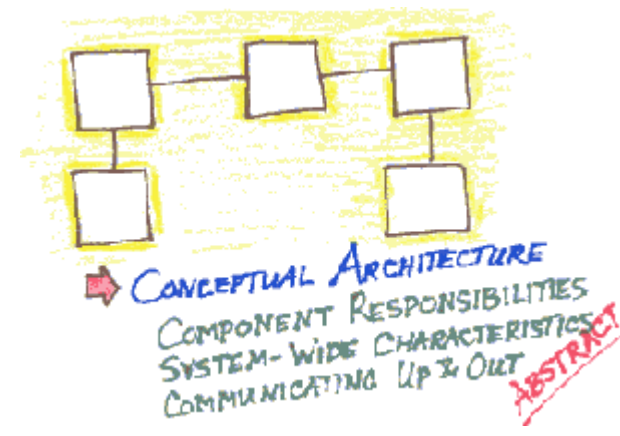


ARQUITECTURA DE SOFTWARE





System Context

The system plus users
and system dependencies



Containers

The overall shape of the architecture
and technology choices



Components

Logical components and their
interactions within a container



Classes

Component or pattern
implementation details

**Overview
first**



**Zoom and
filter**



**Details
on demand**

In *Software Architecture - Foundations, Theory and Practice*, I can find definitions for both. The problem is that I don't get what each one of them means in plain english:

Architectural Pattern.

An Architectural Pattern is a named collection of architectural design decisions that are applicable to a recurring design problem parameterized to account for different software development contexts in which that problem appears.

Architectural Style.

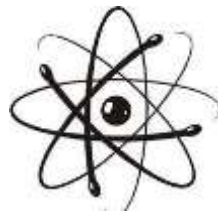
An Architectural Style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.

Un **patrón**:

- Es un paquete de decisiones de diseño que encuentra de manera repetida
- Tiene propiedades conocidas que permiten su reutilización
- Describe una clase de arquitectura
- Los patrones se “descubren” ... No se inventan → por eso no existe una lista definitiva.

Una **táctica**:

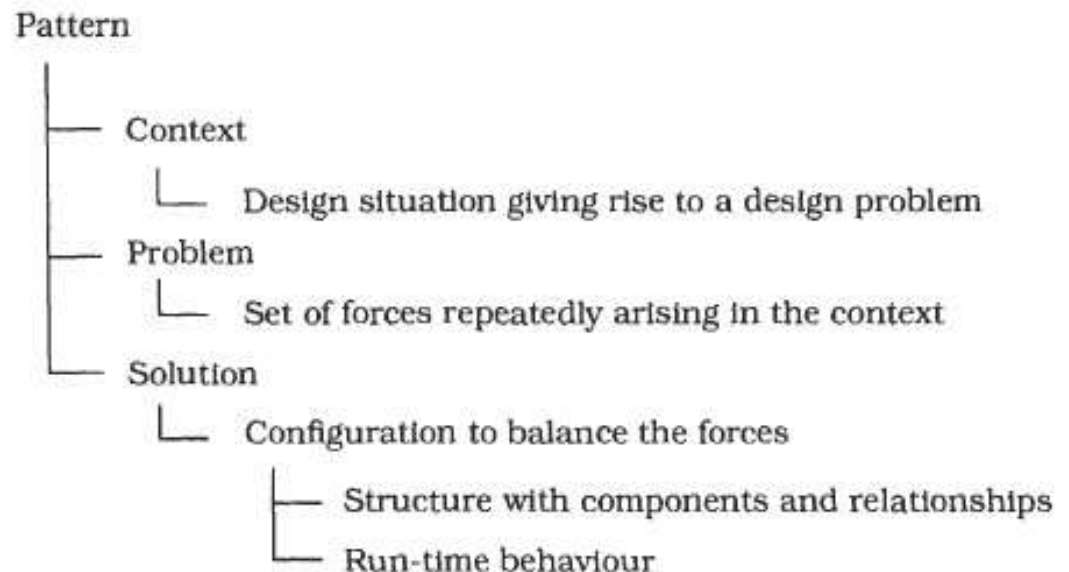
- Aplican sobre una estructura o artefacto del sistema
- La táctica es el bloque de construcción de un diseño
- La táctica es un átomo ... el patrón es una molécula. → muchos patrones se construyen a partir de diferentes tácticas



¿ Qué es un Patrón ?

Libro : “The Timeless Way of Building”, el arquitecto **Christopher Alexander** define el término “patrón” (pattern) como:

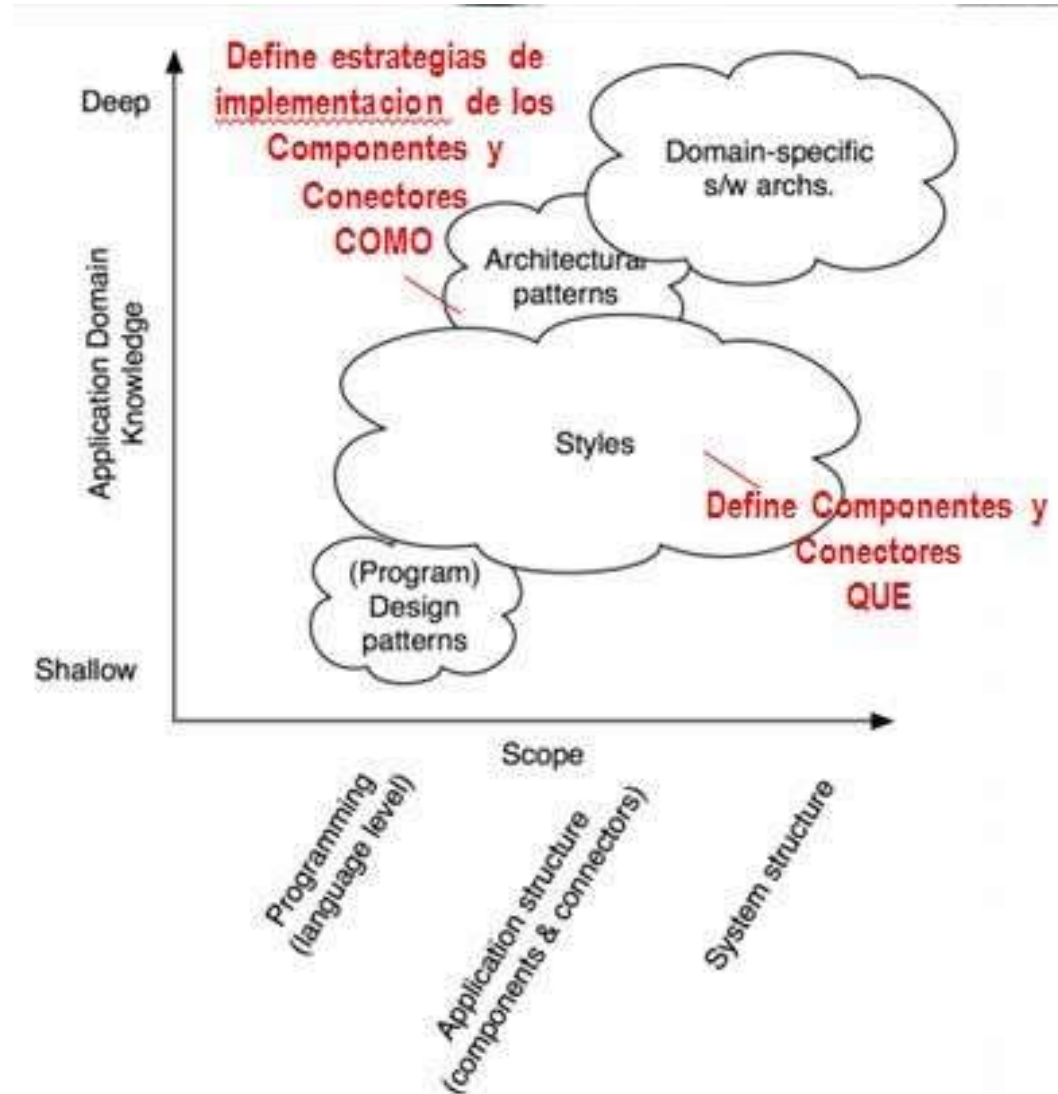
“Cada patrón es una regla de tres partes, los cuales expresan una relación entre un cierto contexto, un problema y una solución”.



