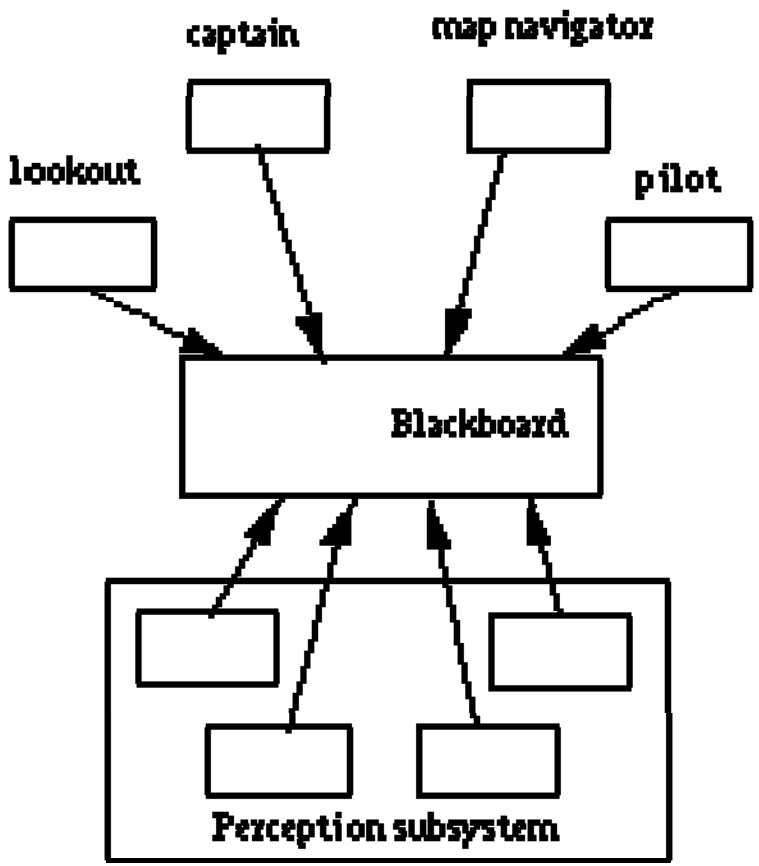
CPSC 875

John D. McGregor

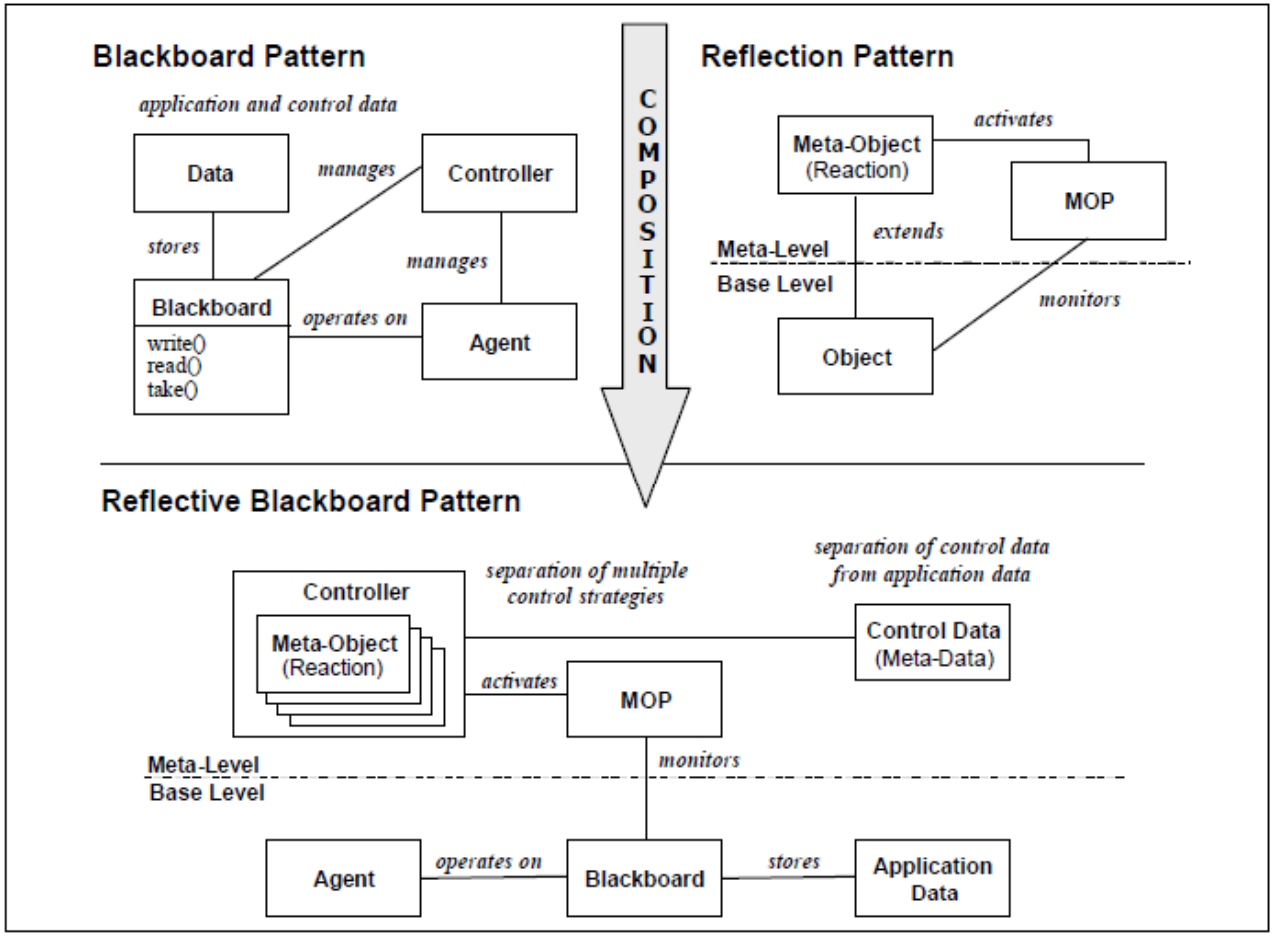
C9 - Tactics

- Query optimization
- eLearning
- Planning

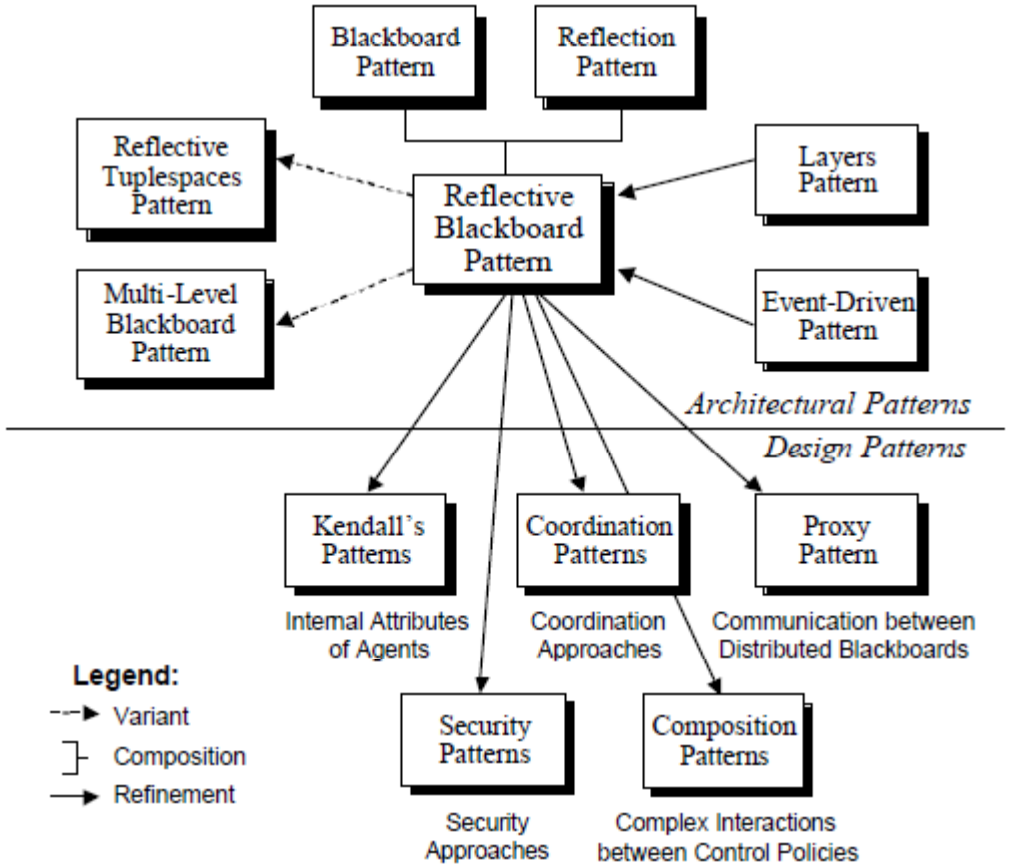
http://www.cs.cmu.edu/~ModProb/MRsol4.html



Consider a marketplace application where buyers and sellers negotiate products and services. Sellers advertise their desire to sell products or services, submitting offers to the marketplace. Buyers access the marketplace to submit bids in order to buy products and services, and simultaneously to find prospective sellers. Once the buyers have found an appropriate seller, they continue to communicate indirectly through the marketplace in order to negotiate and make proposals and counterproposals. Some buyers eventually join up with each other to buy products together and minimize costs. The marketplace is open, i.e. agents can join or leave it at any time, and agents are not initially aware of their counterparts. Buyers and sellers visit different marketplaces in the network in order to achieve their individual goals.

