# Módulo 6: Diseño arquitectónico


## Diseño y Desarrollo de Software (*1er. Cuat. 2019*)

Profesora titular de la cátedra:
Marcela Capobianco

Profesores interinos:
Sebastian Gottifredi
Gerardo I. Simari

**Licenciatura en Ciencias de la Computación – UNS**

# Licencia



# Una Analogia…

| **Construcción** | **Software** |
| --- | --- |
| • Relevar requerimientos para la construcción | • Relevar requerimientos para el sistema |
| • Crear un diseño/plano para satisfacer los requerimientos | • Crear un diseño de alto nivel que considere esos requerimientos |
| • Construir fisicamente el edificio en base a esos planos | • Escribir el código para implementar los elementos de ese diseño |
| • El edificio es habitado y usado | • Instalar el sistema para que luego sea usado. |

# Qué es una arquitectura de software

**Definición:** Conjunto de todas las ***principales*** decisiones de diseño sobre el Sistema.

- Es el "**plano**" para la **construcción** y **evolución** del sistema.
- Las decisiones de diseño abarcan:
  - Estructura
  - Comportamiento
  - Interacción
  - Propiedades no funcionales

# ¿Qué significa "*principales*"?

**Definición:** Conjunto de todas las ***principales*** decisiones de diseño sobre el Sistema.

- **Implica un grado de importancia** que le da a la decisión de diseño un estado arquitectónico.
- **No todas las decisiones impactan** en la arquitectura del sistema.
- **Depende de lo que los participantes** definan como *objetivos del sistema*.

# Qué es una arquitectura de software

- La arquitectura de software es un ***modelo***:
  - Simplificación de la realidad.
  - Abstracción cerrada: tiene cierta independencia.
- El modelado puede ser:
  - Informal
  - Semi-formal
  - Formal
- En la academia suele ser formal, mientras que en la industria generalmente es semi-formal o informal.

# Componentes y conectores

- Garlan (1993) ataca la tarea de definir la arquitectura planteando una tupla:

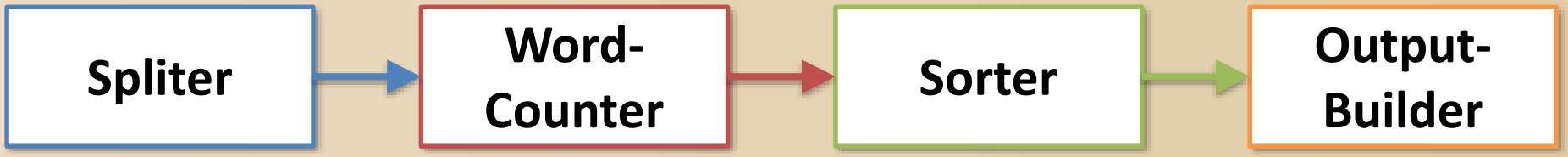
  SA={componentes, conectores, restricciones}

- Los **componentes y conectores** son elementos en un nivel arquitectónico.
  - Los componentes **no necesariamente** son clases de sistemas OO.
  - Los componentes y conectores pueden reflejar el ***estado en tiempo de ejecución*;** no son estáticos como una clase.

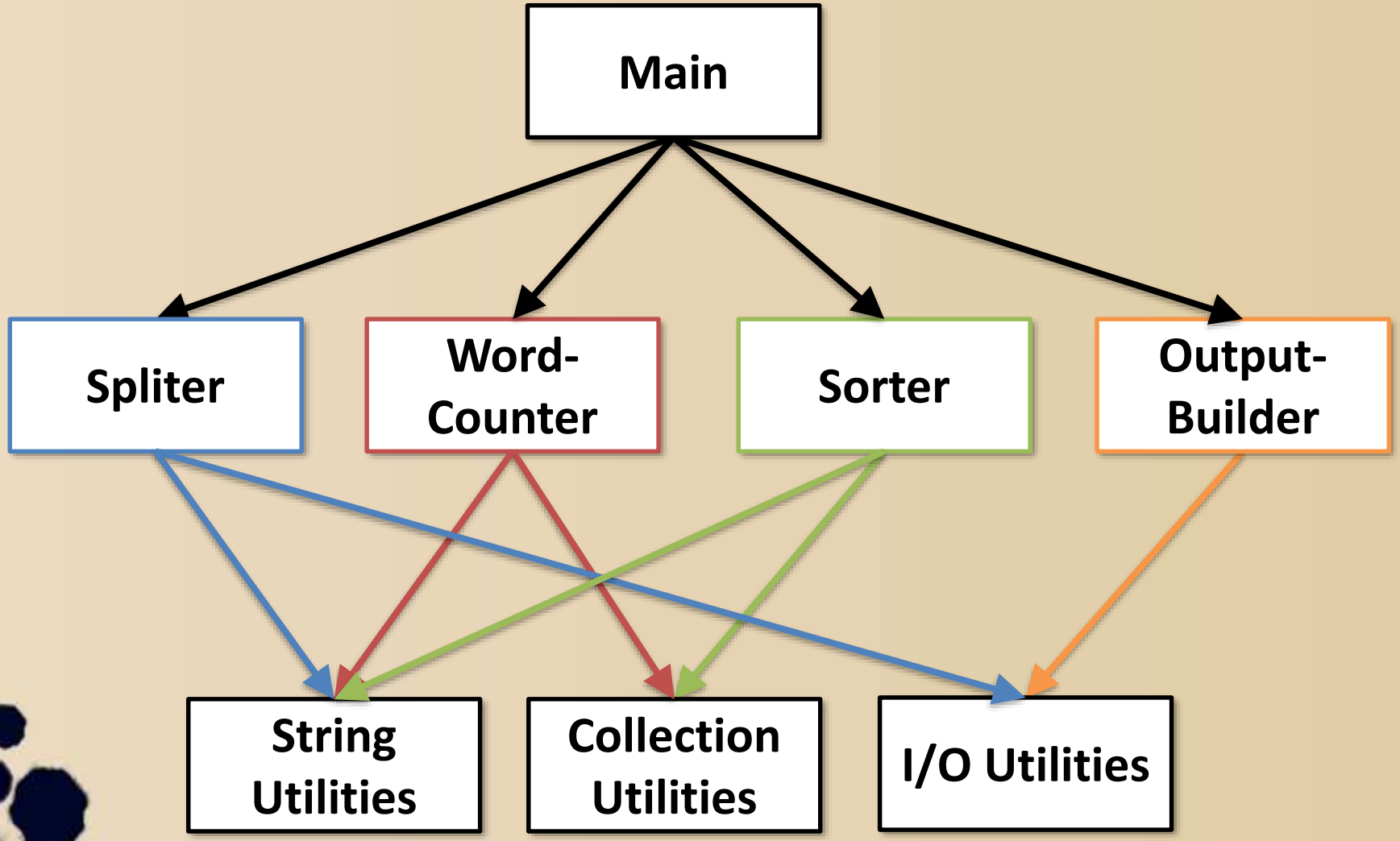
# *Word Frequency Counter*

Supongamos que queremos desarrollar un programa que toma un archivo de texto y devuelve pares (*palabra*, *frecuencia*) para cada palabra que ocurre en el texto, ordenados por frecuencia.

# *Word Frequency Counter*: Primera idea



# *Word Frequency Counter*: Otra posibilidad



# Arquitectura en el Tiempo

# Aspectos temporales

- Las **decisiones de diseño** pueden **realizarse** y/o **deshacerse** durante la **vida** del sistema.
- Por lo tanto, la arquitectura tiene un aspecto ***temporal***.
- En un **punto en el tiempo**, el **sistema** tiene una sola **arquitectura**.
  - En esa arquitectura podemos identificar que cosas quedan por hacer y cuales fueron realizadas/deshechas

# Prescriptiva vs. Descriptiva

**Arquitectura *prescriptiva***: Captura las decisiones de diseño tomadas previo a la construcción del sistema:

*Es cómo fue **concebido** el sistema*

**Arquitectura *descriptiva***: describe cómo el sistema ha sido construido:

*Es la arquitectura de acuerdo a cómo fue **implementado***

# Evolución arquitectónica

- Cuando el sistema **evoluciona**, idealmente va **realizando/refinando** decisiones de diseño de la ***arquitectura prescriptiva*** en decisiones de la ***arquitectura descriptiva***
- En ese ideal, en el tiempo, si **P** son las decisiones de diseño **prescriptivas** y **D** las **descriptivas**, tendríamos que **D** $\subseteq$ **P** (y al final D = P)

En la práctica, esto no suele pasar…

- Pereza de los desarrolladores
- Fechas de entrega demasiado estrictas
- Falta de documentación de la prescriptiva
- Técnicas o herramientas inadecuadas

# Degradación Arquitectónica

- ***Architectural drift* (desvío arquitectónico):** Cuando introducimos decisiones de diseño en la arquitectura descriptiva que:
  - *no están incluidas* en (o implicadas por) la arquitectura prescriptiva
  - pero *no violan* las decisiones de diseño de la arquitectura prescriptiva
- ***Architectural erosion* (erosión arquitectónica)*:*** Cuando introducimos decisiones de diseño en la arquitectura descriptiva que:
  - *violan* su arquitectura prescriptiva.

# Descripción de Arquitecturas

# Descripción de Arquitectura

- ***Architecture description (AD)*:** Producto usado para expresar una arquitectura.
- ***Architecture Description Language (ADL):*** El lenguaje utilizado para crear la AD.
- ***Concern:*** Interés en un sistema que es relevante a uno o mas de los participantes (stakeholders).