Categorías de Patrones

- Architectural patterns
- Design patterns
- Idioms

Cada categoría consiste de patrones que tienen un rango similar de escala o abstracción.



¿ Qué es un Estilo de Arquitectura ?

- Estilo de arquitectura (architectural style) se refiere a una colección de decisiones de diseño arquitectónico que dan forma general al sistema a ser construido.
- Escoger el estilo apropiado es importante debido a que las decisiones de diseño posteriores se hacen en el contexto de este estilo.

El estilo define características y reglas que dan forma a la arquitectura como:

- Los **elementos básicos** de construcción : Cuales son los elementos claves y como se llaman típicamente los componentes y servicios.
- El **diseño topológico** entre los elementos básicos de construcción: Cómo los componentes interactúan unos con otros.
- Las **reglas** que especifican cómo los servicios pueden combinarse y utilizarse juntos.
- La familia de soluciones y **mecanismos de interacción** que determinan cómo los elementos coordinan a través de la topología.
- Los contextos y situaciones en las cuales el estilo es aplicable.

Ejemplito ...

1. Análisis

“Los vehículos deben poder viajar desde la parte alta a 120 km/h en línea recta y llegar a la parte baja en no más de 3 minutos.”

2. Clases de Dominio

Vehículo

Camino

3. Arquitectura

Cable

Columna

Barandilla

vehículo

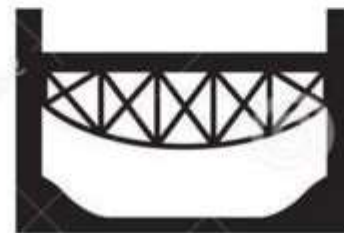
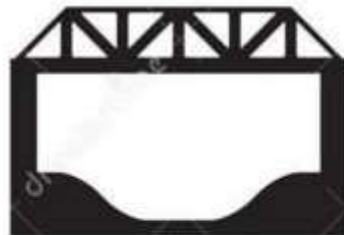
4. Diseño Detallado

Cable

Camino

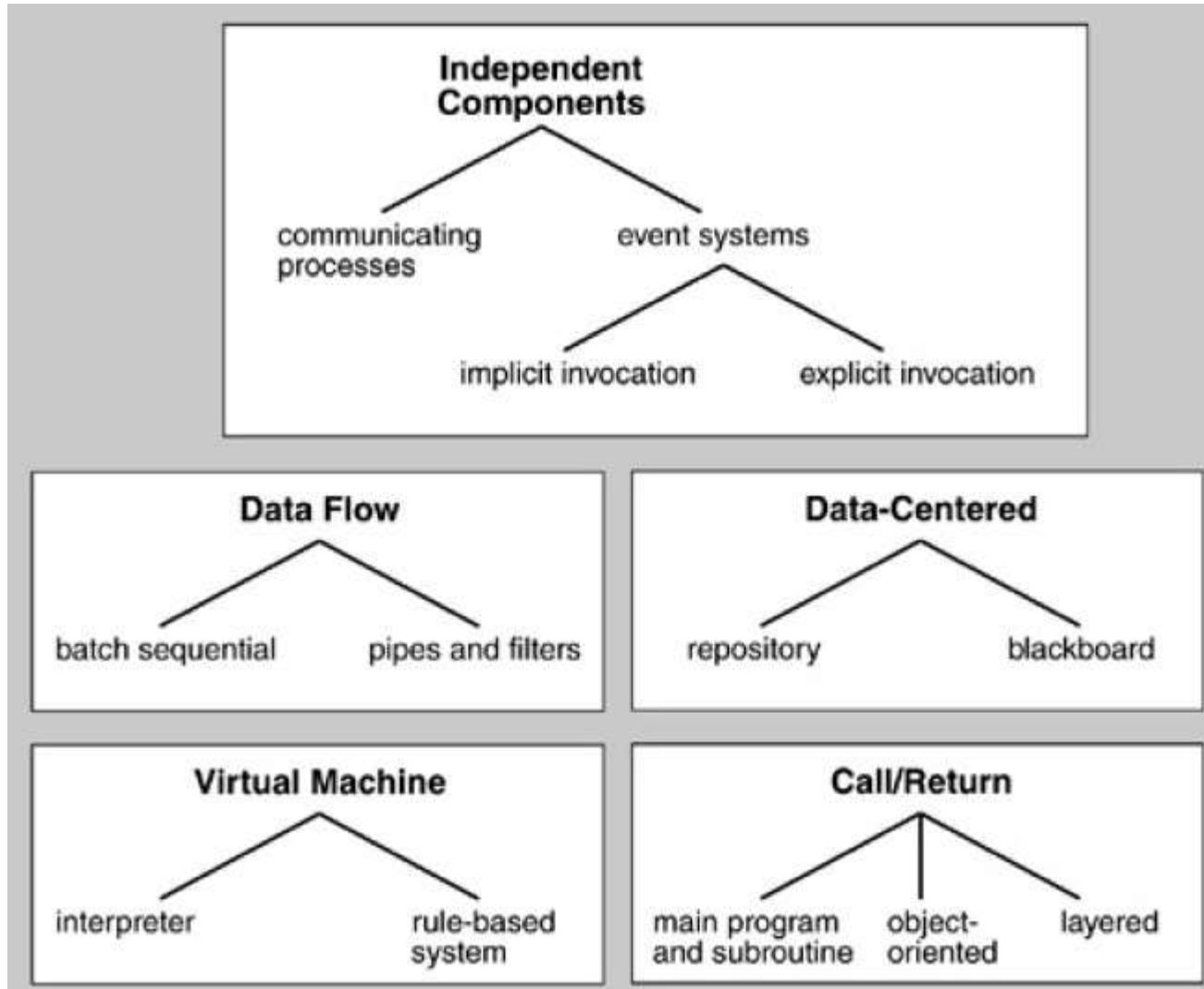
Columna

¿ que puente diseñamos ?