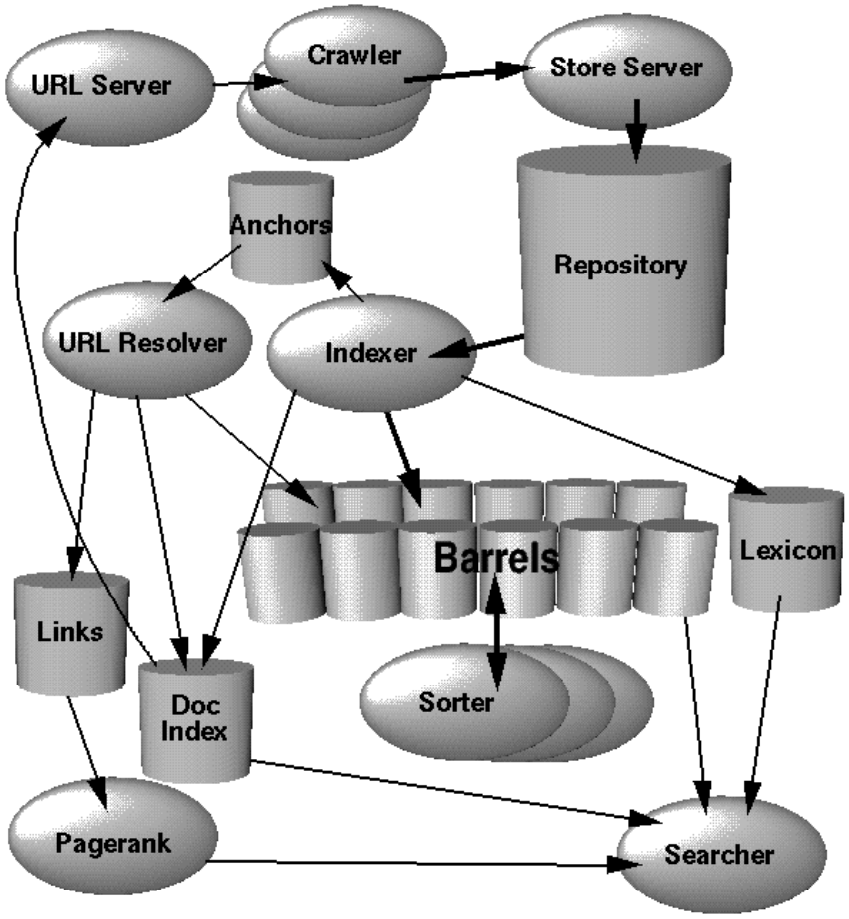


Doi: 10.1.1.4.9627

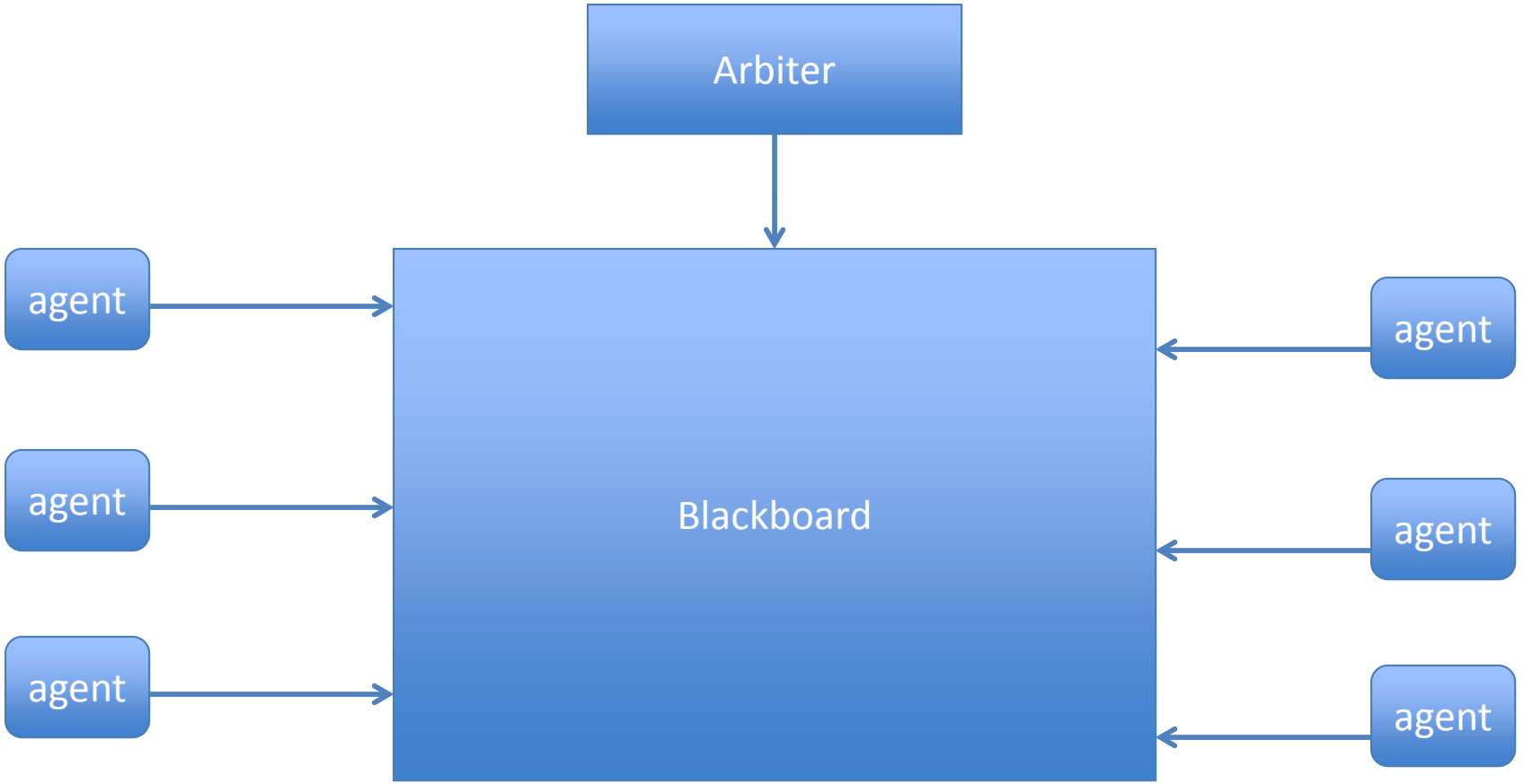


Doi: 10.1.1.4.9627

Google search engine



Blackboard architecture



Tactics

- A tactic is a transformation
- Given that the pre-condition of the tactic is true
- The tactic defines changes that should be made to the as-is architecture to get the will-be architecture
- The tactic description explains the corresponding changes to quality attributes