**Algunos Concerns:**
Funcionalidad – Factibilidad – Uso - Propiedades del sistema – Limitaciones – Estructura – Comportamiento – Performance – Recursos

# Descripción de Arquitectura

- ***Architecture view (vista):*** Producto que expresa la arquitectura de un sistema desde la perspectiva de un aspecto (*concern*) específico del mismo.

- ***Architecture viewpoint (punto de vista)*****:** Producto que establece las convenciones para la construcción, interpretación y uso de las vistas arquitectónicas para enmarcar un *concern* específico.

Una vista es lo que se ve.
Un punto de vista es desde dónde se mira; el punto o perspectiva determina lo que se ve.

# Descripción de Arquitectura

- ***Architecture framework*:** *Convenciones*, principios y *prácticas* para la *descripción de arquitecturas* que están establecidas en un *dominio específico*.
  - Los participantes del dominio (stakeholders),
  - Los concerns de ese dominio,
  - Los puntos de vista que nos permiten enmarcar esos concerns y
  - Las reglas de correspondencia integrando eso puntos de vista.

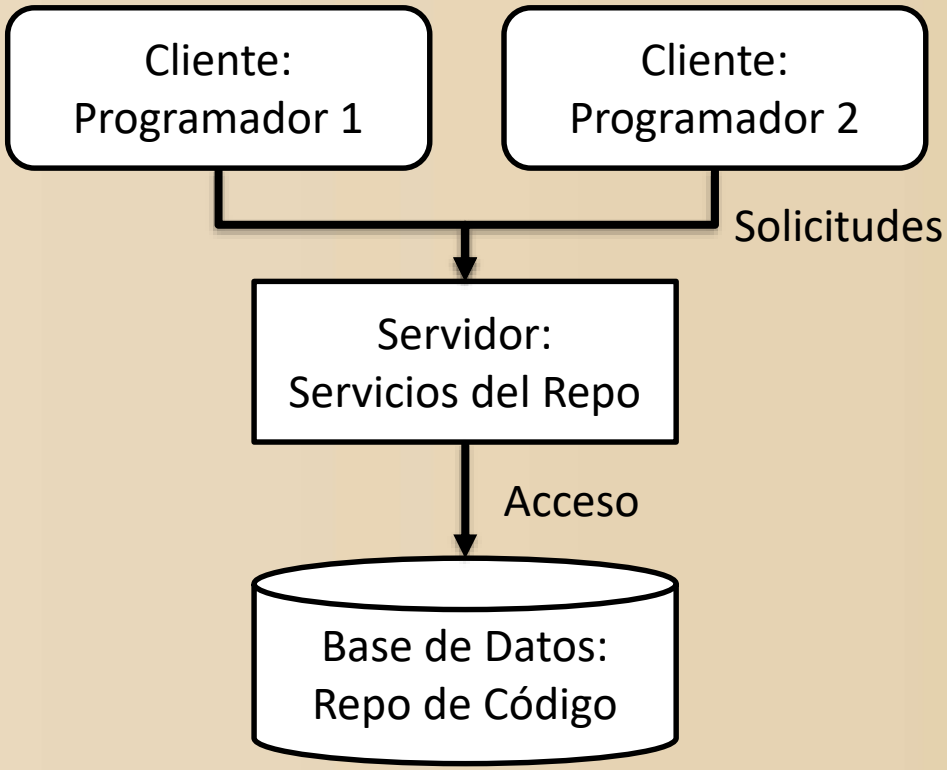
Nos ayuda a elegir qué viewpoints y vistas usar en cada caso.

# Componentes y conectores

- ***Componente***: es una entidad arquitectonica que
  - **Encapsula** un subconjunto de **funcionalidad** y/o **datos**,
  - **Restringe el acceso** a ese subconjunto con una **interface**,
  - Define que **dependencias** tiene para su **ejecución**

- ***Conector***: es una entidad arquitectónica que efectiviza y regula la interacción entre dos o mas componentes.

# *Ejemplo*: Sistema de gestión de versiones



# Componentes y conectores

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

- Los componentes suelen proporcionar los **servicios específicos de la aplicación.**

Ejemplos:

- Componentes **específicos de la aplicación**: bancarias, almacenes, vehículos
- Componentes de **reutilización limitada**: servidores web, los relojes, las conexiones
- Componentes **reutilizables**: bibliotecas de componentes GUI, librerías matemáticas

# Componentes y conectores

La interacción puede llegar a ser tan compleja e importante como la funcionalidad de los componentes

- Un **conector** es una pieza de la arquitectura que se ocupa de **llevar adelante y regular interacciones** entre los componentes.
- Se pueden describir independiente de los componentes.
- En muchos sistemas, los conectores son por llamadas a procedimientos, conexiones, streams o accesos a datos compartidos

# Componentes y conectores

- Es la herramienta **más importante y comunmente** usada.
- **Abstrae la *esencia*** de la ejecución del sistema.
  - Cada elemento tiene un **significado** en tiempo de ejecución.
  - Por ejemplo, el ***code repository server*** puede ser implementado por varias clases OO.
- Cada elemento debe tener un **significado claro y bien definido (*notación*).**

# *Viewpoints* más comunes

- **Lógico:** Captura las entidades lógicas de software del sistema y sus conexiones
- **Físico:** Captura las entidades físicas de hardware del sistema y sus conexiones
- **Deployment:** Captura como las entidades lógicas son mapeadas a entidades físicas
- **Behavioral:** Captura como es el comportamiento esperado del sistema

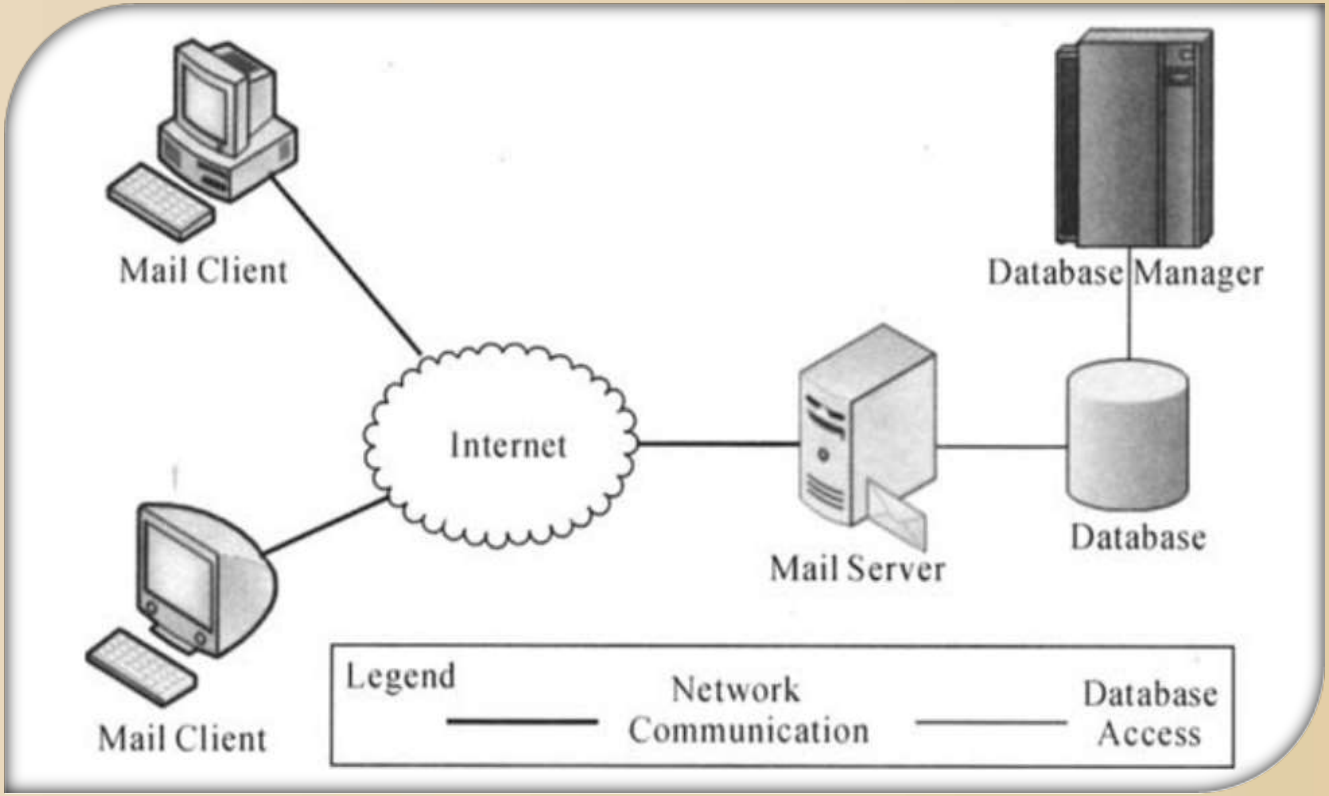
# Vista de Lógica

- **Descomponemos** el sistema completo en **varios conceptos** con un estilo *top-down.*
  - El proceso puede continuar hasta cumplir con el objetivo de los desarrolladores.
- Los **conceptos y sus asociaciones** construyen el **modelo** lógico del sistema.
- Se puede usar en primer lugar para **definir el vocabulario del sistema** y luego construirlo.
- Permite **dividir el trabajo** entre los **desarrolladores**.

# Vista de *deployment*

- Llena la **brecha** entre el **software** y **hardware**.
- Los elementos de software incluyen módulos, objetos, procesos, etc.
- Los de hardware se llaman *nodos*, y pueden ser un workstation, un mainframe, un server, un router o un dispositivo móvil.

# Vista de *deployment*



# Vista de *deployment*

- Principal utilidad:
  - Analizar las propiedades de los nodos (CPU, memoria, ancho de banda de las redes) expone los problemas existentes en este aspecto.
- Ejemplo:
  - Calcular y rastrear qué parte del sistema completo degrada la performance por motivos de HW, y luego simplificar u optimizar los algoritmos.