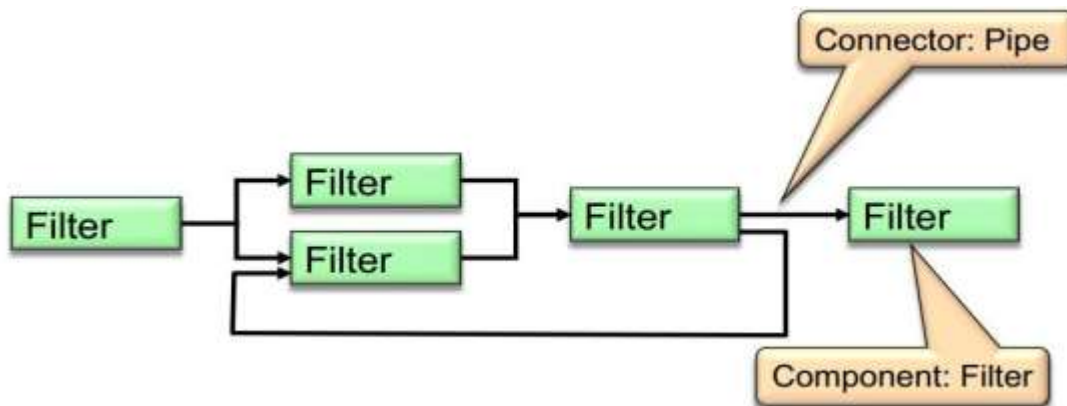


Catálogo organizado por relaciones IS-A



DataFlow: Pipes & Filters

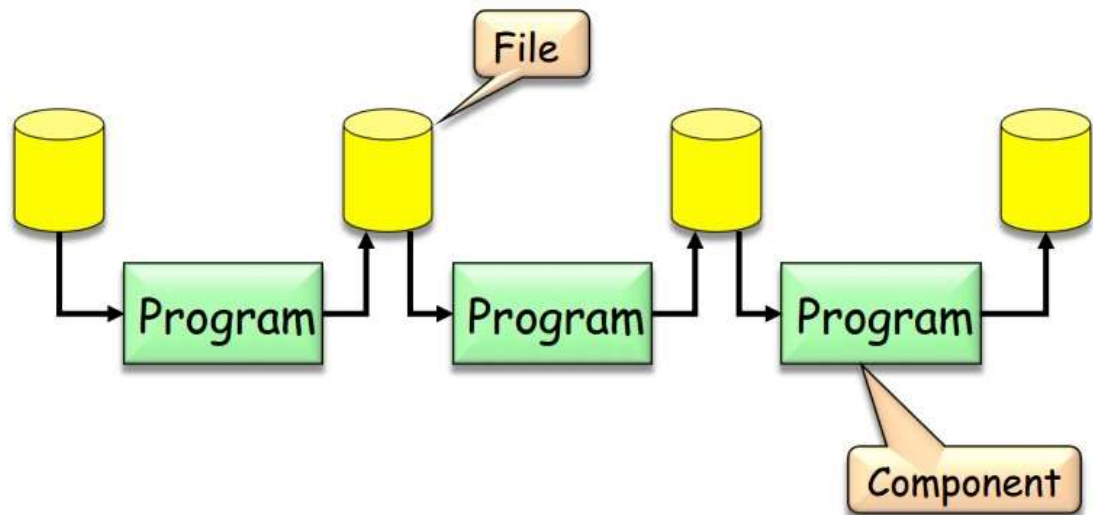
- Componentes (Filtros)
 - Son flujos de entrada para leer
 - Transforman datos localmente
 - Producen un flujo de salida
 - Conectores (Pipes)
 - Son los Flujos
- Los datos se procesan de forma incremental a medida que llegan.
 - La salida puede comenzar antes que la entrada sea totalmente consumida
- No comparten estado
 - No se conocen entre si



Los filtros deben ser independientes:

DataFlow: Batch Secuencial

- Arquitectura frecuente en computación científica y empresarial de procesamiento de datos
- Los componentes son programas independientes
- Los conectores son típicamente archivos.
- Cada paso se ejecuta hasta su finalización antes de que comience el siguiente paso



Call & return : main program

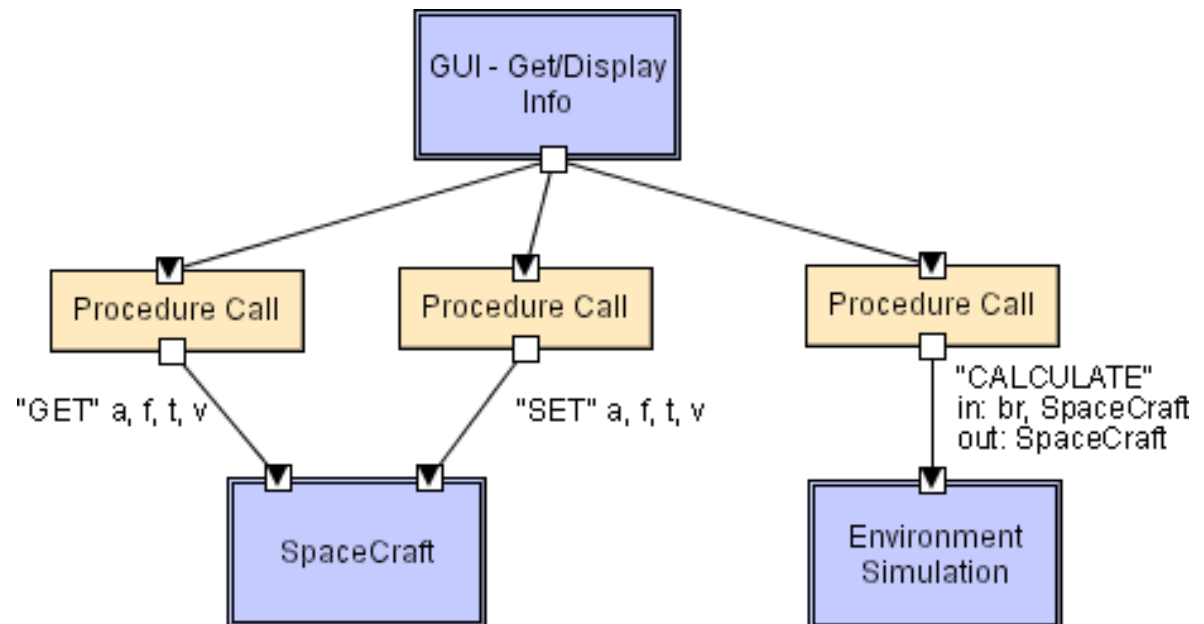
- Componentes: Rutinas
- Conectores: Llamadas a Rutinas
- Aspectos importantes:
 - Las rutinas corresponden a unidades de tarea que se deben ejecutar.
 - Las rutinas se combinan mediante estructuras de control.
 - Las rutinas se invocan por medio de interfaces (con argumentos)

Call & return : Object Oriented

- Componentes: Clases
- Conectores: mensajes e invocaciones a métodos

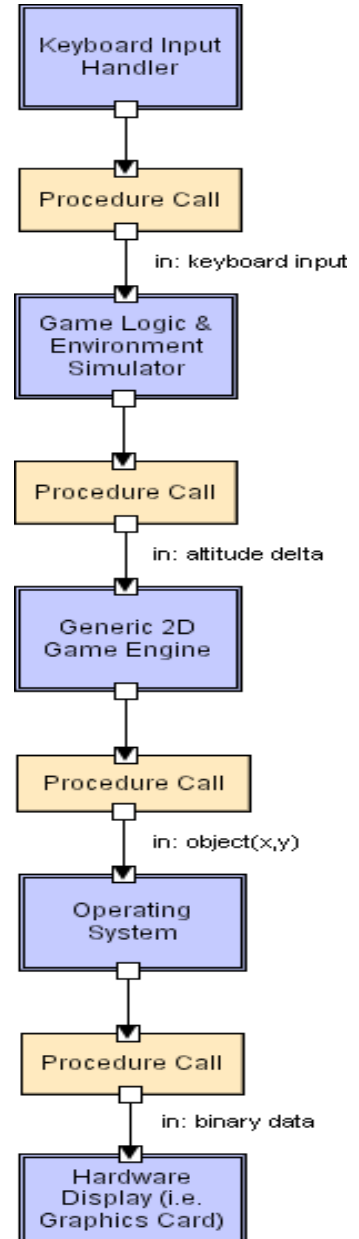
Aspectos importantes:

- Una clase describe un tipo de recurso y todos los accesos a él (encapsulación)
- Ocultamiento de información a la clase cliente



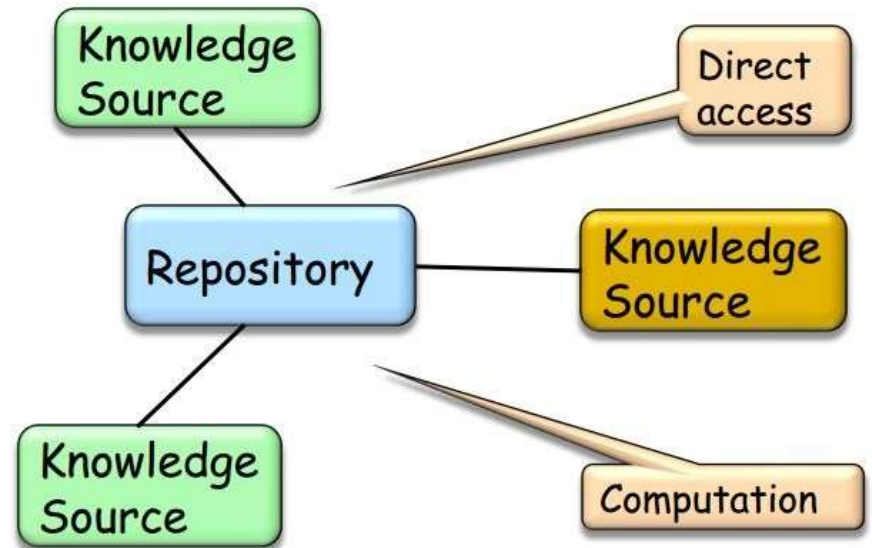
Call & return : Layered

- Tambien se cataloga como **Jerárquico**
- Componentes :
 - Grupos de subtareas que implementan una abstracción en determinado nivel de la jearquía.
- Conectores :
 - Una capa depende solamente de capas inferiores no tiene conocimiento de las capas superiores
- Cada capa proporciona servicios a la capa encima de ella y actúa como un cliente de la capa de abajo.
- Cada capa recoge servicios a un determinado nivel de abstracción.



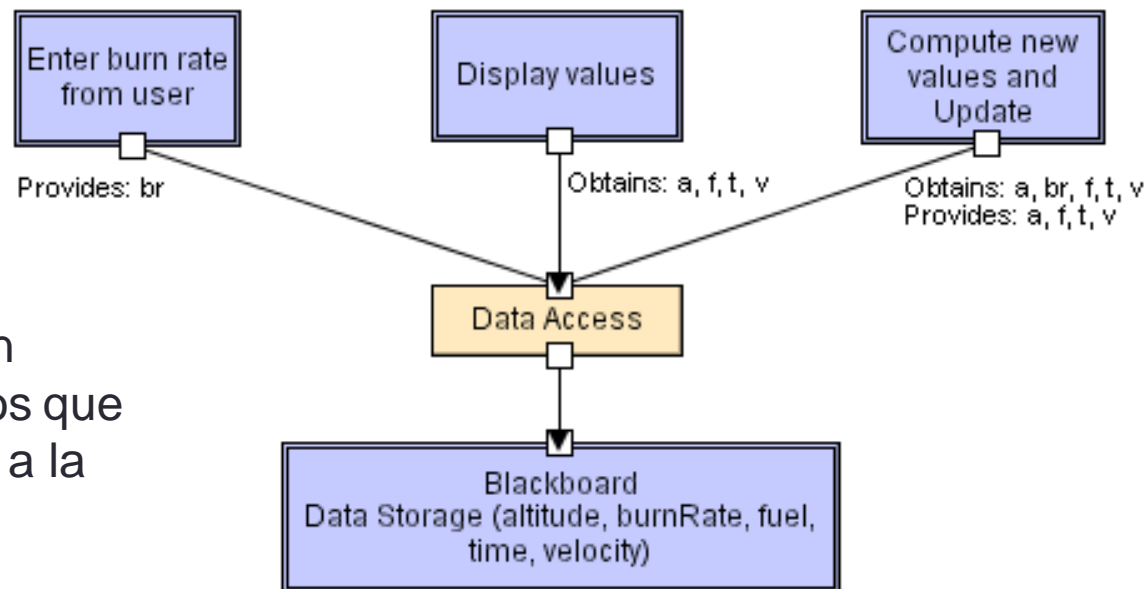
Data-Centered : Repository

- Componentes:
 - El almacén de datos central representa el estado
 - Los componentes independientes operan sobre el almacen central (data store)



Data-Centered : Blackboard

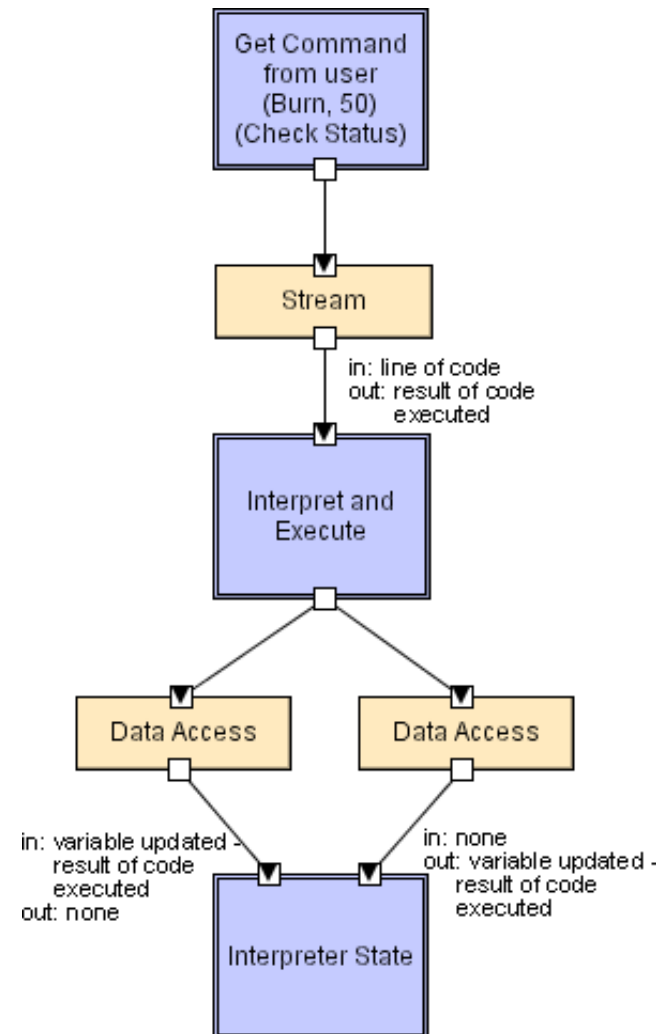
- Dos tipos de componentes
 - ✓ Estructura de datos central
 - pizarra
 - ✓ Componentes que operan en la pizarra
- El control del sistema está impulsado en su totalidad por el estado de la pizarra



- Fuentes de conocimiento hacen cambios a los datos compartidos que pueden conducir gradualmente a la solución

