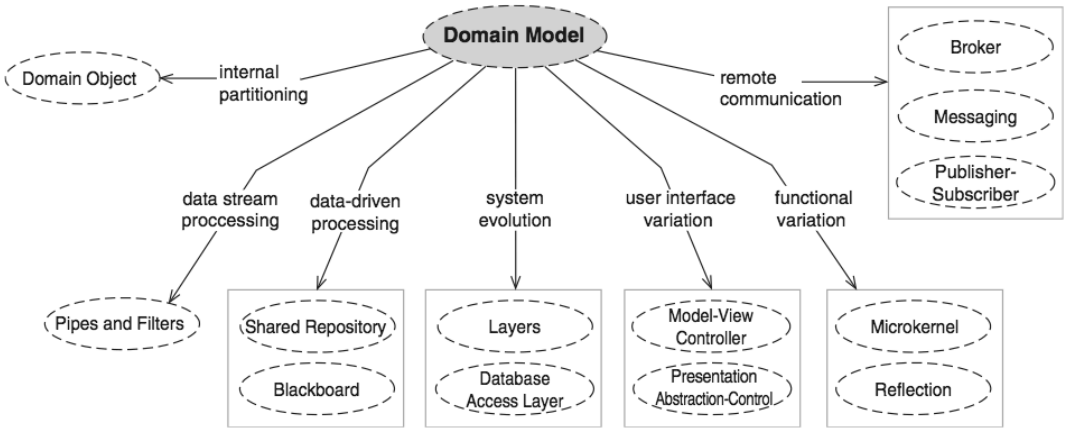
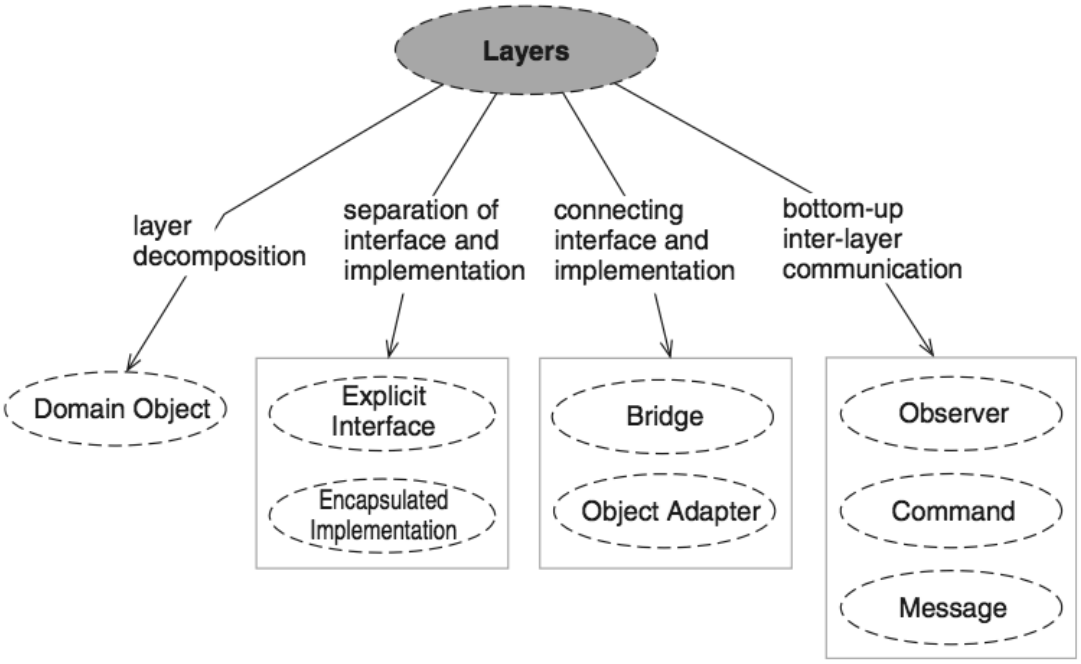
A Pattern Language for Software Architecture