# Reutilizando la Experiencia en Arquitectura

# Experiencia en Arquitecturas

- Algunas **decisiones de diseño son mejores** que otras ya que resultan regularmente en soluciones con **propiedades superiores**.
- Comparado con otras alternativas posibles, estas soluciones son **más elegantes, eficientes, efectivas, escalables**, etc.
- Las **experiencias previas y de otros** se pueden utilizar para tomar estas decisiones

DSSA

Patrones

Estilos

# Domain-Specific Software Architectures (DSSA)

Un **DSSA** enmarca el conocimiento de como estructurar aplicaciones para un **dominio especifico**, mediante un **conjunto de componentes**:

- **Especializados** para un tipo de tarea
- Que posee una **estructura estandarizada (topología)** que resulta efectiva para construir aplicaciones exitosas en ese domino

- Los DSSAs **son de valor solo** para aplicaciones del **dominio** donde fueron **concebidos**.

**Ejemplos**:

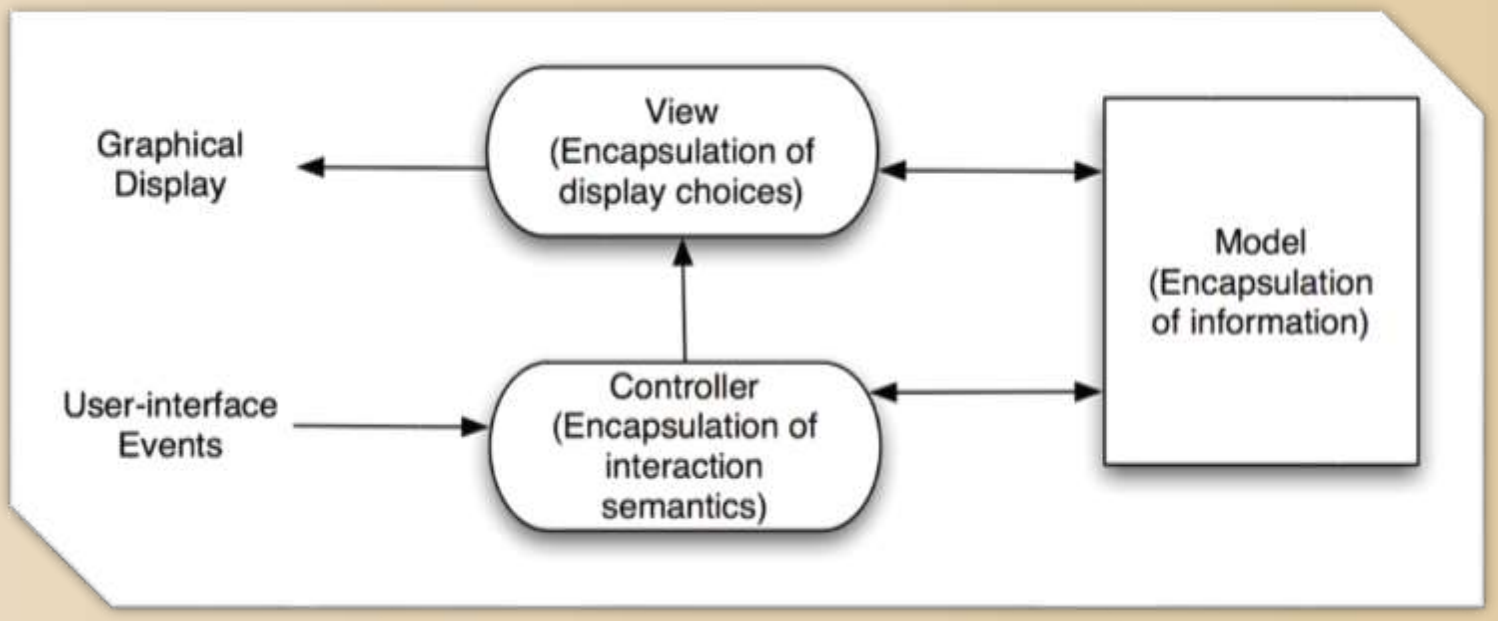
- ADAGE (*avionics*)
- AIS (*adaptive intelligent systems*)
- MetaH (*missile guidance, navigation and control systems*)

# Patrones arquitectónicos

- Un **patrón arquitectónico** es un conjunto de **decisiones de diseño** arquitectónicas:
  - **Aplicables** a un **problema recurrente** de diseño y
  - **Parametrizado** para poder ser aplicado en distintos contextos.
- Los patrones son una **abstracción de características de composición e interacción** de un conjunto de arquitecturas.

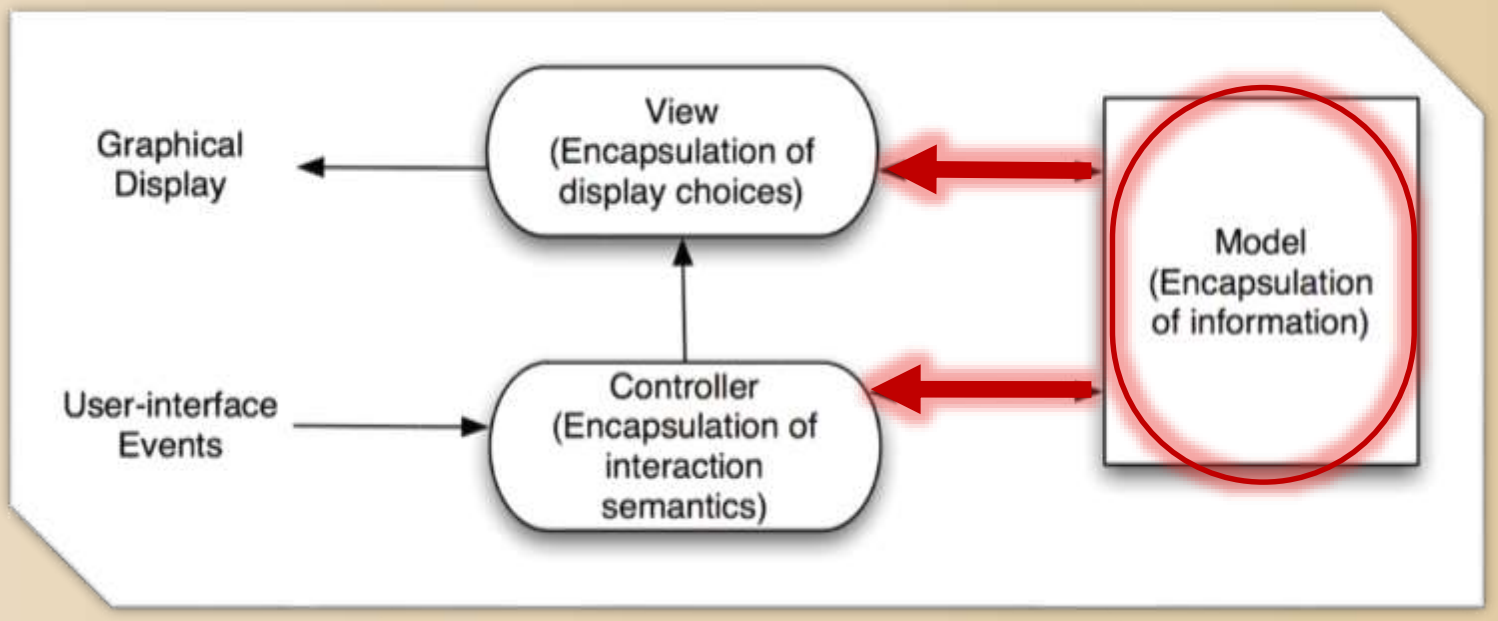
# Patrón *Model-View-Controller* (MVC)

- **Objetivo**: Separación entre información, presentación e interacción con el usuario.



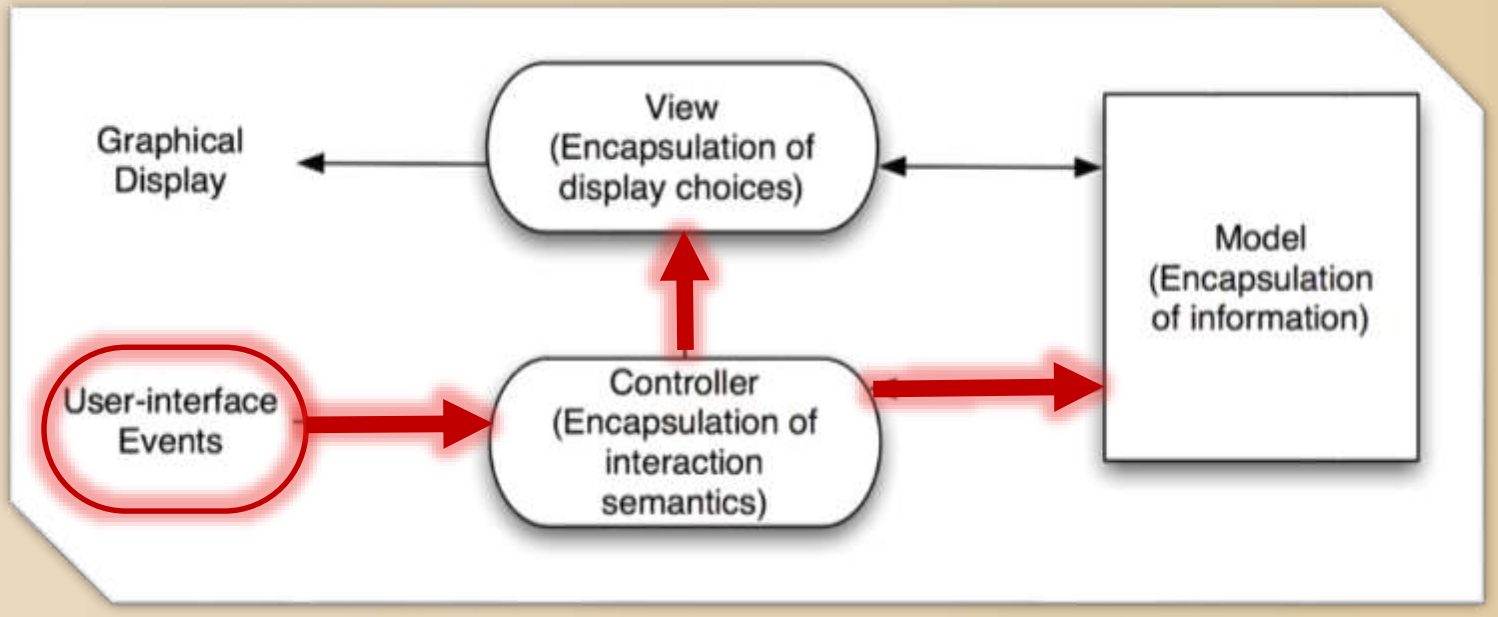
# Patrón *Model-View-Controller* (MVC)

- **Objetivo**: Separación entre información, presentación e interacción con el usuario.
- Cuando algo **cambia en el modelo**, se envía una **notificación** a la **vista** y al **controlador**.