Virtual Machine: Interpreter

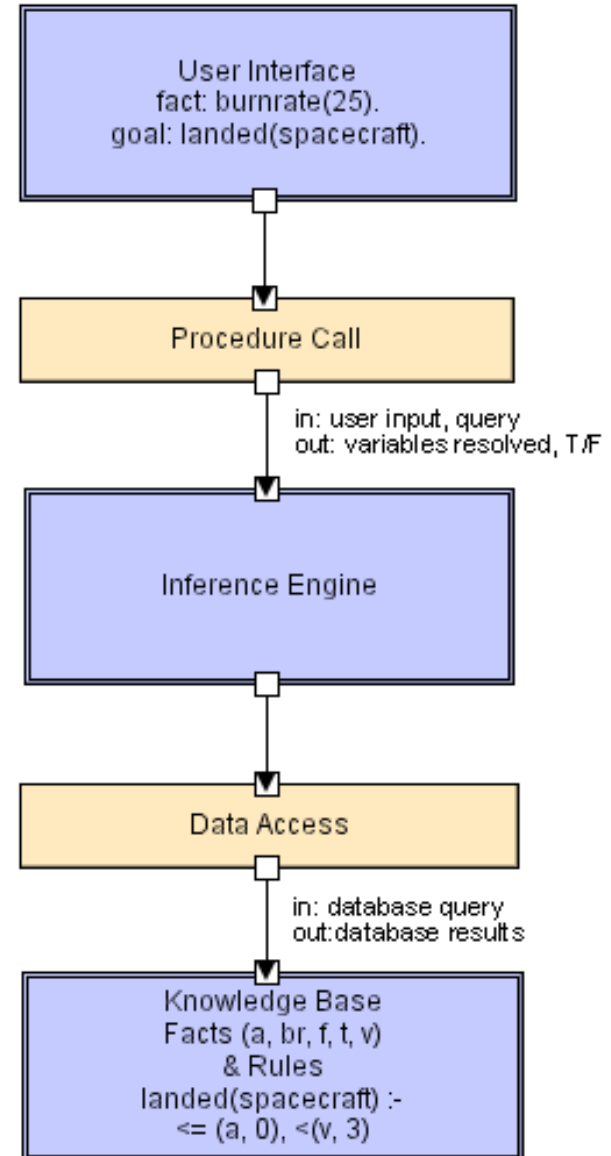
- Analiza y ejecuta comandos de entrada, actualizando el estado mantenido por el intérprete.
- Componentes: intérprete de comandos, estado del intérprete, interfaz de usuario.
- Conectores: Normalmente ligados con las llamadas a procedimientos directos y estado compartido.



Virtual Machine: Rule based

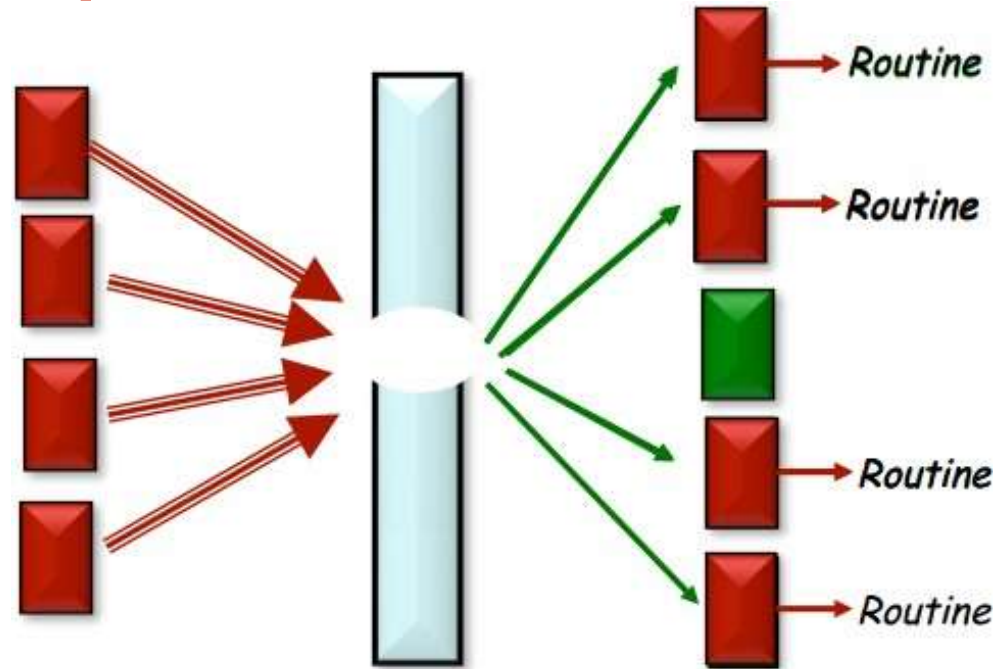
- Componentes: interfaz de usuario, motor de inferencia, base de conocimientos
- Conectores: Los componentes están estrechamente interconectados, con llamadas a procedimientos directos y/o memoria compartida.
- Elementos de datos: Datos y consultas

Cuando existe un gran número de reglas, la comprensión de las interacciones entre múltiples reglas afectadas por los mismos hechos puede llegar a ser muy difícil



Event-based, Implicit Invocation

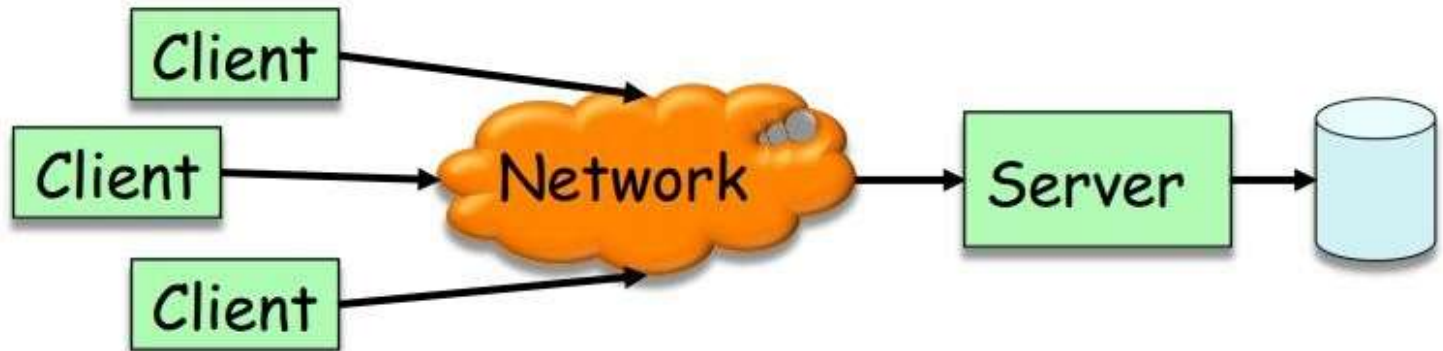
- Anuncio del evento en lugar de invocación de método
 - “Listeners” registran el interés y métodos asociados con eventos
- El sistema invoca todos los métodos registrados implícitamente
 - Componentes : son los métodos y eventos
 - Conectores: dos tipos dependiendo si la invocación es explícita o implícita en respuesta a los eventos



- Los componentes se comunican con el bus de eventos, no directamente el uno al otro
- Variantes: Push o Pull
- Altamente escalable y eficaz para aplicaciones altamente distribuidas