Modifiability tactics

Pattern	Modifiability									
	Increase Cohesion		Reduce coupling					Defer Binding Time		
	Maintain Semantic Coherence	Abstract Common Services	Use Encapsulation	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Start-Up Time Binding	Use Runtime Binding
Layers	X	X	X		X	X	X			
Pipe-and-Filter	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							

This and other diagrams from http://www.sei.cmu.edu/library/abstracts/reports/07tr002.cfm

Baldwin's Modularity Operators

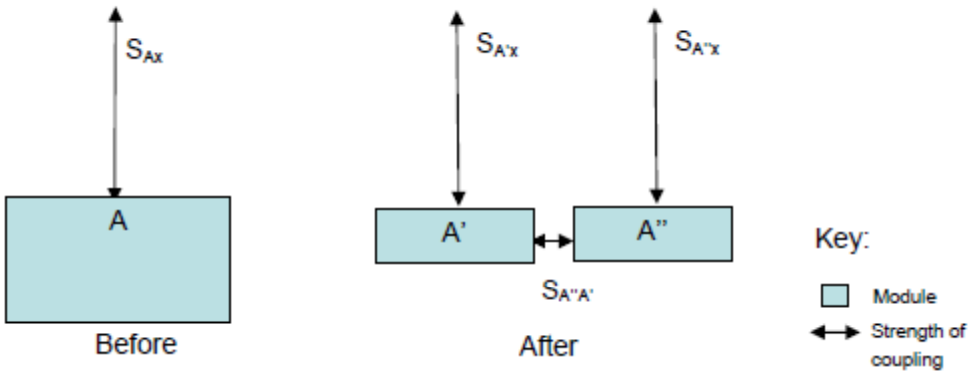
- Modularity reduces complexity and enhances maintainability
- Baldwin and Clark define 6 operators
- Any system
  - Splitting
  - Substitution
- Assumes a modular system
  - Augmenting
  - Excluding
  - Inversion
  - Porting

Splitting

- AKA decomposition
- A monolithic system or a module is divided into two or more modules
- Client/server is a split that enhances value by allowing multiple clients to access a single server – the assumption being that not all clients want to access the server at the same time

Splitting

- Reducing cost of modifying a single responsibility



Substitution

- AKA plug compatible
- One module is replaced by another with equivalent behavior but presumably a different implementation
- A desktop, laptop, and mobile device all have a bluetooth connection that obeys the bluetooth protocol but each has a different implementation; substituting will allow one system to be used on all three platforms but with a different driver

Augmenting

- An additional module is added to the system
- Perhaps a new type of communication connection such as USB is added to the system

Excluding

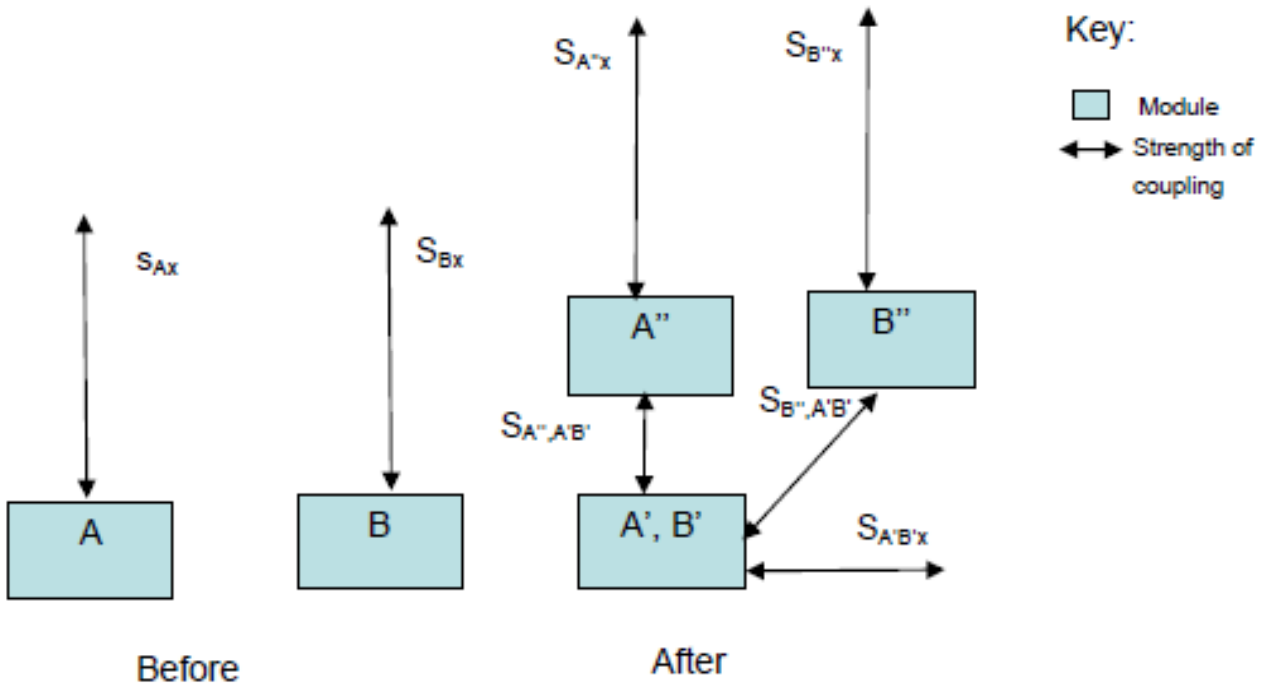
- A module is removed from the system.
- A generic software system may be tailored for a specific installation. The standard stereo module is excluded and the system is augmented with a surround sound module