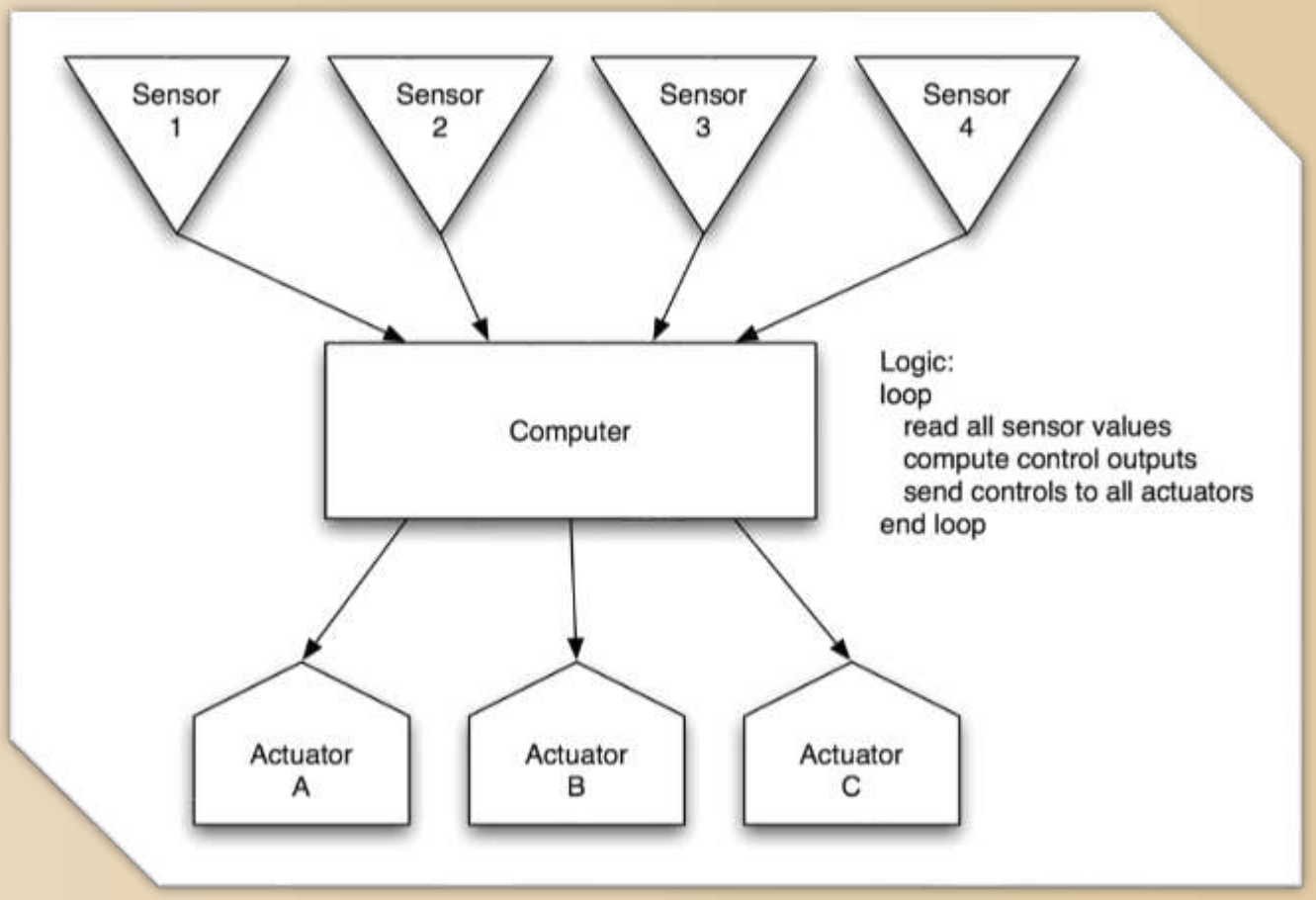
# Patrón *Model-View-Controller* (MVC)

- **Objetivo**: Separación entre información, presentación e interacción con el usuario.
- Ante **input del usuario**, el sistema de ventanas envía el **evento** del usuario al **controlador**. Si se requiere un cambio, el **controlador actualiza el modelo** y la **vista**



# Patrón *Sense-Compute-Control*

**Objetivo:** Estructurar aplicaciones de control embebido



# Estilos arquitectónicos

Un ***estilo arquitectónico*** es una colección de **decisiones de diseño** con un nombre determinado que:

- Son aplicables en un **contexto determinado amplio**.
- **Restringen** las decisiones de diseño arquitectónicas que son específicas a un sistema particular dentro de ese contexto.
- Determinan **cualidades beneficiosas** en el sistema resultante.

Existen muchos diferentes; los discutiremos en detalla más adelante.

# Propiedades básicas de los estilos

- Un ***vocabulario*** para elementos de diseño.
- Un conjunto de ***reglas de configuración***.
- Restricciones ***topológicas***
  - Por ejemplo: "*Un componente puede estar conectado a lo sumo con otras dos componentes*."
- Una ***interpretación semántica*** de los elementos que intervienen y que determinar las propiedades globales del sistema a partir de sus partes.

# Estilos vs Patrones

- Un **Patron** es una forma de resolver un problema arquitectónico recurrente.
  - **MVC** por ejemplo **resuelve el problema** de separar la interfaz de usuario del modelo y manejo,
  - **Sense-Comput-Control** es un patrón que **resuelve el problema** de tener que actuar en el contexto de multiples inputs de información

- Un **Estilo Arquitectónico** por el otro lado, es el nombre de un **diseño arquitectónico recurrente** (no existe para resolver un problema)
  - No resuelve un problema especifico solo nos dan una forma de organizar los componentes y los conectores, por ende el código

# Estilos Arquitectonicos: Casos de Estudio

# Estilos más comunes

- Tradicionales:
  - Programa principal y subrutina
  - OO
- En capas:
  - Sistema basado en Capas
  - Cliente-servidor
- Flujo de Datos:
  - Batch sequencial
  - Pipes and filters
- Intérpretes:
  - Interprete básico/metaintérprete
  - Código móvil
- Invocación Implícita:
  - Basada en eventos
  - Publish-subscribe
- Peer-to-peer
- Memoria Compartida:
  - Blackboard
  - Basado en reglas

# Estilos arquitectónicos

Para analizar los distintos estilos, veremos a la arquitectura como una *colección de componentes* junto con una *descripción de su interacción* (conectores).

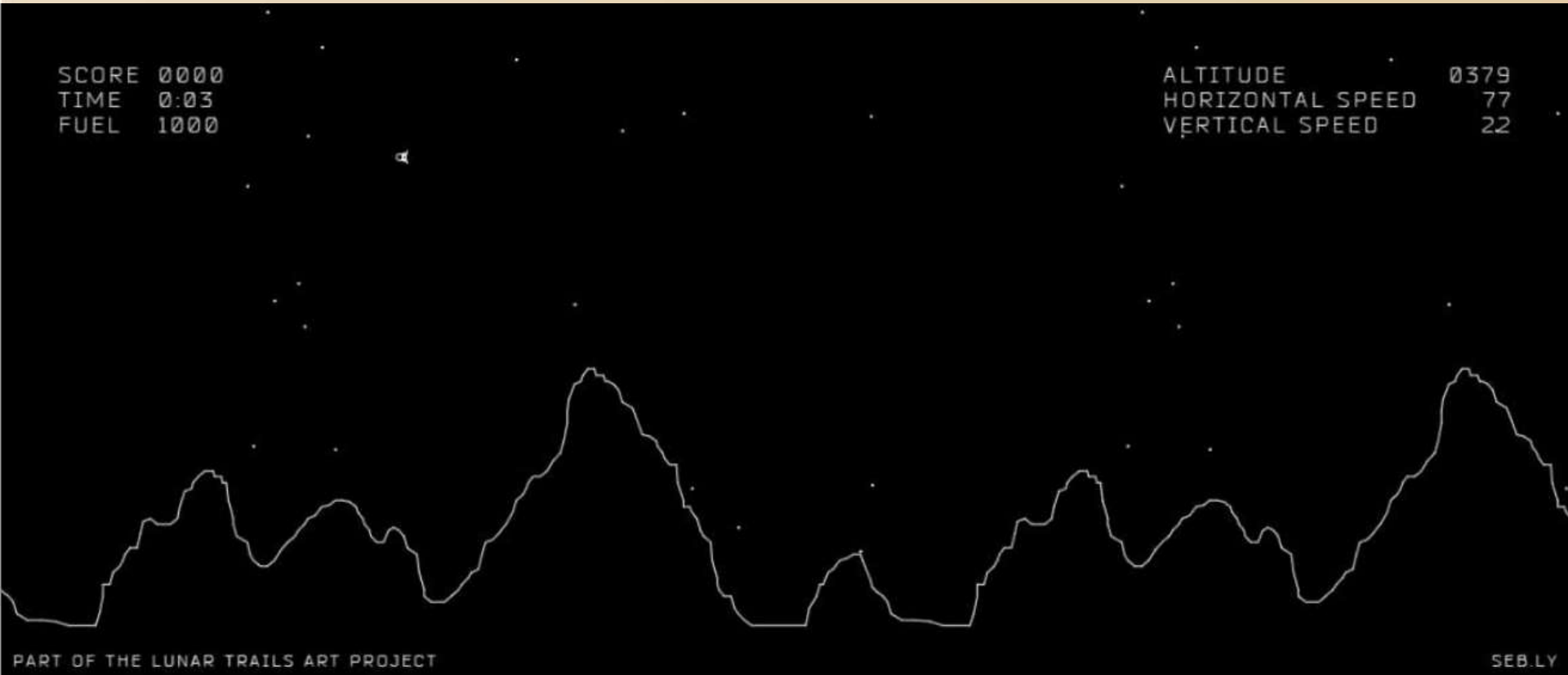
Clientes Servidores
Filtros
Niveles
Bases de datos