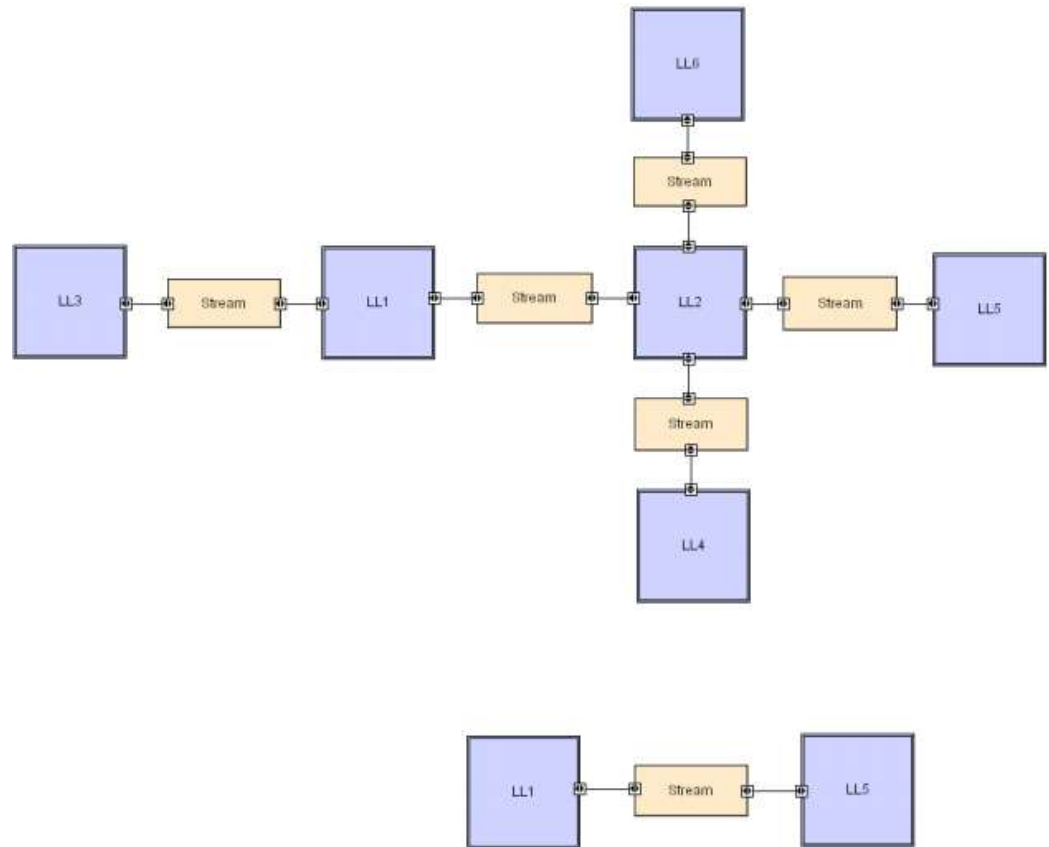
Client / Server

- Los componentes son los clientes y servidores
- Los servidores no saben la cantidad e identidad de los clientes
- Los clientes saben la identidad del servidor
- Los conectores son protocolos de interacción de red.



Peer-to-Peer

- Similar al estilo de cliente-servidor, pero cada componente es a la vez cliente y el servidor.
- Peer: componentes independientes, que tienen su propio control de estado.
- Conectores: Los protocolos de red, normalmente personalizados.
- Elementos de datos: Mensajes de red



Reflection

- Proporciona un mecanismo para cambiar la estructura y comportamiento de un sistema de software de manera dinámica.
- Soporta la modificación de aspectos fundamentales como los tipos de estructuras y mecanismos para invocación de funciones.
- En este patrón, la aplicación se divide en dos partes:
 - ✓ Un nivel “meta” que proporciona información sobre las propiedades
 - ✓ Un nivel “base” que incluye la lógica de la aplicación y cuya implementación se basa en el nivel “meta”.

Leer:

<http://wiki.ifs.hsr.ch/APF/files/Reflection.pdf>

Service Oriented Architecture (SOA)

- Una colección de servicios – funciones bien definidos, auto-contenidos y sin dependencias con otros servicios.

Repeatable task within a business process.

- Servicios compuestos para crear secuencias mayores e implementar procesos de negocio.
- Define para su operación un Enterprise Service Bus (ESB).
- Típicamente usa servicios Web usando WS-*, SOAP (XML).
- Alinea tecnología con negocio.

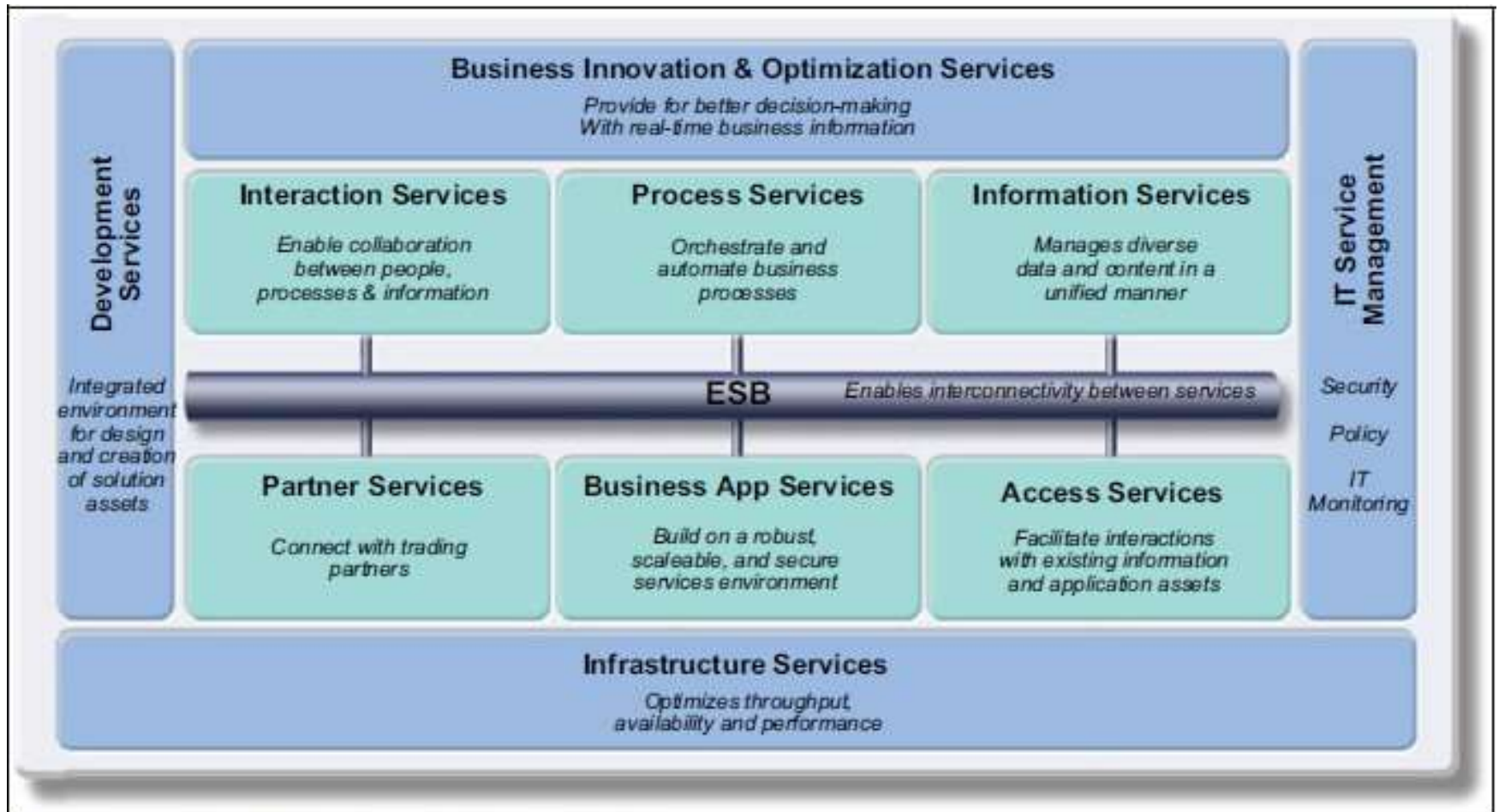


Figure 1-2 SOA reference architecture model

Service Oriented Architecture (SOA)

■ Ventajas:

- Mejora tiempo de implementación y ajustes a cambios de negocio.
- Aumenta reusabilidad.
- Desacopla componentes.
- Separación de aspectos.

