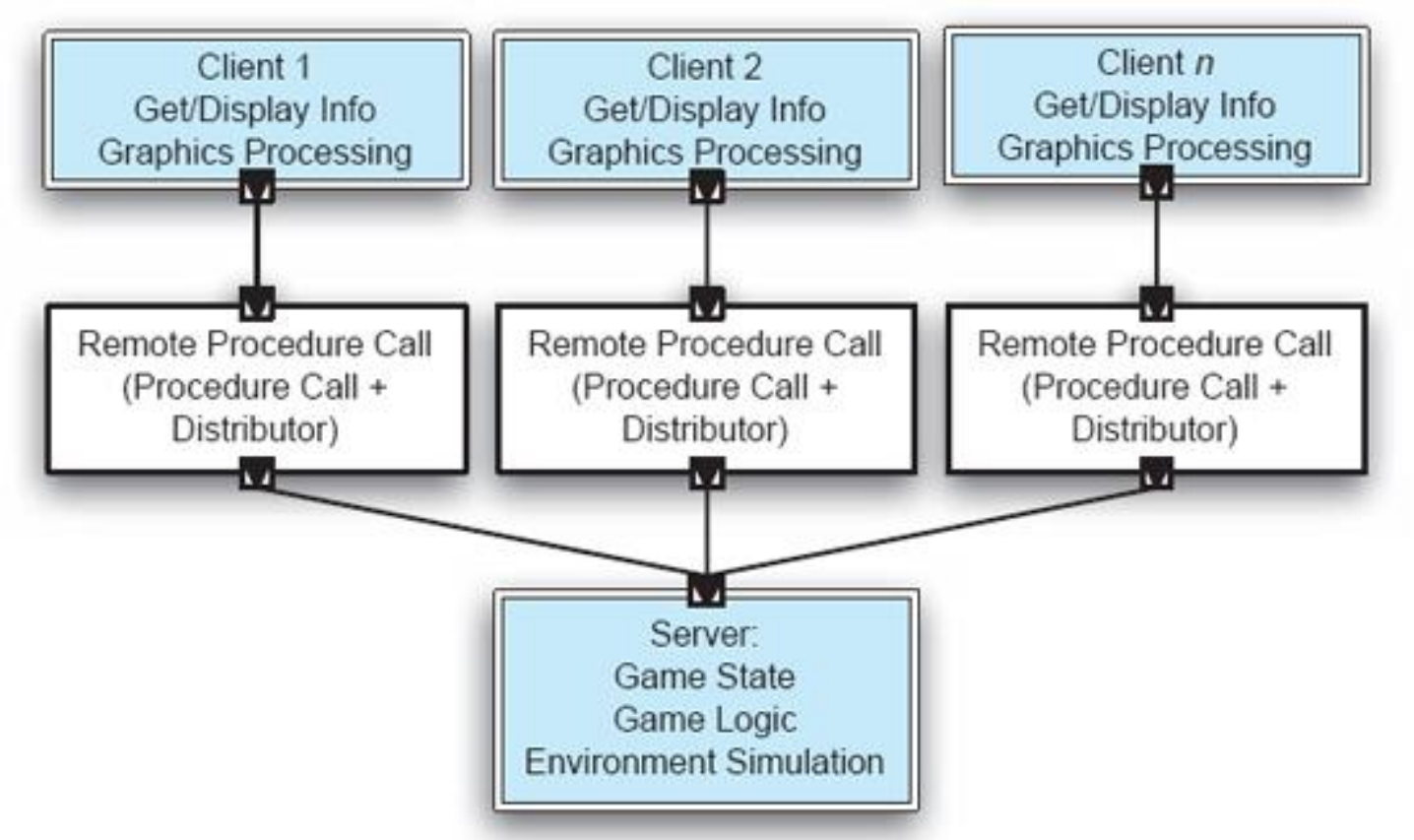
- Múltiples clientes independientes accediendo al mismo server
- Especialización de máquina virtual de 2 capas:
  - Capa 1: Cliente
  - Capa 2: Servidor
  - Conexión por red
- Clientes conocen al Servidor
- El Servidor no conoce a los Clientes
- Los Clientes son independiente entre ellos