Llamadas a procedimiento
Broadcast de eventos
Protocolos de BD
Pipes

# Lunar Lander

Lunar Lander es un videojuego que trata sobre pilotear un módulo lunar y alunizarlo de manera segura en la luna.

url: http://moonlander.seb.ly/



# Lunar Lander

Reglas

- El piloto controla el descenso de un lunar Lander estilo Apollo
- Con las teclas “←” y “→” controla la rotación del Lander
- Con la tecla “↑” activa el propulsor
- Cuando el propulsor esta activo el Lander acelera en la dirección hacia donde este orientado
- El propulsor gasta combustible, que además es limitado
- La altitud inicial y la velocidad están predefinidas
- Si aterriza con una velocidad de descenso < 5, gana aunque no tenga combustible

# Sistema basado en Capas

Organización **jerárquica del sistema en capas**

- "Multi-level client-server"
- Cada capa **provee una interface (API)** para que sea usada por las capas de más arriba.
- Cada capa actúa como:
  - **Servidora:** provee servicios a las capas superiores.
  - **Clienta:** consume los servicios de las capas inferiores.

# Sistema basado en Capas

Los conectores son protocolos de interacción entre las capas

- Ejemplos:
  - Sistemas operativos,
  - Protocolos de redes.
  - máquinas virtuales

- El estilo clasico resulta de layers (capas) totalmente opacas.

# Sistemas en Capas: Notación

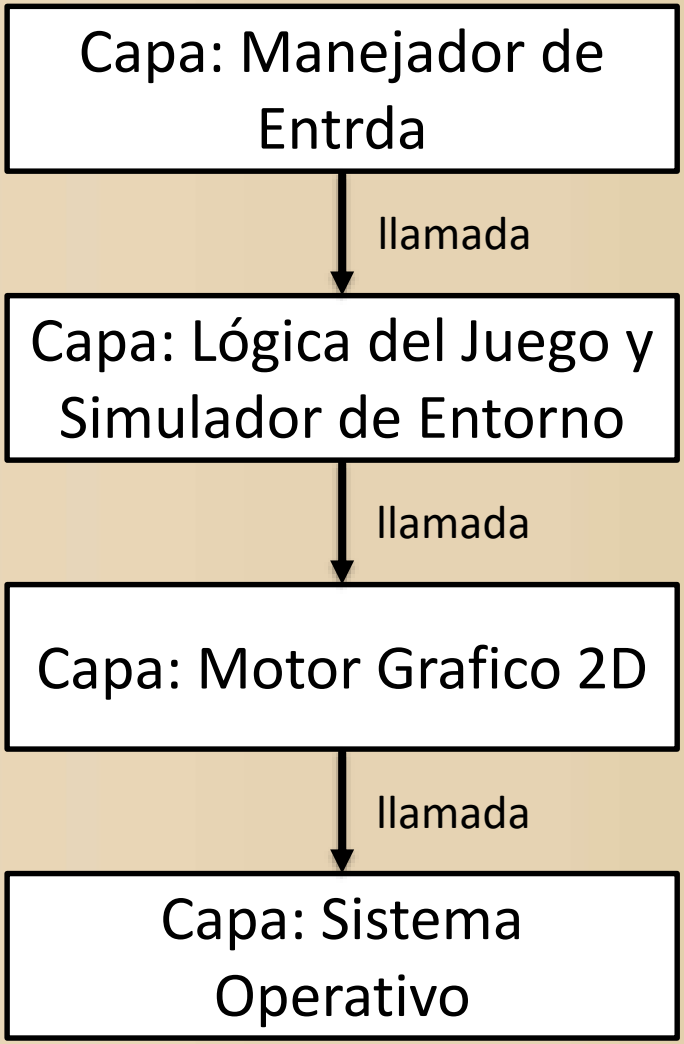
- Componentes


Son Capas que brindan uno o mas servicios a otras capas. Pueden estar compuestas de subprogramas o “subcomponentes”

- Conectores


Representan las llamadas entre las capas. Típicamente son llamadas a suprogramas

# Sistemas en Capas : Lunar Lander



# Sistemas en Capas : Lunar Lander

Capa: Manejador de Entrda

llamada

Capa: Lógica del Juego y Simulador de Entorno

llamada

Capa: Motor Grafico 2D

llamada

Capa: Sistema Operativo

Administra los inputs de teclado del usuario. Llama a la siguiente capa con esa info

Contiene todas las reglas del juego y administra el entorno simulado. Llama a la siguiente capa para mostrar gráficamente como se actualizo el estado del juego

Motor genérico capaz de trabajar con objetos en 2D. Llama a funcionalidades de la siguiente etapa para mostrar los gráficos

Además de las tareas comunes de admin de procesos, hace de interfaz con la capa anterior para abstraerlo de HW, haciendo llamadas a la capa siguiente

# Sistemas en Capas: Análisis

**Resumen:** Consiste de una secuencia ordenanda de capas. Cada capa provee servicios que pueden ser accedidos por subprogramas o componentes de una capa superior.

- **Componentes**: Capas
- **Conectores**: Llamadas
- **Datos**: Parametros de llamada y valores de retorno
- **Topología**: Lineal o grafo dirigido aciclicos
- **Cualidades**: Clara dependencia en la estructura, cambios en una capa C no afectan a capas no conectadas con C
- **Usos Típicos**: Diseño de Sistemas Operativos, Maquinas Virtuales y Protocolos de Red
- **Precauciones**: Diseños con muchos niveles pueden ser ineficientes.

# Cliente-Servidor

Buscan modelar situaciones donde varias piezas del sistema requieren los servicios computacionales de una entidad central

- Los **componentes** son **clientes** y **servidores**.
- Los **conectores** son protocolos de **interacción** en **redes** y llamadas a procesos remotos (**RPC**).
- Los **servidores no conocen** el número o identidades de los **clientes**.
- Los **clientes conocen** la identidad del **servidor**.
- Típicamente **múltiples clientes** pueden acceder al **mismo servidor**
- Los **clientes** son **independientes** entre si

# Cliente-Servidor: Notación

- Componentes

Servidor:
*<<Detalle>>*

Cliente:
*<<Detalle>>*

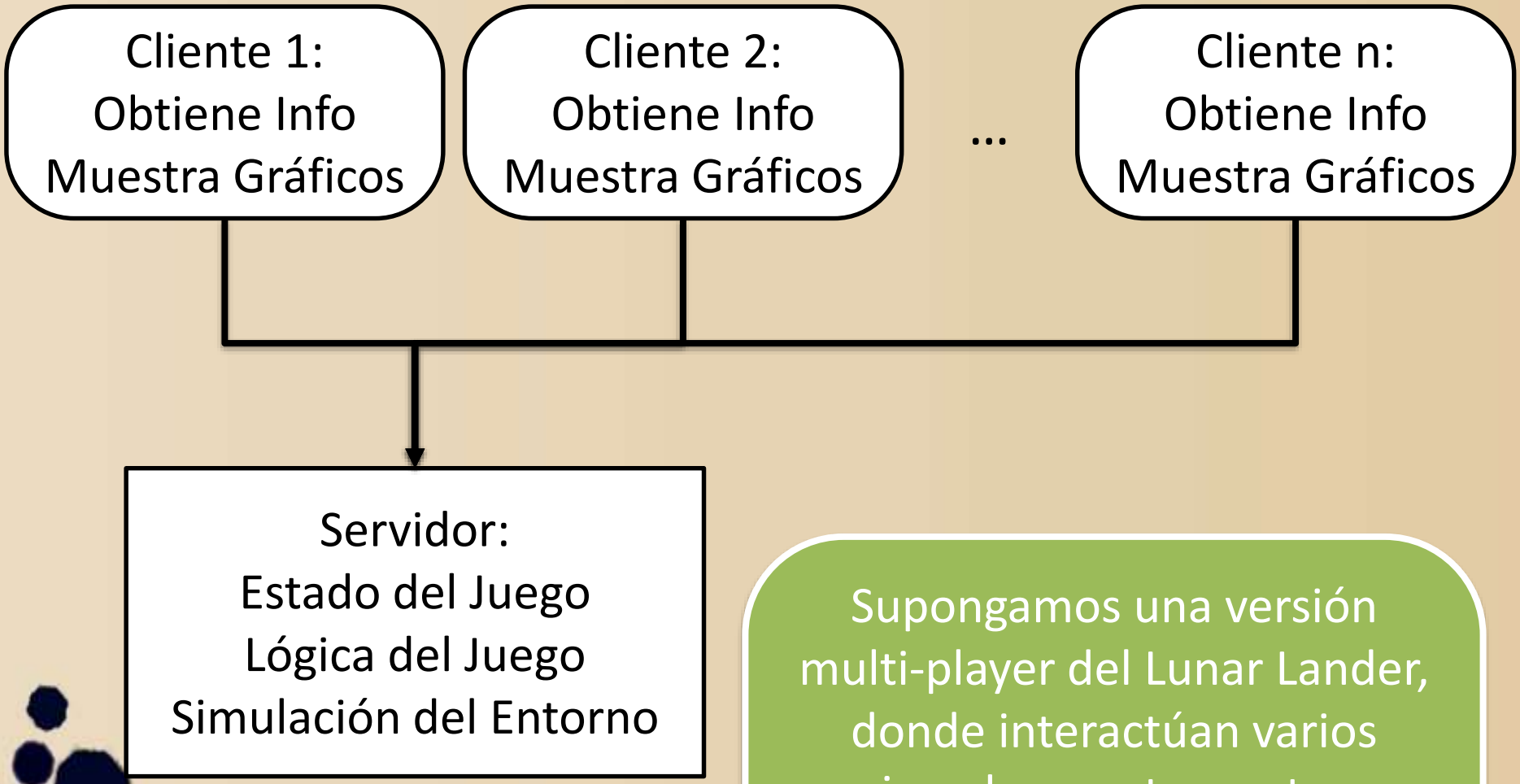
En el detalle indicamos de que funcionalidades del sistema se encarga