Inversion

- Two or more modules are modified
- The result is a third module that captures the commonality among the initial modules
- A stereo sound system module and a surround sound module are analyzed and their common behavior made into a sound system module which is then related to the reduced stereo and surround sound modules
- Enhances the maintainability and extensibility

Inversion

- Increases cohesion



Porting

- A module is divided into a module that is more tightly coupled to the system under design and a module that is free from the single system
- Making a system easily used by multiple OSs is a typical example.
- Some new module may be needed in between the tightly coupled module and the free one

Encapsulation

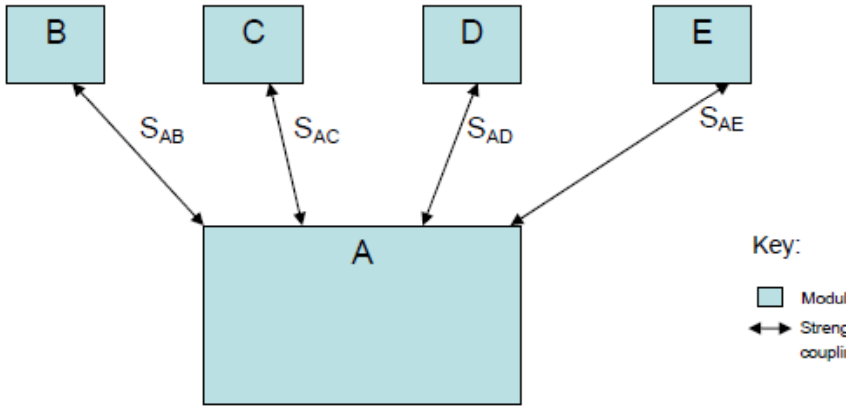
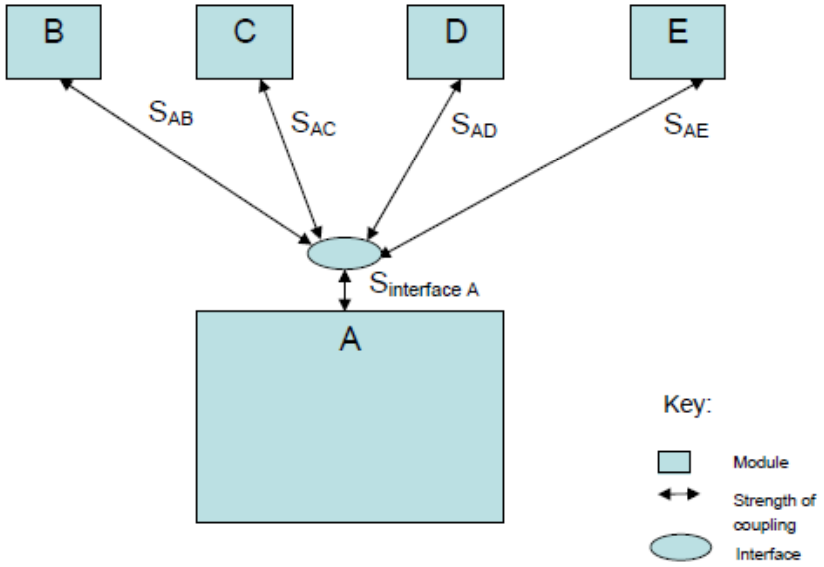
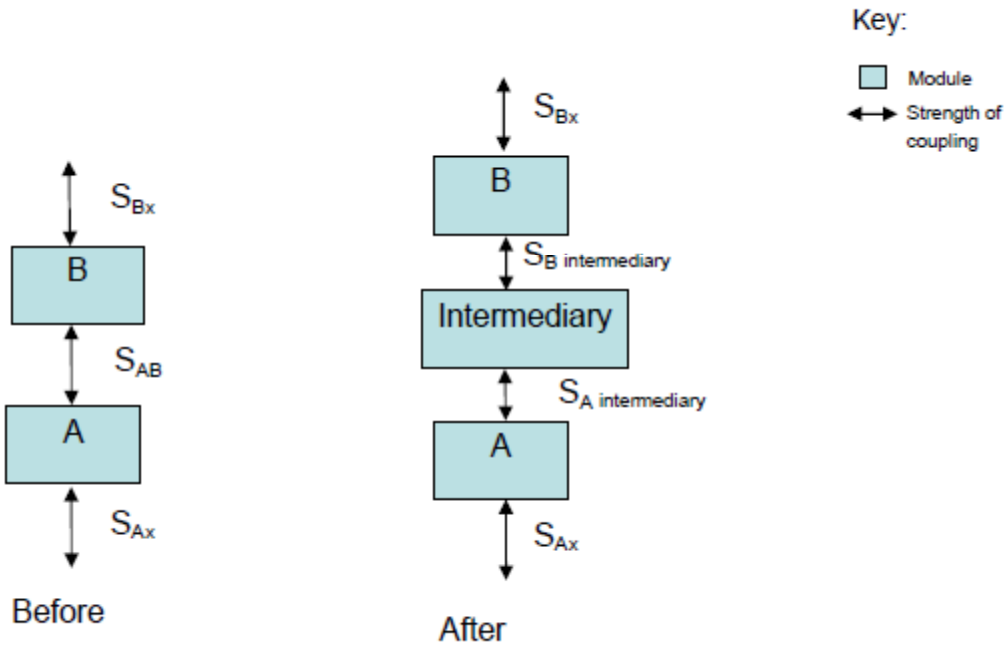


Figure 5: Modules Accessing Module A Prior to Encapsulating A

Figure 6 shows the situation after A has been encapsulated.