# CLIENTE SERVIDOR - ANÁLISIS

| | |
|---|---|
| DESCRIPCIÓN | Clientes le envían requerimientos al servidor, el cual los ejecuta y envía la respuesta (de ser necesario). La comunicación es iniciada por el cliente. |
| COMPONENTES | Clientes y Servidor |
| CONECTORES | Llamadas a procedimientos remotos (o equivalente) |
| ELEMENTOS DE DATOS | Parámetros y valores de retorno |
| TOPOLOGÍA | Dos niveles. Múltiples clientes haciendo requerimientos al server |
| RESTRICCIONES ADICIONALES | Prohibida la comunicación entre clientes |
| CUALIDADES | ▪ Sencilla y muy utilizada<br>▪ Centralización de cómputos y datos en el server.<br>▪ Mantenible<br>▪ Un único servidor puede atenderá a múltiples clientes |
| USOS TÍPICOS | ▪ Apps con datos y/ o procesamiento centralizados y clientes GUI<br>▪ Stacks de protocolos de red<br>▪ Aplicaciones empresariales |
| PRECAUCIONES | ▪ Condiciones de la red vs crecimiento de clientes. |

# DISEÑO MULTIJUGADOR




- El estado del juego, la lógica y la simulación del entorno se realiza en el servidor.
- Los clientes realizan las funciones de interface de usuario.
- Conectores: RPC: llamada a procedimientos y un “Distributor” que identifica los caminos de interacción de la red y las rutas de comunicación

# UNA PROPUESTA - REVISIÓN

## ESTILOS SIMPLES

Estilos influenciados por lenguajes tradicionales
- Programa principal y subrutinas
- Orientación a objetos

✓

Por capas
- Maquinas virtuales
- Cliente-servidor

✓

Estilos de Flujo de Datos
- Batch sequencial
- Pipe and Filters

Memoria compartida
- Blackboard
- Basado en reglas

Intérprete
- Intepreter
- Mobile Code

Invocación Implícita
- Publish-Subscriber
- Event Based

Peer to Peer

## ESTILOS COMPLEJOS

- C2
- Objetos distribuidos

Ref: *Software Architecture, Foundations Theory and Practice*

# ESTILOS DE FLUJOS DE DATOS

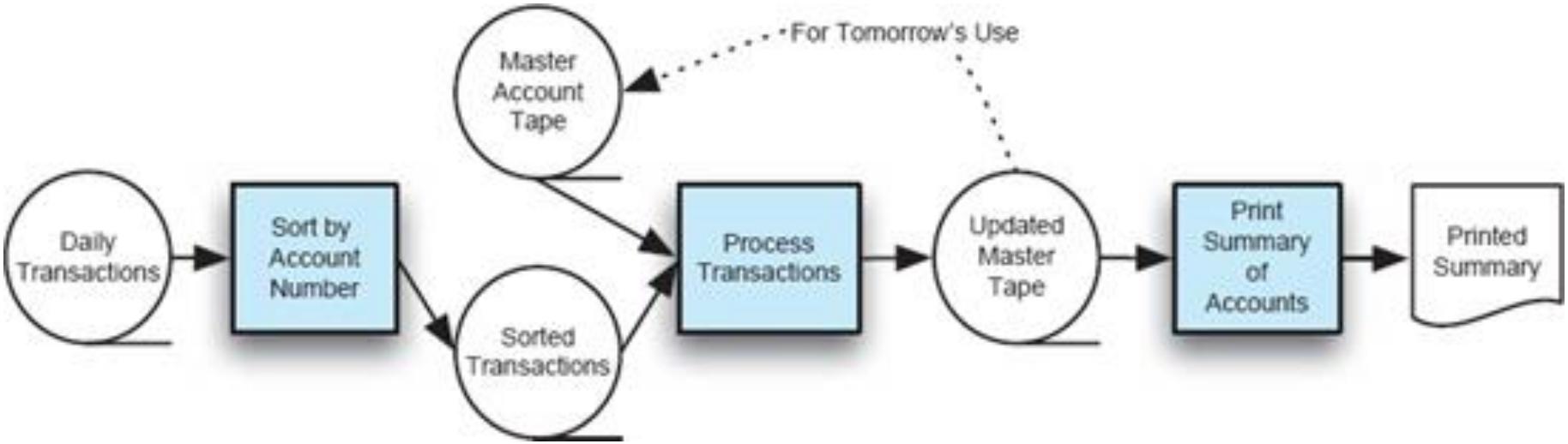
- En esencia considera el movimiento de datos entre unidades de procesamiento independientes.
- La estructura del sistema está basada en transformaciones sucesivas de los datos.
- Los datos entran al sistema y fluyen a través de los componentes hasta su destino final.
- Normalmente un programa controla la ejecución de los componentes (lenguaje de control)