- Conectores

*<<Detalle>>*

Son llamadas a procedimientos remotos o protocolos de red

# Cliente-Servidor : Lunar Lander




Supongamos una versión multi-player del Lunar Lander, donde interactúan varios jugadores astronautas

# Cliente-S

**Los clientes** se encargan capturar el input del usuario y mostrar los gráficos del juego. La info del input la envían al servidor y le solicitan información del entorno actualizada para reflejarla reflejen gráficamente

Cliente 1:
Obtiene Info
Muestra Gráficos

Cliente 2:
Obtiene Info
Mues

Cliente n:
Obtiene Info
os

**El servidor** administra el estado global del juego y la simulación del entorno, recibe el input de los jugadores mediante los clientes y responde a los clientes con información del entrono actualizada

Servidor:
Estado del Juego
Lógica del Juego
Simulación del Entorno

Supongamos una versión multi-player del Lunar Lander, donde interactúan varios jugadores astronautas

# Cliente-Servidor: Analisis

**Resumen:** Los clientes piden servicios al/los servidores los cuales realizan las computaciones adecuadas y contestan con la información requerida. La comunicación se inicia por los clientes

- **Componentes:** Clientes y Servidores
- **Conectores:** Llamadas remotas, protocolos de red
- **Datos:** Parámetros de llamada y valores de retorno
- **Topología:** Dos niveles, con multiples clientes conectados con uno o mas servidores
- **Restricciones:** Los clientes no se pueden conectar entre si
- **Cualidades:** Centraliza la computación y los datos en el servidor. Un servidor poderoso puede atender a varios clientes
- **Usos Típicos:** Aplicaciones donde se necesita centralizar datos/procesos, o donde el computo requiere una maquina de alta capacidad y los clientes solo realizan tareas simples de interface
- **Precauciones:** Cuando esta limitado el ancho de banda y hay mucho clientes

# Tubos y s Filtros (Pipes and Filters)

- Buscan modelar el **flujo 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)

# Tubos y s Filtros (Pipes and Filters)

- Los **componentes** son **filtros**:
  - **Transforman** los flujos de datos de entrada en datos de salida.
  - La producción de la salida puede ser incremental.
- Los **conectores** son "**tubos**" (**pipes**), conductos para los flujos de **datos**.
  - Los **Datos** son considerados "streams"
- Los **filtros** son **independientes** (sin estado común) y no tienen conocimientos acerca de otro filtros.
  - Los **filtros** se pueden ejecutar **concurrentemente**

# Pipes and Filters : Notación

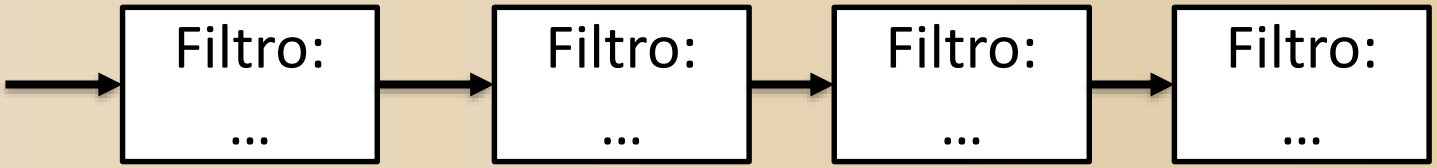
- Componentes

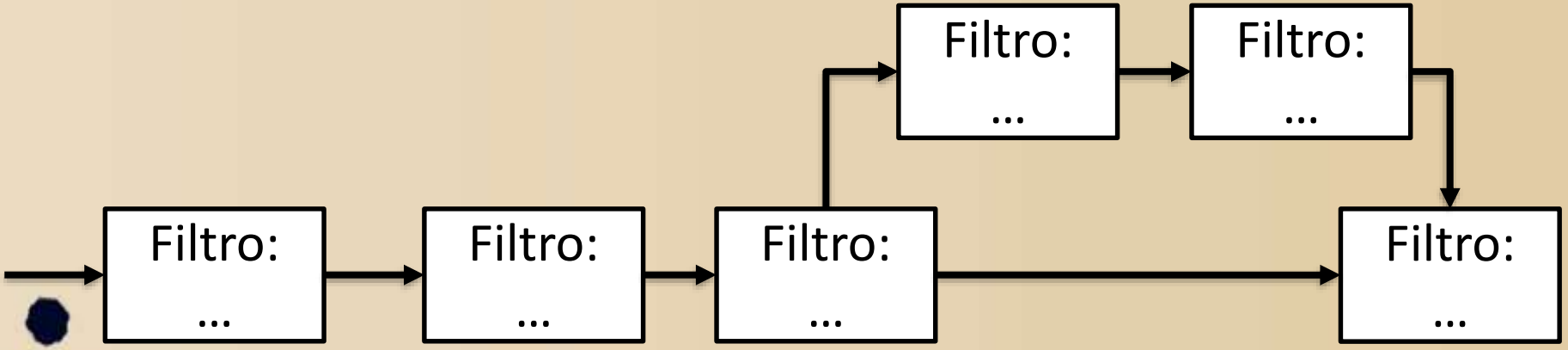

**Filtros**: Programas, procemientos o funciones que pueden ejecutarse independientemente

- Conectores

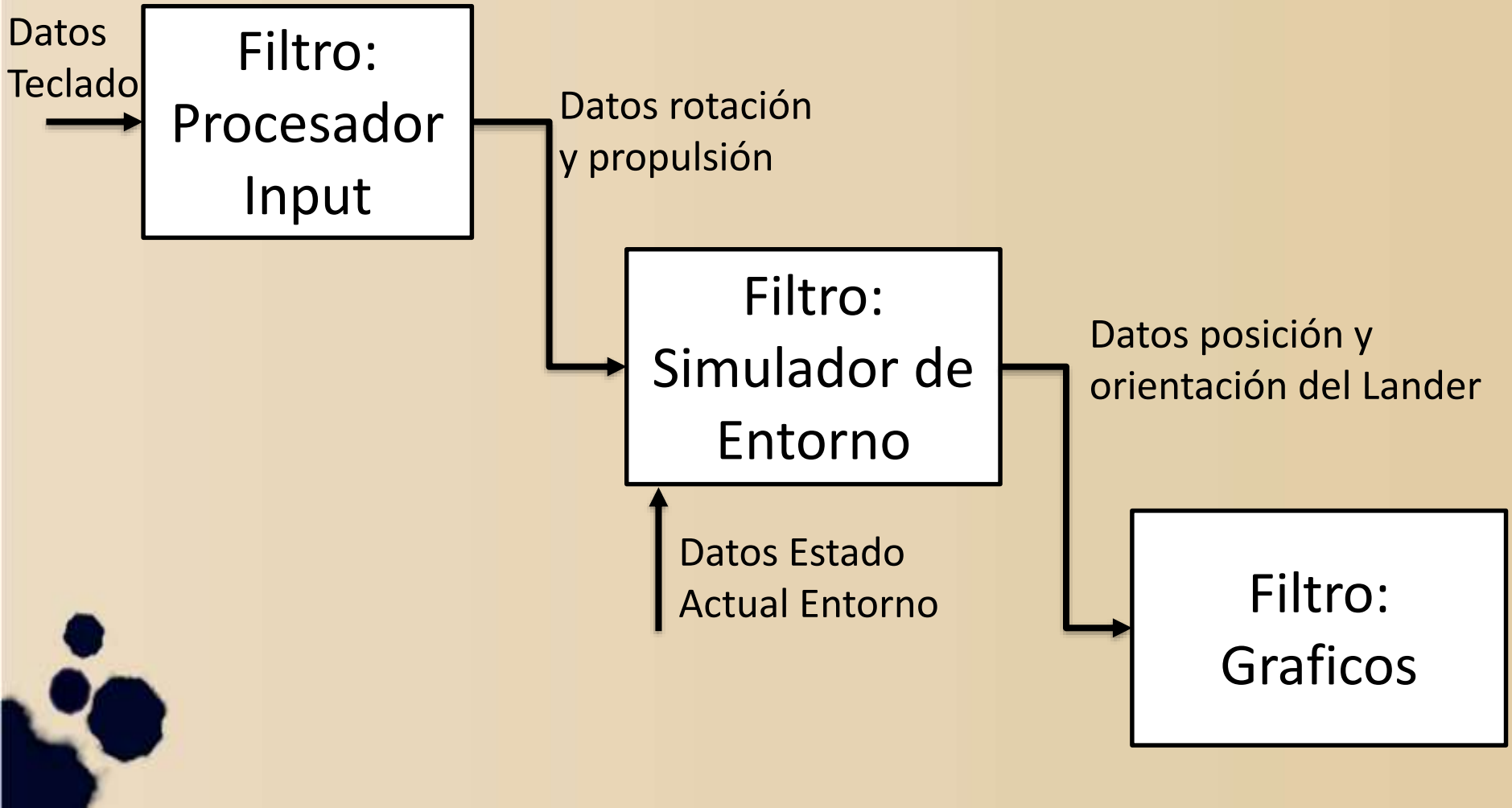

Conexiones explicitas entre los filtros o facilitadores/enrutadores de información de un filtro a otro