Intermediary



Blackboard

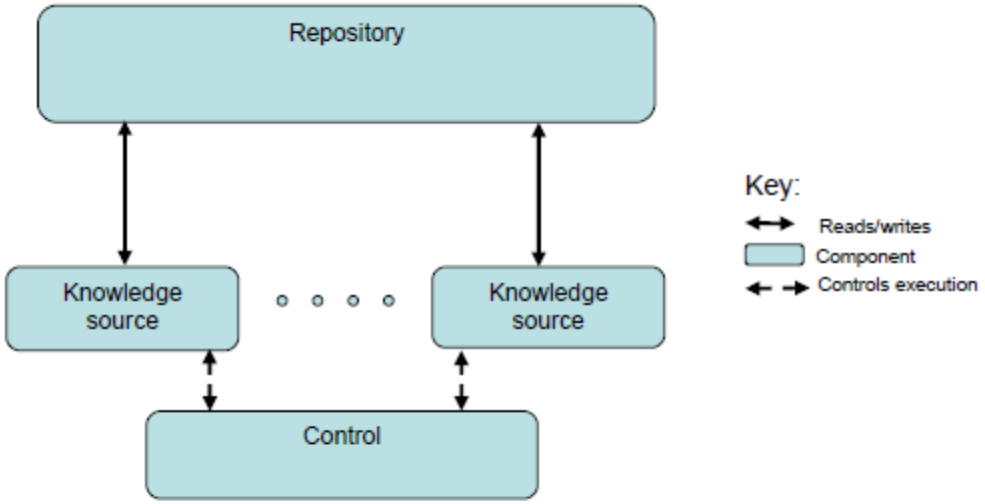


Figure 10: Blackboard Pattern Structure

Proxy

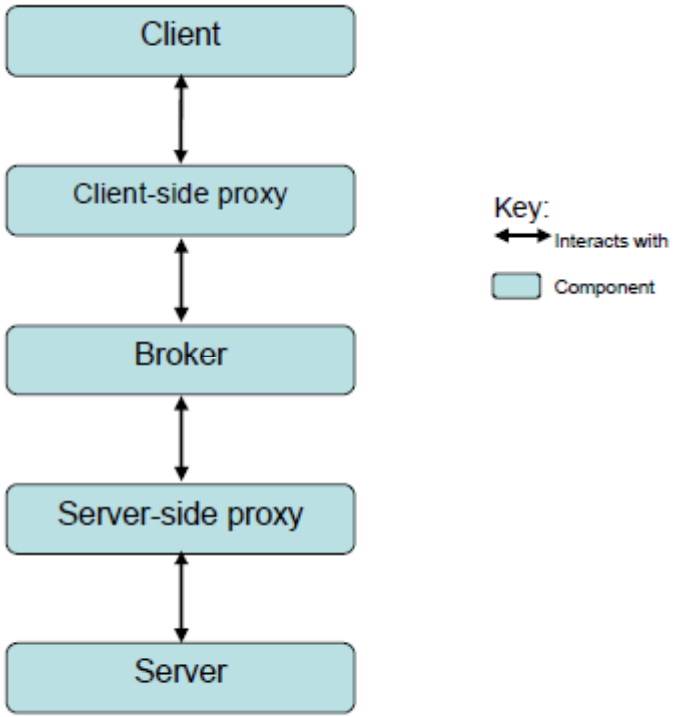


Figure 11: Broker Pattern Structure

Reflection

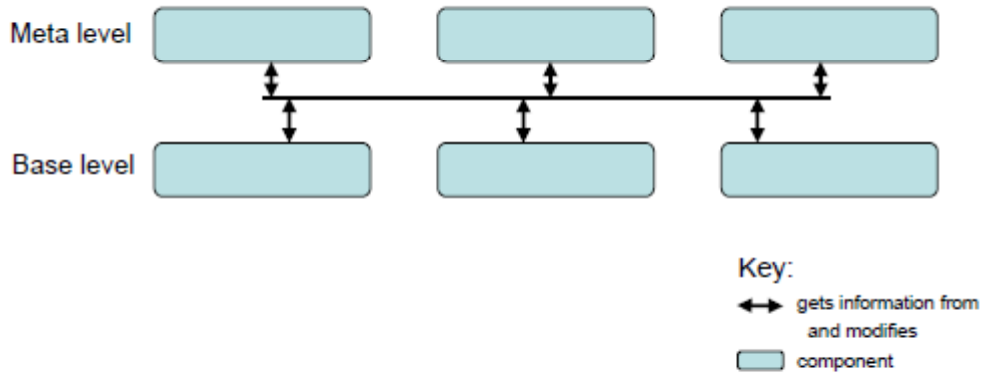


Figure 15: Reflection Pattern Structure

Modifiability tactics

Pattern	Modifiability									
	Increase Cohesion		Reduce coupling					Defer Binding Time		
	Maintain Semantic Coherence	Abstract Common Services	Use Encapsulation	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Start-Up Time Binding	Use Runtime Binding
Layers	X	X	X		X	X	X			
Pipe-and-Filter	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							