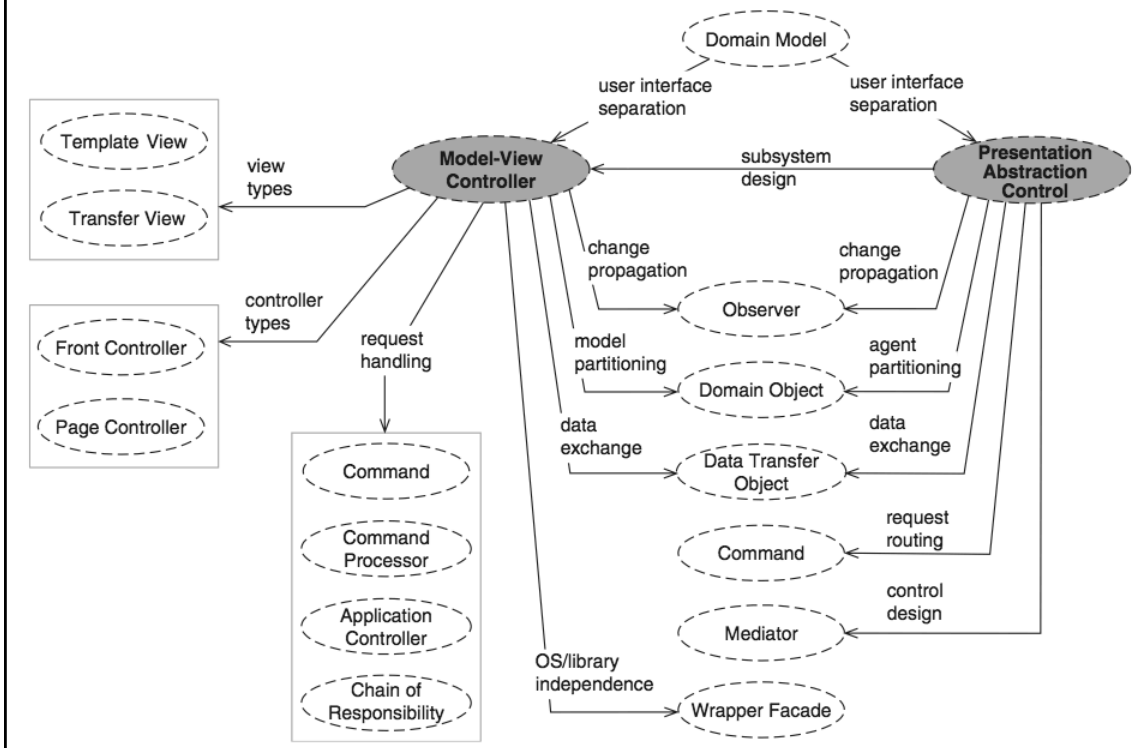


- The architectural patterns we have seen offer different solutions to the problem of how to organize a domain model into a software architecture.