# 5- BATCH – SECUENCIAL

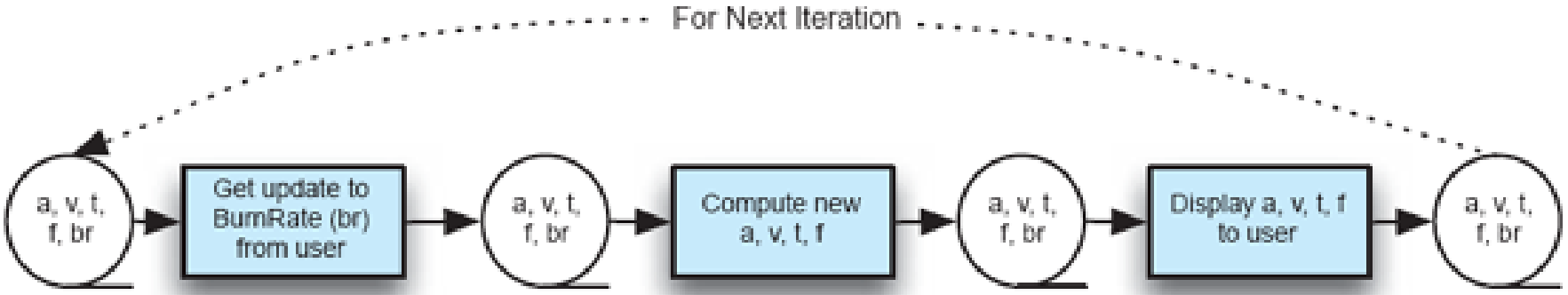
| | |
|---|---|
| DESCRIPCIÓN | Programas separados y ejecutados en orden. Los datos son pasados como un lote de un programa al siguiente. |
| COMPONENTES | Programas independientes |
| CONECTORES | Distintos tipos de interfaces: desde humana hasta web services |
| ELEMENTOS DE DATOS | Lotes de datos pasados de un programa al siguiente. |
| TOPOLOGÍA | Lineal |
| RESTRICCIONES ADICIONALES | Se ejecuta un programa a la vez, hasta que termina. |
| CUALIDADES | ▪ Sencillez<br>▪ Ejecuciones independientes |
| USOS TÍPICOS | ▪ Procesamiento de transacciones en sistemas financieros |
| PRECAUCIONES | ▪ Cuando se requiere interacción entre componentes.<br>▪ Cuando se requiere concurrencia entre componentes. |