■ Desventajas:

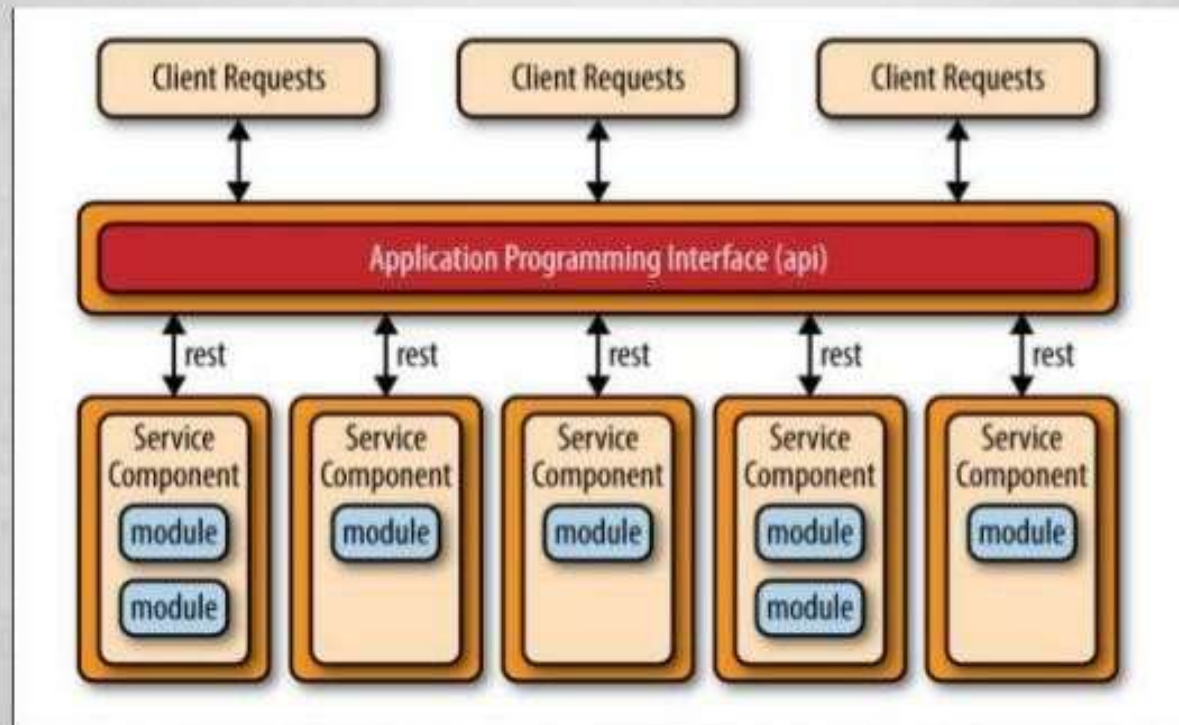
- Complejidad
- Requiere Governance para asegurar objetivos de negocio.

Web Oriented Architecture (WOA)

- Es un estilo de arquitectura de software que amplía SOA para aplicaciones basadas en Web. Es considerado como un subestilo de SOA.
- WOA tiene la finalidad de maximizar las interacciones entre el navegador (Browser) y el servidor (Server).
- Mecanismo que usa el Web (HTTP, URL).
- Ventajas:
 - Simple para implementar y consumir
 - Permite orquestación con BPEL (Business Process Execution Language) en el uso del Navegador (Browser).
- Desventajas:
 - No es tan flexible como SOA.



API REST-Based



MSDN:

Categoría	Estilo de Arquitectura	Definición
Comunicación	Message Bus	Un estilo de arquitectura que prescribe el uso de un sistema de software que puede recibir y enviar mensajes a través de uno o más canales de comunicación, por lo que las aplicaciones puedan interactuar sin necesidad de conocer los detalles específicos sobre la otra.
	Service-Oriented Architecture (SOA)	Se refiere a las aplicaciones que exponen y consumen funcionalidad como un servicio a través de contratos y mensajes.
Despliegue	Client/Server	Segrega el sistema en dos aplicaciones, donde el cliente hace peticiones al servidor. En muchos casos, el servidor es una base de datos con la lógica de la aplicación representada como procedimientos almacenados.
	N-Tier / 3-Tier	Segrega funcionalidad en segmentos separados de la misma manera que el estilo en capas, pero con cada segmento siendo un nivel situado en un equipo separado físicamente.
Dominio	Domain Driven Design	Un estilo arquitectónico orientado a objetos centra en modelar un dominio del negocio y la definición de los objetos de negocio basado en las entidades del dominio del negocio.
Estructura	Component-Based Architecture	Se descompone el diseño de aplicaciones en componentes funcionales o lógicos reutilizables que exponen bien definidas las interfaces de comunicación.
	Object-Oriented	Un paradigma de diseño basada en la división de responsabilidades para una aplicación o sistema en objetos reutilizables y autosuficientes individuales, cada uno con los datos y el comportamiento pertinentes al objeto.
	Layered Architecture	Particiones el dominio de interes de la aplicación en capas.

¿ Y estos en donde encajan ?

- Share Nothing
- Microservicios
- Contenedores
- Cloud



Patrones y Tácticas

Atributo de Calidad

Tácticas

	Modifiability									
	Increase Cohesion		Reduce Coupling					Defer Binding Time		
Pattern	Increase Semantic Coherence	Abstract Common Services	Encapsulate	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Startup-Time Binding	Use Runtime Binding
Layered	X	X	X		X	X	X			
Pipes and Filters	X		X		X	X			X	
Blackboard	X	X			X	X	X	X		X
Broker	X	X	X		X	X	X	X		
Model View Controller	X		X			X				X
Presentation Abstraction Control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							



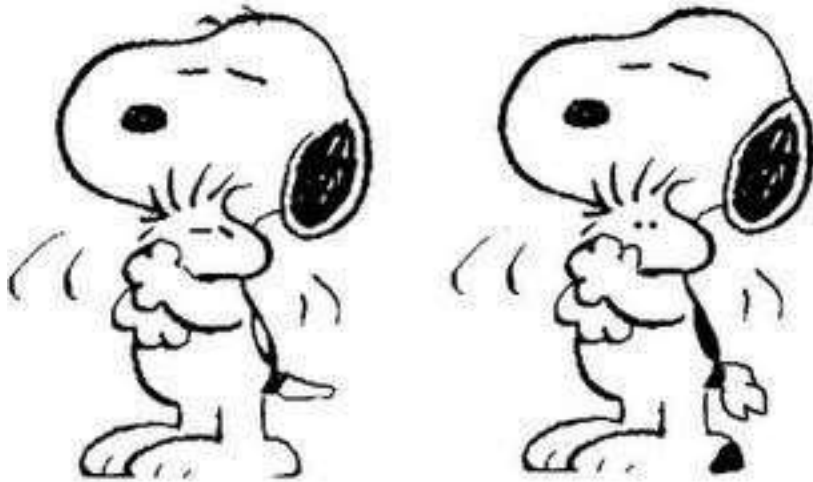


Architectural Patterns

- Un patrón de arquitectura expresa la organización fundamental de los esquemas estructurales de los sistemas de software.
- Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos.
- Los patrones de arquitectura son “plantillas” para las arquitecturas concretas.