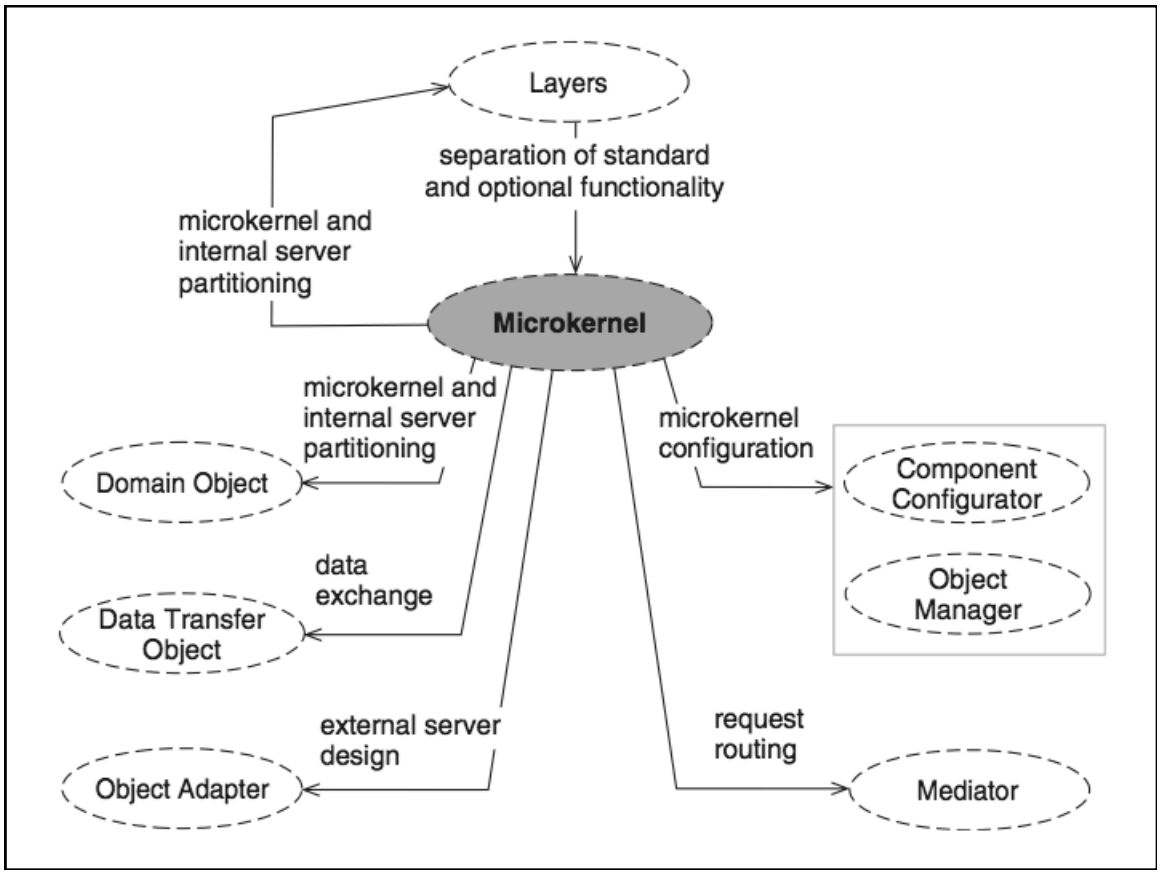
MVC vs. PAC

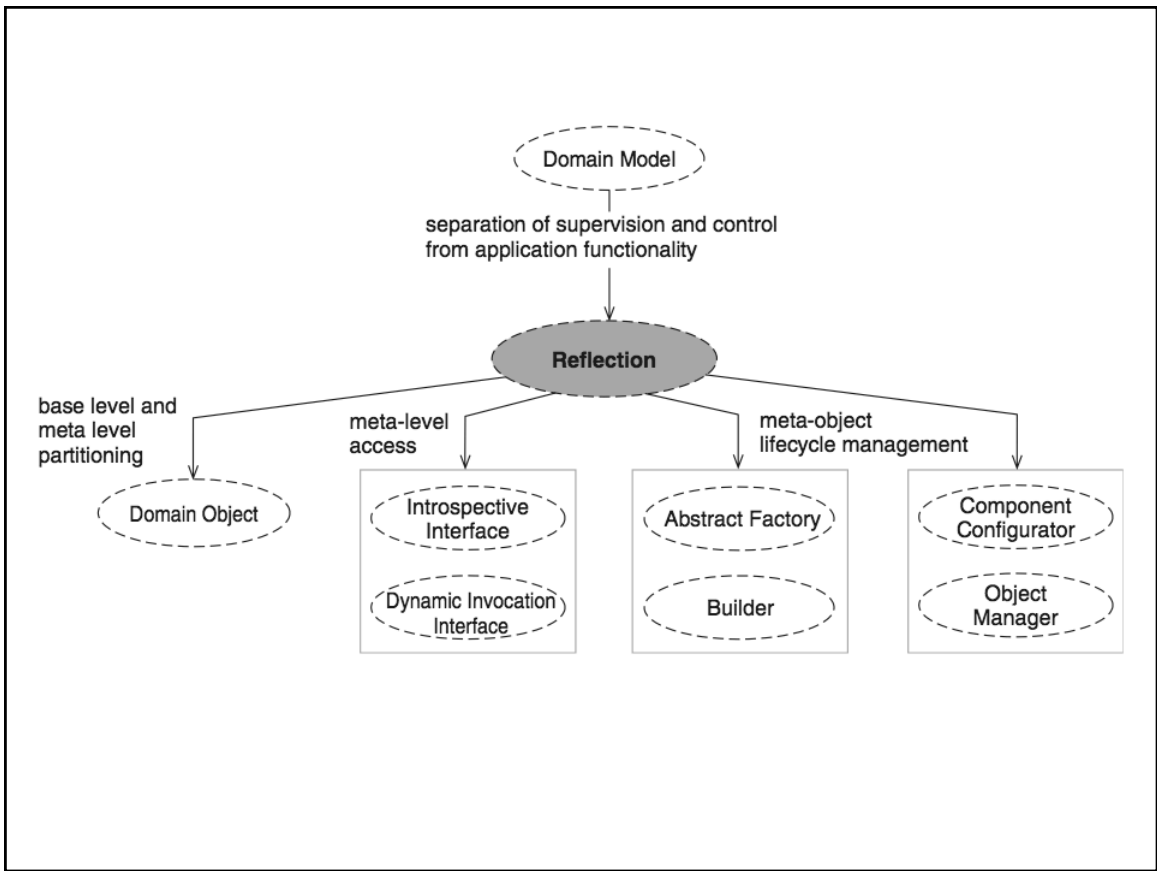
- Both Model-View-Controller and Presentation-Abstraction-Control address the problem of support for variability in user interfaces.
- Model-View-Controller supports variability within one specific user interface, while Presentation-Abstraction-Control supports the use of multiple, distinct user interfaces and their independent variation.
- As most software systems need only one user interface paradigm, Model-View-Controller should always be your first choice.
- Presentation-Abstraction-Control, in contrast, is only useful if a software system is partitioned into multiple, largely independent but sometimes cooperating subsystems, each of which suggests its own user interface paradigm.

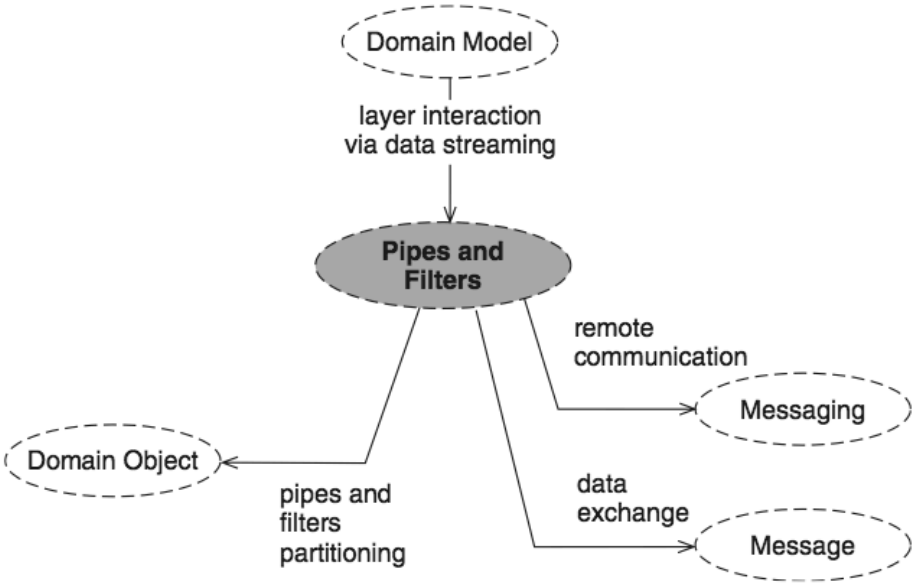


Microkernel and Reflection

- Microkernel and Reflection, both foster the construction of flexible software systems. Both patterns address different aspects of flexibility, however.
- Microkernel, in general, provides a plug-in architecture that supports flexibility in terms of what functionality a system provides to its users.
  - Microkernel has thus evolved as a popular architecture for operating systems, middleware, and product-line architectures.
- Reflection, in contrast, defines an architecture that objectifies specific aspects of a system's structure and behavior, which supports flexibility in terms of how its functionality executes and/or can be used by its clients.
  - It is thus often used in the context of application and service integration scenarios, in which client applications must be able to use or control the functionality of other applications without having an explicit, built-in knowledge of their interfaces and internal behavior.