# BATCH – SECUENCIAL - EJEMPLO

Ejemplo: Viejo procesamiento de transacciones financieras




Ejemplo: Lunar Lander



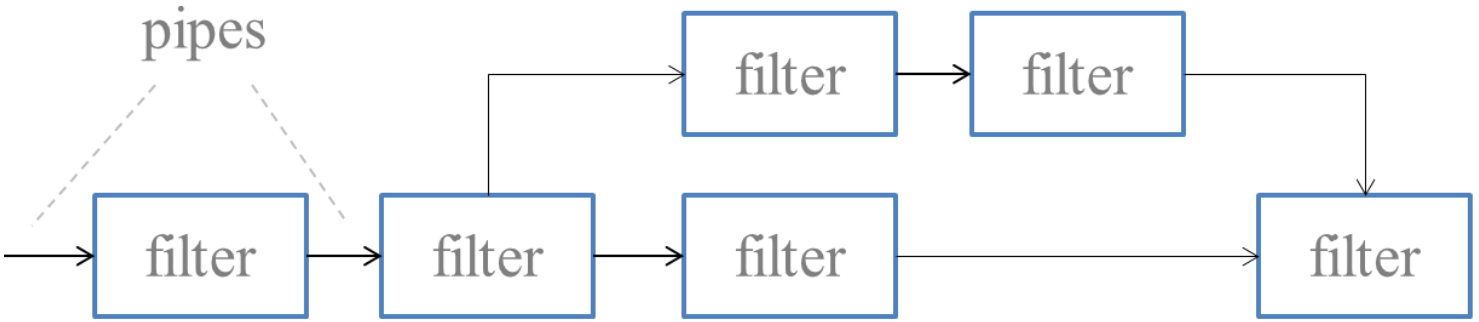
# 6 - PIPES AND FILTERS

- En el estilo Batch-Secuencial el siguiente programa espera hasta que su predecesor finaliza con el procesamiento del lote de datos completo.

- ¿Y si el siguiente programa puede procesar los elementos de datos tan pronto como empiezan a estar disponibles?

  - ✓ Los programas se ejecutan concurrentemente ➔ mayor performance

  - ✓ Los datos son considerados como "streams"



# PIPES AND FILTERS - ANÁLISIS

| | |
|---|---|
| DESCRIPCIÓN | Programas separados y ejecutados, potencialmente de manera concurrente. Datos son pasados como un stream de un programa al siguiente. |
| COMPONENTES | Programas independientes (filtros) |
| CONECTORES | Routers explícitos de streams de datos |
| ELEMENTOS DE DATOS | Stream de datos |
| TOPOLOGÍA | Pipeline (conexiones en T son posibles) |
| CUALIDADES | ▪ Filtros mutuamente independientes.<br>▪ Estructura simple de streams de entrada/salida facilitan la combinación de componentes.<br>▪ Flexibilidad: Agregar, eliminar, cambiar y reusar filtros |
| USOS TÍPICOS | ▪ Aplicaciones sobre sistemas operativos<br>▪ Procesamiento de audio, video<br>▪ Web servers (procesamiento de requerimientos HTTP) |
| PRECAUCIONES | ▪ Cuando estructuras de datos complejos deben ser pasadas entre filtros.<br>▪ Cuando se requiere interacción entre filtros. |

# EJEMPLO LUNAR LANDER

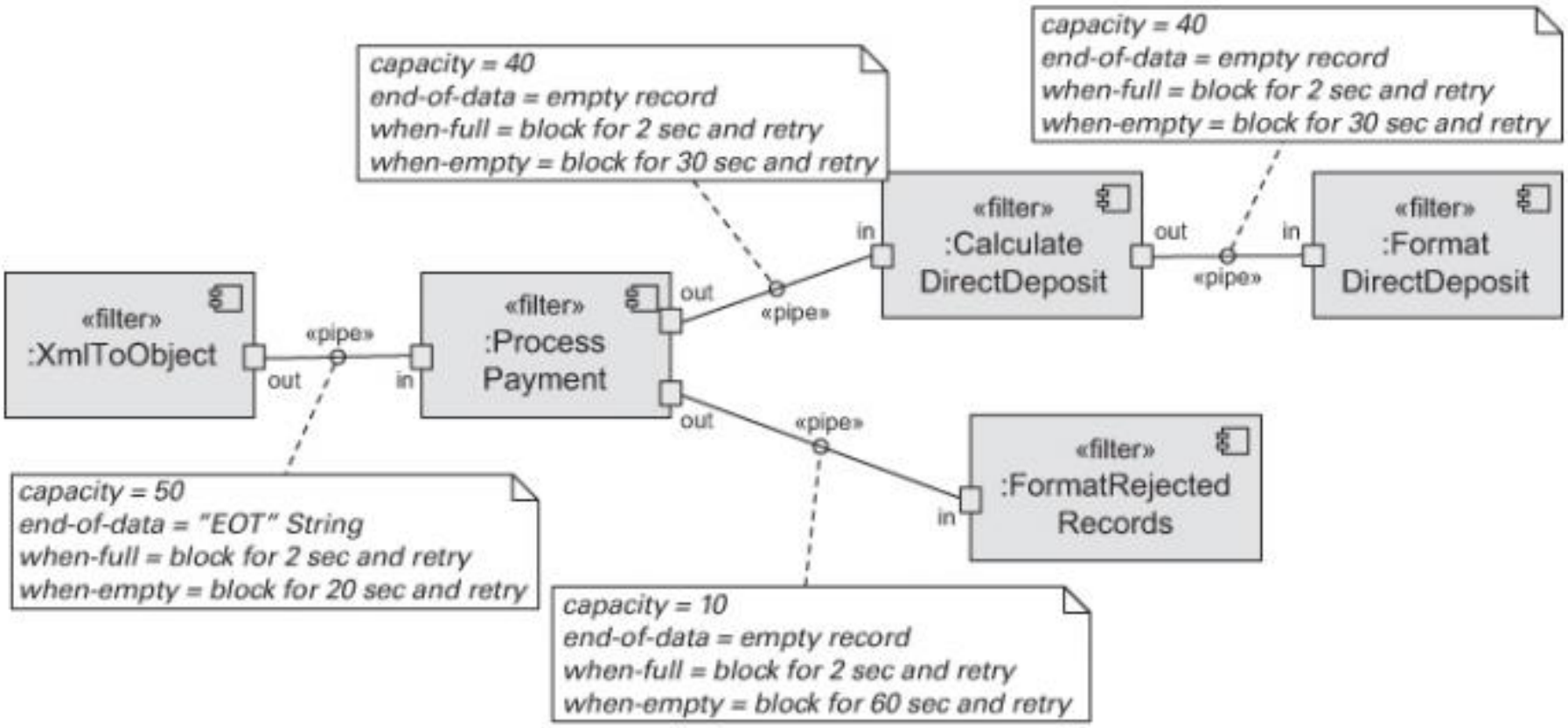
Cada componente cicla continuamente:

- `Get Burn Rate from User` - polling sobre la entrada del usuario
- `Compute New Values` - Calcula la velocidad de descenso de la nave
- `Display New Values to User` - actualiza la pantalla de acuerdo a los valores recibidos

# PIPES AND FILTERS - VARIANTES

| ESTILO | DESCRIPCIÓN |
|---|---|
| PIPELINES | Secuencias lineales de filtros |
| PIPES ACOTADOS | Cantidad limitada de datos en un pipe |
| PIPES TIPADOS | Datos fuertemente tipados |

# EJEMPLO – SISTEMA DE PROCESAMIENTO DE PAGOS



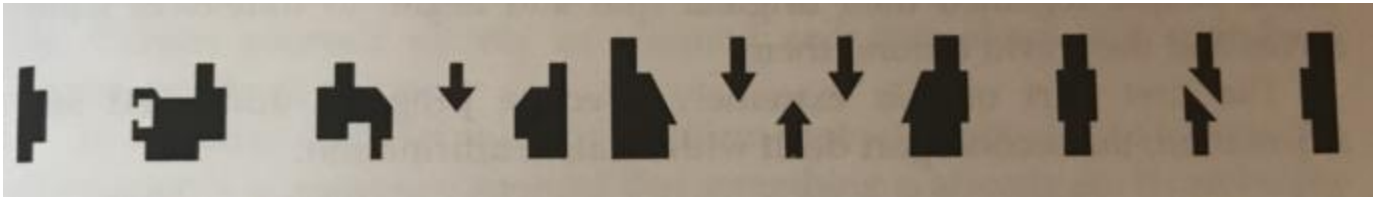
# RESUMEN

- CONCEPTOS Y ELEMENTOS
- COMPONENTES Y CONECTORES
- DIFERENCIAS ENTRE ESTILOS Y PATRONES
- ESTILOS SIMPLES:

1) ESTILOS INFLUENCIADOS POR LENGUAJES DE PROGRAMACIÓN
   - ✓ Programa Principal y Subrutinas
   - ✓ Orientación a Objetos
2) ESTILOS POR CAPAS
   - ✓ Máquinas Virtuales
   - ✓ Cliente Servidor
3) ESTILO DE FLUJO DE DATOS
   - ✓ Batch Secuencial
   - ✓ Pipe and Filter

# PARA PENSAR ... 1

# PARA PENSAR ... 2



**Elsa Estevez**
**ece@cs.uns.edu.ar**