# Pipes and Filters

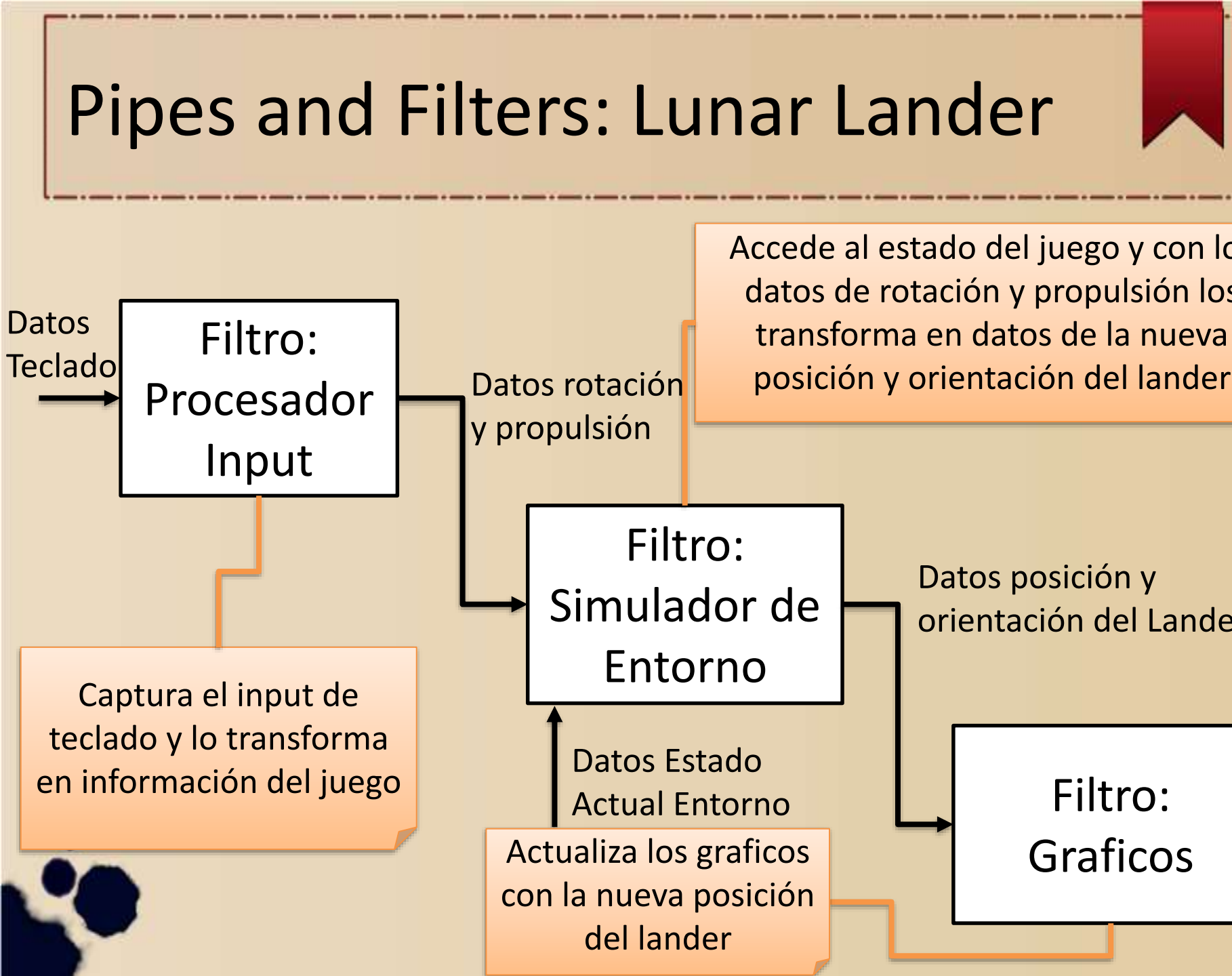


# Pipes and Filters: Lunar Lander



# Pipes and Filters: Lunar Lander



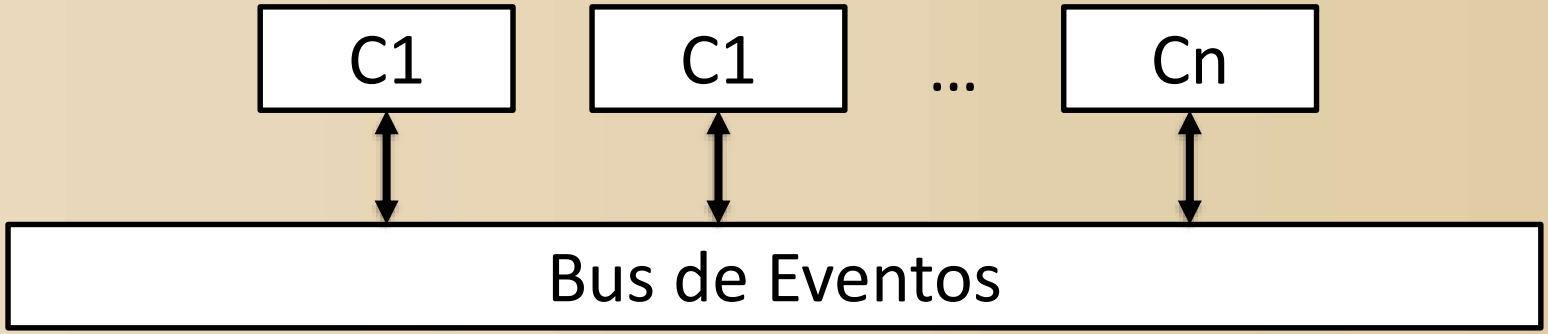
# Pipes and Filters: Análisis

**Resumen**: 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
- **Datos**: Stream de datos
- **Topología**: Pipeline (conexiones en T son posibles)
- **Cualidades**: Filtros mutuamente independientes. Estructura simple de streams de entrada/salida facilitan la combinación de componentes. Flexibilidad: Agregar, eliminar, cambiar y reusar filtros
- **Usos Típicos**: Aplicaciones sobre sistemas operativos. Procesamiento de audio, video.
- **Precauciones**: Cuando estructuras de datos complejos deben ser pasadas entre filtros. Cuando se requiere interacción entre filtros.

# Sistemas basados en eventos

- **Componentes** independientes **comunicándose sólo** enviando **eventos** a través de **conectores** a un **event-bus**
- Los **componentes emiten eventos** al event-bus en forma asincrónica, el cual luego los transmite a los otros componentes. Cada **componente puede reaccionar** ante la recepción de **un evento**, o ignorarlo.
- Los **buses de eventos** se encargan de:
  - optimizar la distribución de eventos
  - la replicación de eventos (transparente al emisor y receptor)



# Eventos: Notación

- Componentes

Gen/Cons:
*<<Detalle>>*

Bus de Eventos:
*<<Detalle>>*

En el encabezado se indica si el componente es un generador y/o un consumidor.

- Conectores

*<<Detalle>>*

Las flechas conectan a los Componentes con el bus de eventos. La dirección de la flecha indica si el componente genera o consume

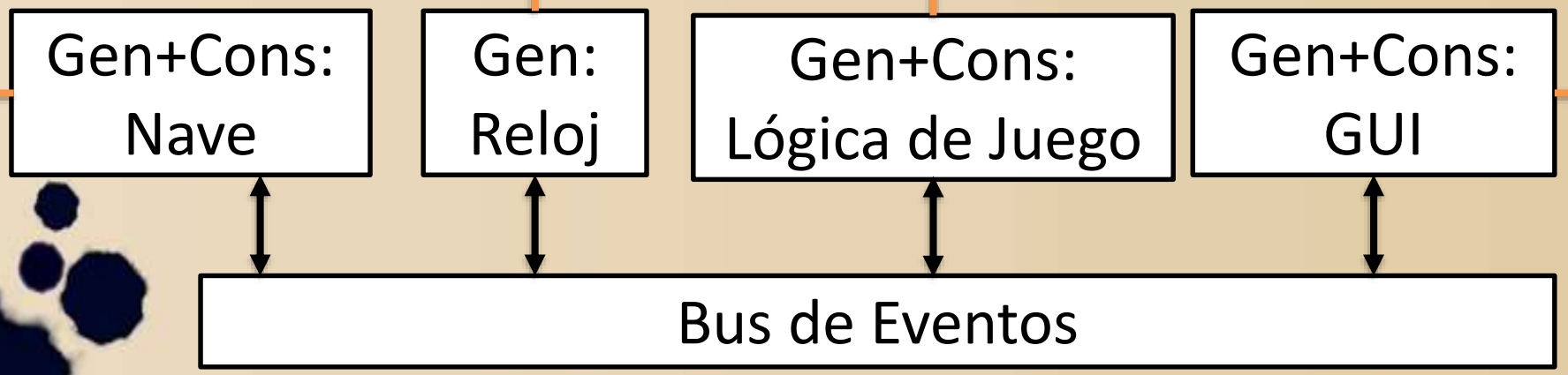
# Eventos: Lunar Lander

Mantiene y actualizael estado de la nave (altitud, combustible, velocidad, aceleración)

Maneja la pantalla. Recibe eventos del estado de la nave para mostrar los datos en pantalla Obtiene el input del usuario y emite notificaciones al bus de eventos

Envia ticks para que los otros componentes se actualicen

Recibe info del estado de la nave y del tiempo, determinando si el juego finalizó y, en ese caso, cuál fue el resultado final y el puntaje, notificándolo al event-bus.