Layers
separation of standard
and optional functionality
microkernel and
internal server
partitioning
Microkernel
microkernel and
internal server
partitioning
Domain Object
microkernel
configuration
Component
Configurator
Object
Manager
data
exchange
Data Transfer
Object
external server
design
Object Adapter
request
routing
Mediator

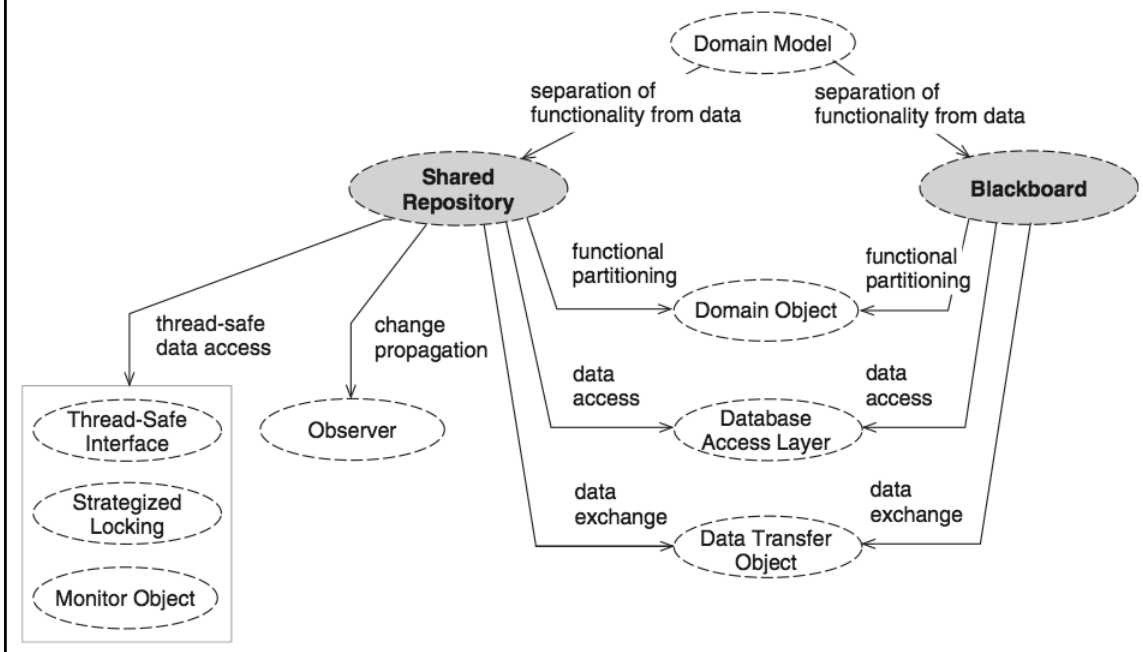

Domain Model
separation of supervision and control
from application functionality
Reflection
base level and
meta level
partitioning
Domain Object
meta-level
access
Introspective
Interface
Dynamic Invocation
Interface
meta-object
lifecycle management
Abstract Factory
Builder
Component
Configurator
Object
Manager



- The Pipes and Filters pattern is suited for applications that process data streams, or whose components communicate via the exchange of data streams.
  - Image processing is a prime example of a domain that can best be modeled in software via a Pipes and Filters architecture.