Layers via tactics

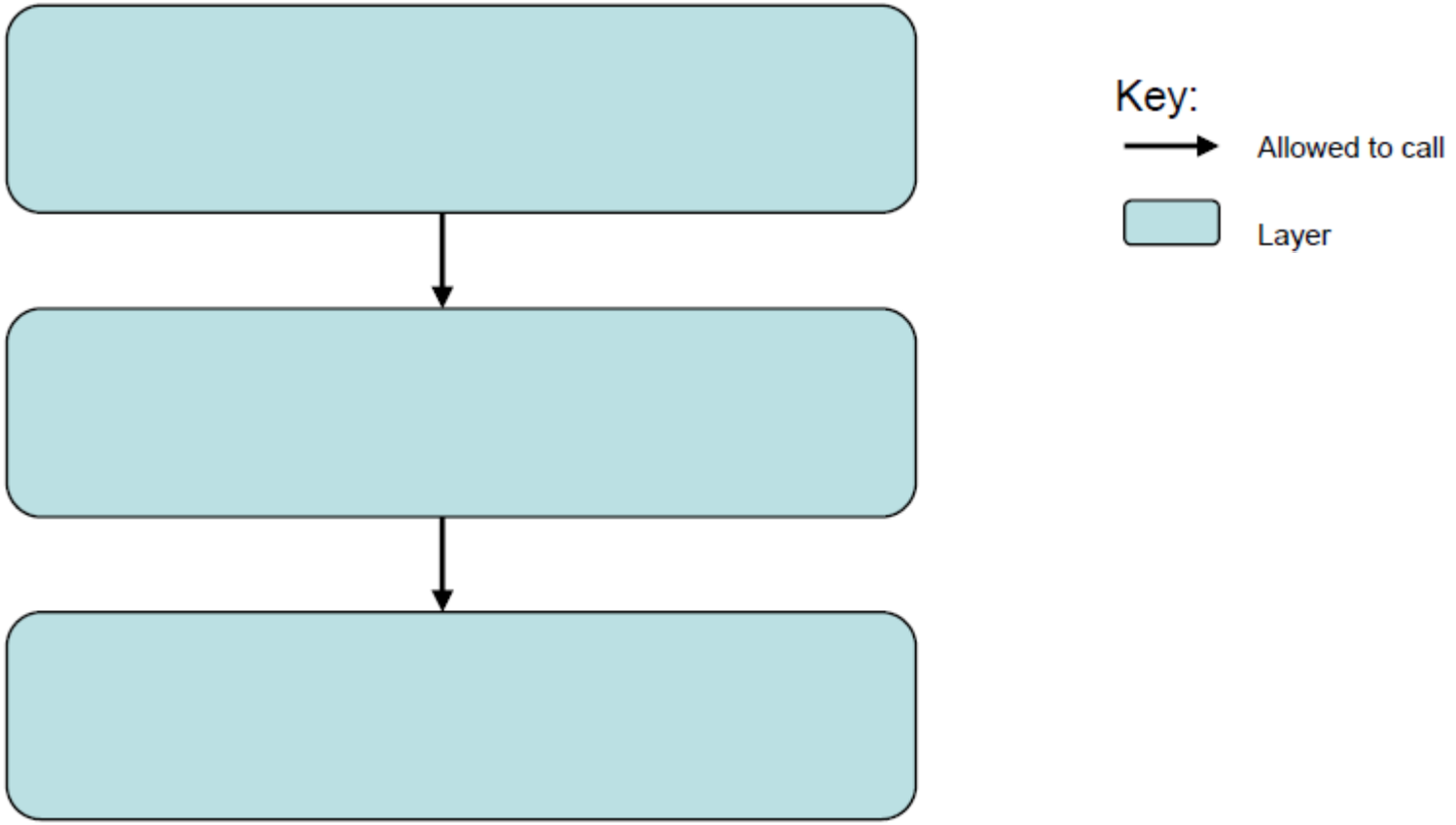


Figure 8: Layers Pattern Structure

Layers - 2

- **Maintain Semantic Coherence. The goal of ensuring that a layer's responsibilities all** work together without excessive reliance on other layers is achieved by choosing responsibilities that have some sort of semantic coherence.
- **Raise the Abstraction Level**. Layers represent an abstract ladder of services.

Layers - 3

- **Abstract Common Services.** Typically the responsibilities of a layer are grouped together into services.
- **Use Encapsulation.** There are two design considerations of the Layers pattern with respect to interfaces: (1) each layer may have its own interface and (2) particular layers may act as an interface (e.g., API, façade) for another layer.