# Eventos: Análisis

**Resumen**: Componentes independientes que asincrónicamente emiten y reciben eventos comunicados a través de event-buses.

- **Componentes**: Generadores y/o consumidores de eventos independientes y concurrentes. Buses de Eventos (Podría existir más de uno)
- **Conectores**: Conexiones con los Event-buses.
- **Datos**: Eventos
- **Topología**: Los componentes se comunican con el event-bus, no directamente entre ellos.
- **Cualidades**: Altamente escalable, Fácil de evolucionar, Efectivo para aplicaciones heterogéneas altamente distribuidas.
- **Usos Típicos**: Software de UI y Aplicaciones de área amplia que involucran partes independientes (mercados financieros, logística, redes de sensado)
- **Precauciones**: No existen garantías que un evento sea procesado, ni cuando lo será.

# Peer-to-Peer

- Consiste de una **red** de **componentes autónomos** y débilmente acoplados (**peers** o pares) que **colaboran** para proveer un servicio.
- **Todos los componentes son iguales** y ninguno puede ser crítico para la salud del sistema
- Cada **componente** provee y consume los **mismos servicios** y usa el mismo protocolo

# Peer-to-Peer

- La **información**, por lo general, es mantenida **localmente** en cada **componente**.
- **Interacción**
  - Un componente puede **interactuar** con **cualquier** otro componente
  - La **comunicación** es típicamente una interacción **requerimiento/respuesta**
- La **interacción** puede ser iniciada por cualquier parte (en el sentido cliente-servidor) y cada **componente es tanto cliente como servidor**

# Peer-to-Peer: Notación

- Componentes

Peer:
*<<Detalle>>*

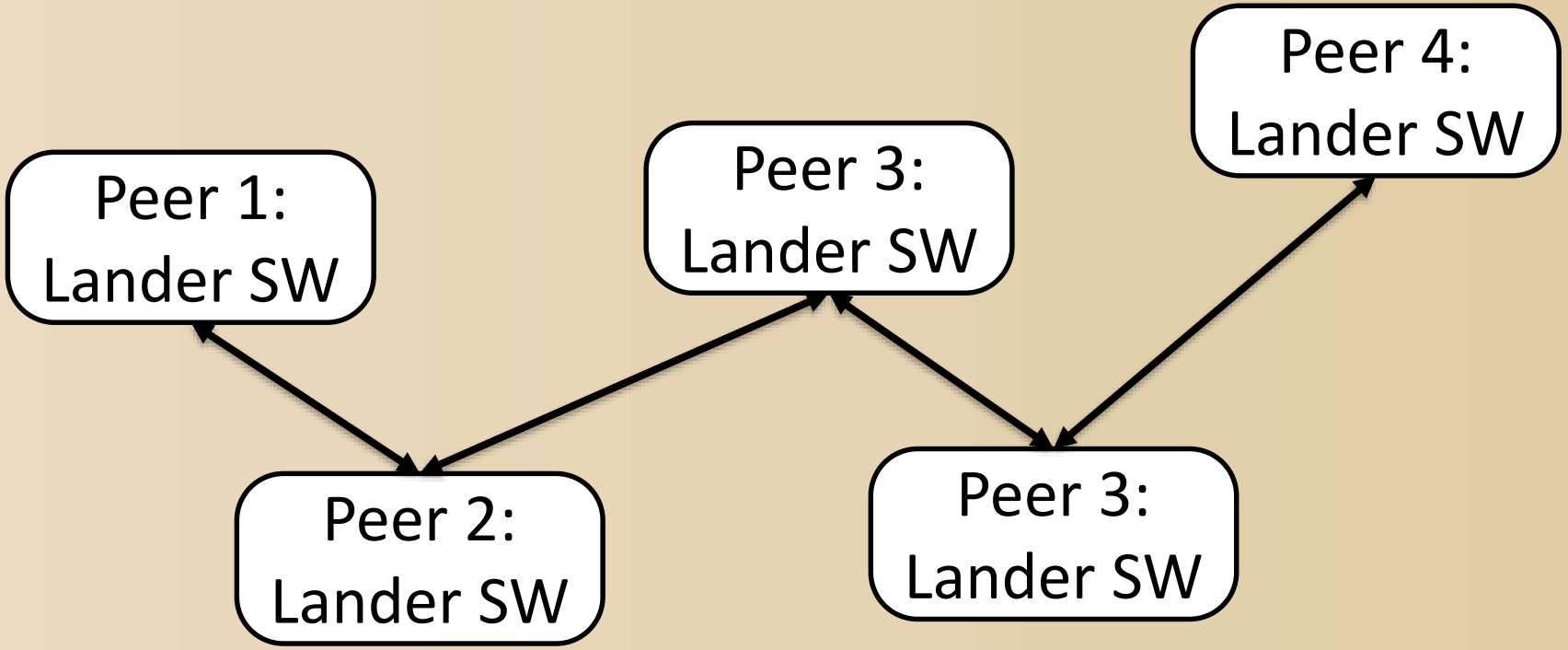
**Peers o Pares:** componentes con estado propio y control que se comunican con otros peers

- Conectores

*<<Detalle>>*

Conexiones explicitas entre los peers. Usualmente protocolos de red

# Peer-to-Peer: Lunar Lander




Supongamos que en la versión multi-player del Lunar Lander, una nave quiere saber si otra ya ha alunizado en un área, para evitar colisión. Un nave sólo  podrá comunicarse con otras dentro de un radio limitado.

# Peer-to-Peer: Análisis

**Resumen**: Estado y comportamientos son distribuidos entre pares que pueden actuar tanto como clientes como servidores

- **Componentes**: Peers-componentes independientes que tienen su propio estado y control.
- **Conectores**: Protocolos de red, generalmente propietarios.
- **Datos**: Mensajes de red.
- **Topología**: Red que puede tener conexiones redundantes entre peer. Puede variar arbitraria y dinámicamente.
- **Cualidades**: Altamente robusto de cara a la falla de un nodo. Escalable, en términos de acceso a los recursos y poder de cómputo. Computación distribuida
- **Usos Típicos**: Cuando las fuentes de información y operación están distribuidas: File sharing, Mensajería instantánea, Gridcomputing, etc.
- **Precauciones**: Cuando la recuperación de la información es crítica en tiempo y no puede hacer frente a la latencia propuesta por el protocolo.

# Arquitecturas Combinadas

La mayoría de los **sistemas** poseen una **combinación de estilos**:

- Combinaciones en forma jerárquica
- A nivel de componentes
- A nivel de conectores
- Permitir que un componente use distintos tipos de conectores.

# Notación en General para la Practica de la Materia

# Notación en General

Cuando se les solicita un diseño arquitectónico de componentes y conectores deben:

- En caso de ser requerido seleccionar **un estilo** o **combinación de estilos**
- Presentar un **diagrama** de **componentes y conectores**
- Dar una **descripción detallada** de cada **componente** (y de los conectores cuando no sea simple la interacción entre componentes)

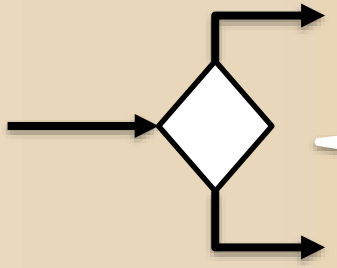
# Notación en General

- Componentes


Rectángulos rectos cuando hay una instancia de ese componente en la arquitectura


Rectángulos redondeados cuando hay varias instancia de ese componente en la arquitectura (clientes, peers, etc)

- Conectores

Ruteo Condicional

# Notación en General

- Componentes

Rol:
*<<Detalle>>*

Rol:
*<<Detalle>>*

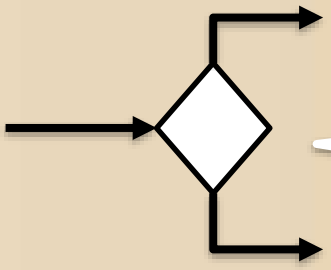
Rectángulos rectos cuando hay una instancia de ese

Si rol de un componente no se corresponde con ninguno de los vistos en clase deben detallarlo!

arquitectura (clientes, peers, etc)

- Conectores

*<<Detalle>>*



Ruteo Condicional

# Bibliografía


- R. N. Taylor, N. Medividovic, E. Dashofy: "*Software Architecture: Foundations, Theory, and Practice*". Wiley, 2009. Capítulos 3 y 4.
- ISO/IEC/IEEE 42010:2011. *Systems and Software Engineering — Architecture Description*.
- Z. Qin, J. Xing, X. Zheng: "*Software Architecture*". Springer, 2008. Capítulo 1.