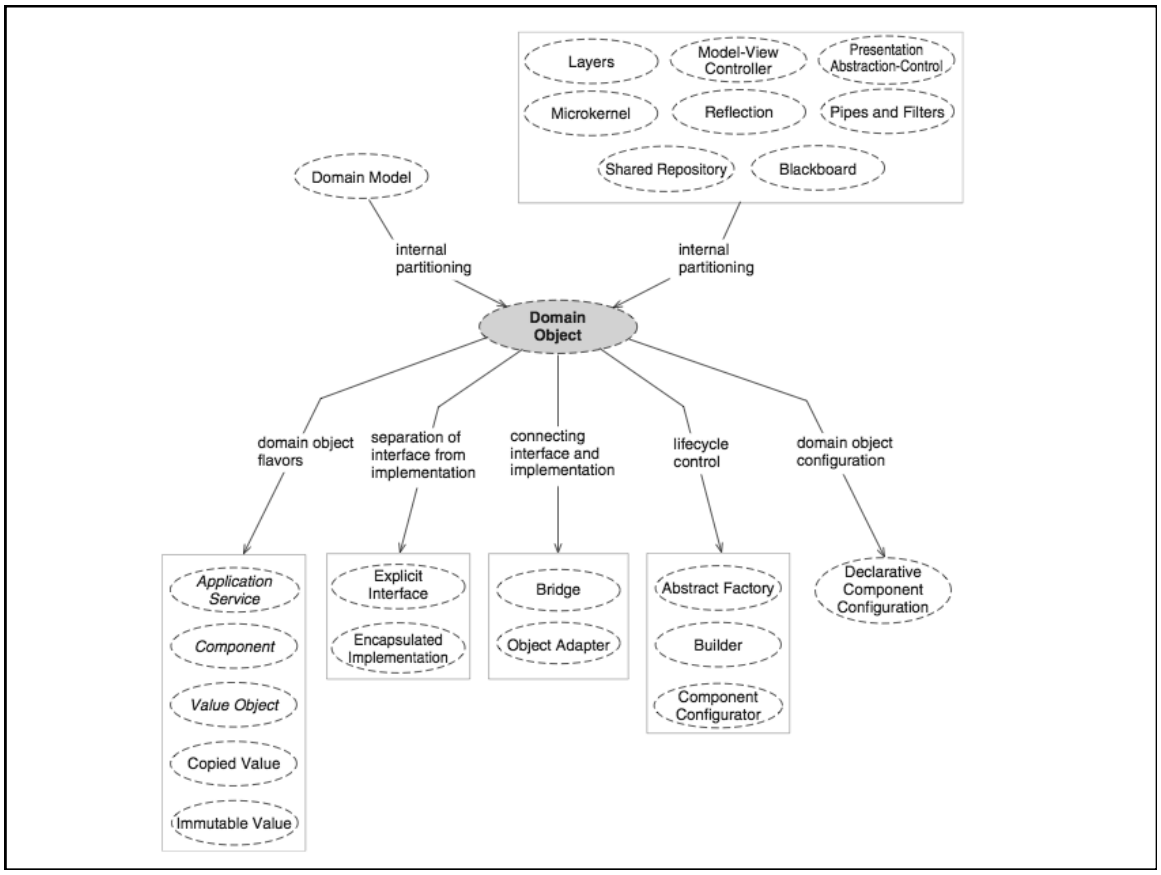
Shared Repository and Blackboard

- The Shared Repository and Blackboard patterns help in designing applications whose components work largely on a common set of (structured) data.
- By separating the data from the functionality of a system, data exchange between components of the application is simplified, and coordination of the components via Change of Value (CoV) notifications in that data becomes possible.



Domain Object Pattern

- The *Domain Object* pattern supports the encapsulation of self-contained responsibilities in an application within a defined software realization.
- Such encapsulation allows us to address the specific functional, operational, and developmental requirements of this responsibility explicitly, directly, and independently of other Domain Object realizations.


Layers
Model-View Controller
Presentation Abstraction-Control
Microkernel
Reflection
Pipes and Filters
Shared Repository
Blackboard
Domain Model
internal partitioning
internal partitioning
Domain Object
domain object flavors
separation of interface from implementation
connecting interface and implementation
lifecycle control
domain object configuration
Application Service
Component
Value Object
Copied Value
Immutable Value
Explicit Interface
Encapsulated Implementation
Bridge
Object Adapter
Abstract Factory
Builder
Component Configurator
Declarative Component Configuration