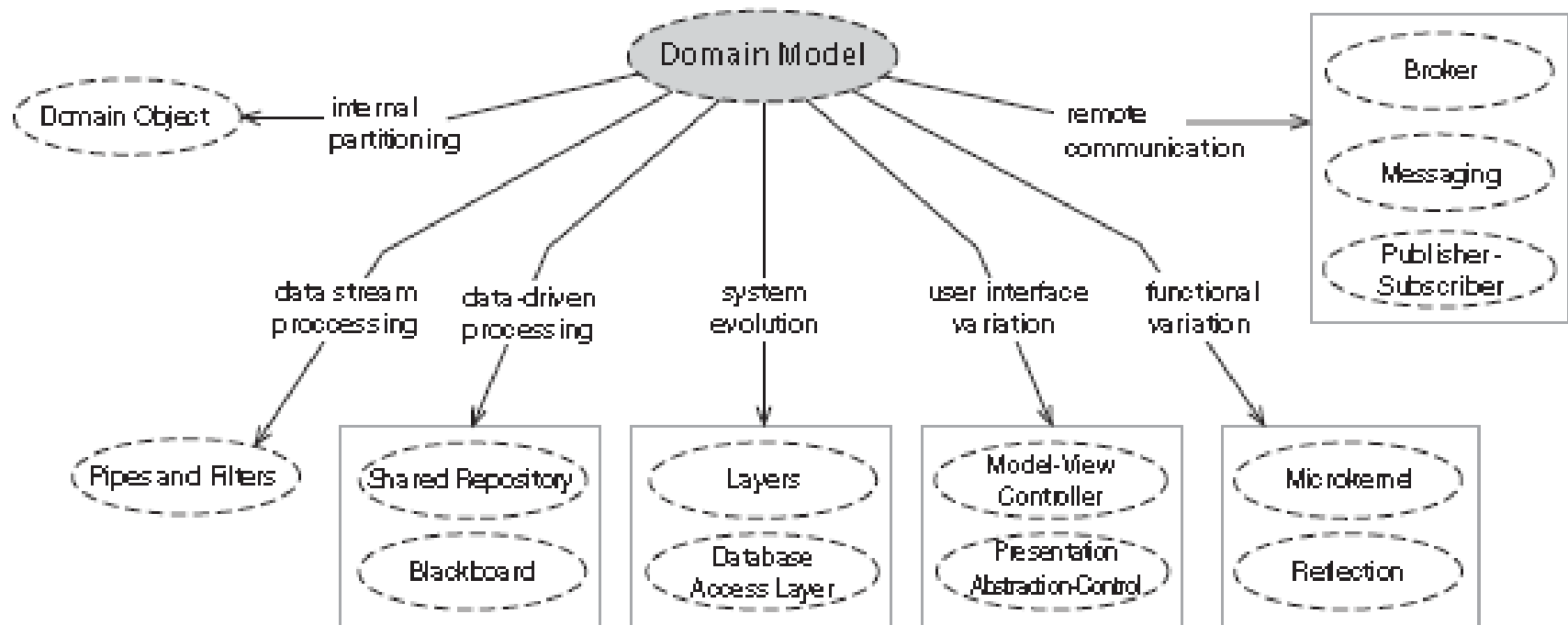
Diferencia entre Estilo y Patrón

La diferencia principal consiste en que un patrón puede ser visto como una solución a un problema, mientras que un estilo es más general y no requiere un problema para ser resuelto.



Diferentes patrones de arquitectura implican diferentes consecuencias, aun y cuando se dirijan a similares problemas.

The DOMAIN MODEL pattern defines a precise model for the structure and workflow of an application domain—including their variations. Model elements are abstractions meaningful in the application domain; their roles and interactions reflect domain workflow and map to system requirements.



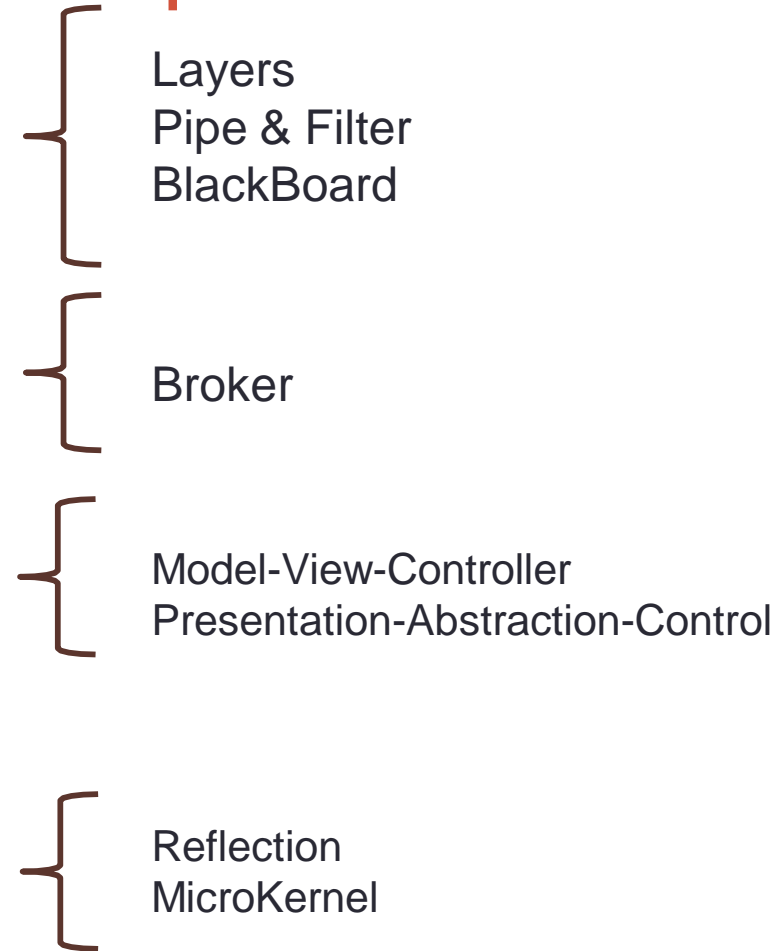
Clasificación: Patrones de Arquitectura

From Mud to Structure. Permiten evitar un océano de componentes u objetos. En particular soportan una descomposición controlada de una gran tarea en subtarefas cooperativas.

Distributed Systems: Proporciona la infraestructura completa para aplicaciones distribuidas

Interactive Systems. Soportan la estructura de sistemas que tiene interacción computadora-humano.

Adaptable Systems. Soportan la extensión de aplicaciones y su adaptación hacia tecnologías evolutivas así como cambios en requerimientos funcionales.



Layers
Pipe & Filter
BlackBoard

Broker

Model-View-Controller
Presentation-Abstraction-Control

Reflection
MicroKernel

Design Patterns

A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them.

It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context

The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem

Idioms

- Patrón de bajo nivel específico para un lenguaje de programación. Comprende aspectos de diseño e implementación.
- UN idioma describe como implementar aspectos particulares de componentes o las relaciones entre ellos utilizando las características de un lenguaje de programación
- Facilitan la comunicación entre programadores y aceleran el desarrollo y mantenimiento del software.
- Son menos “portables” entre diferentes lenguajes de programación.
- No basta con aprender la sintaxis de un lenguaje → se puede aprender trucos y reglas no escritas que permitan mejorar la calidad del código.