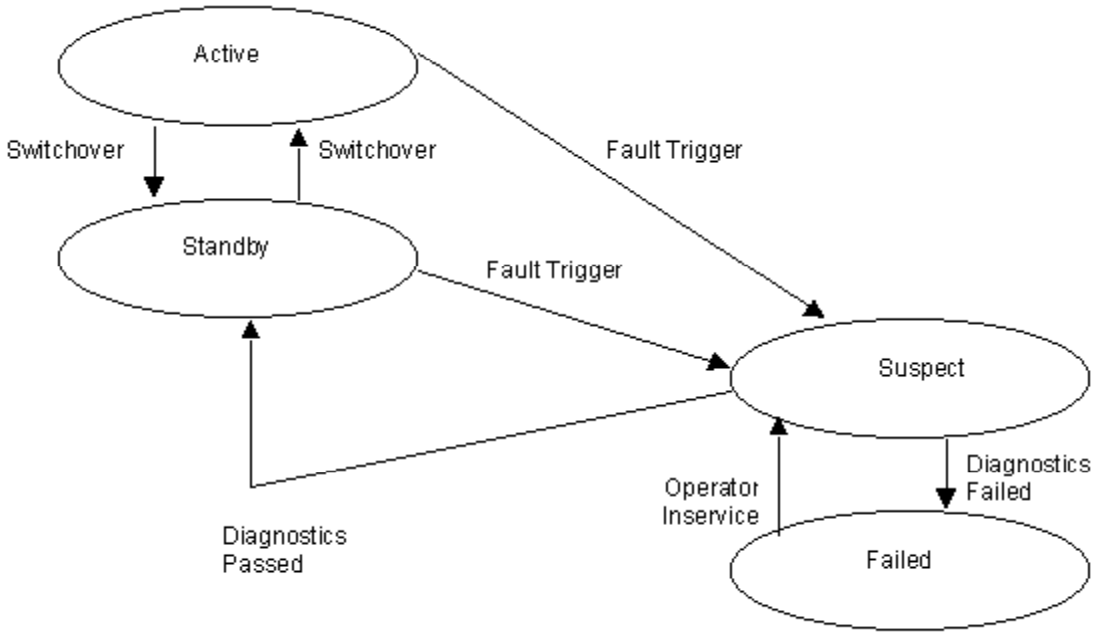
Layers - 4

- **Restrict Communication Paths.** Layers define an ordering and only allow a layer to use the services of its adjacent lower layer.
- **Use an Intermediary.** Particular layers may act as an interface (e.g., API, façade) for another layer.

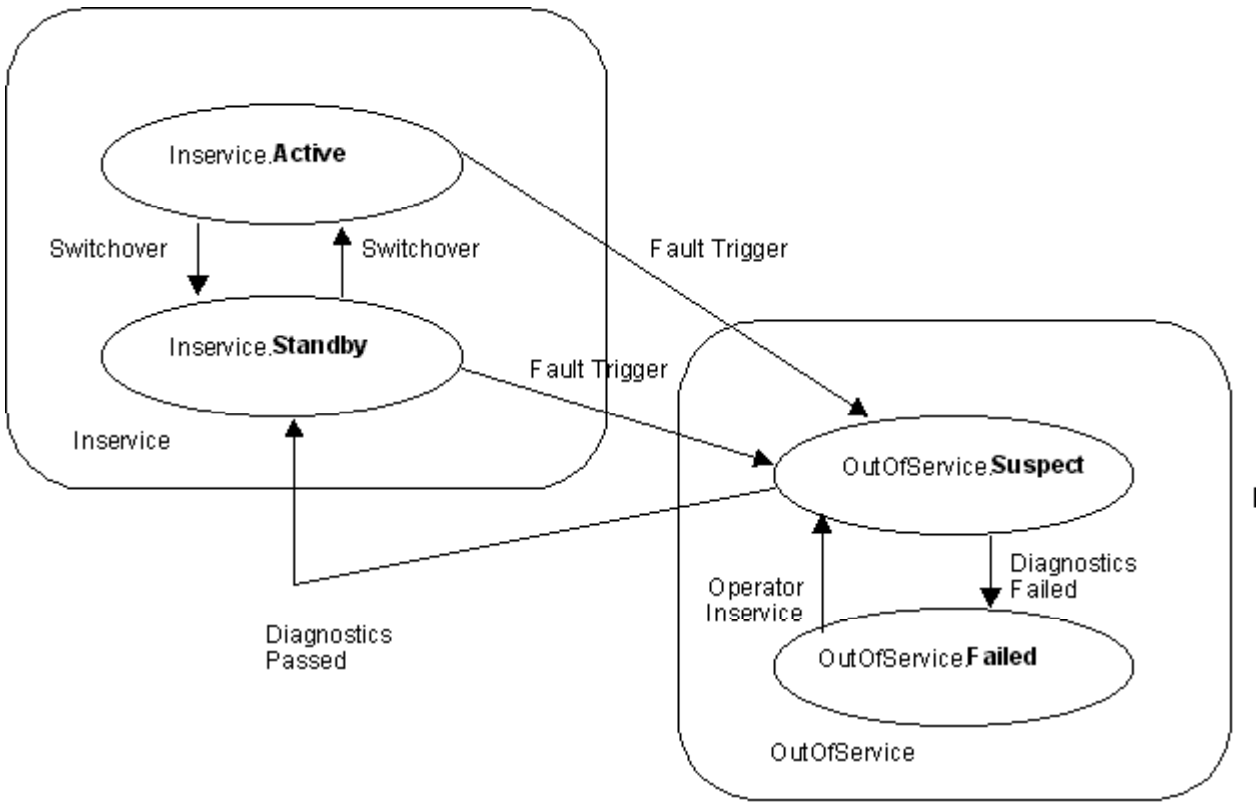
Layers - 5

- **Relaxed layered system.** A relaxed layered system is one in which layer N can invoke any layer below it rather than exclusively layer N-1, which is achieved by removing the Restrict Communication Paths tactic (i.e., removing an intermediary).
- **Layering through inheritance.** This variant refers to how the layers are packaged and, consequently, the binding time between them.

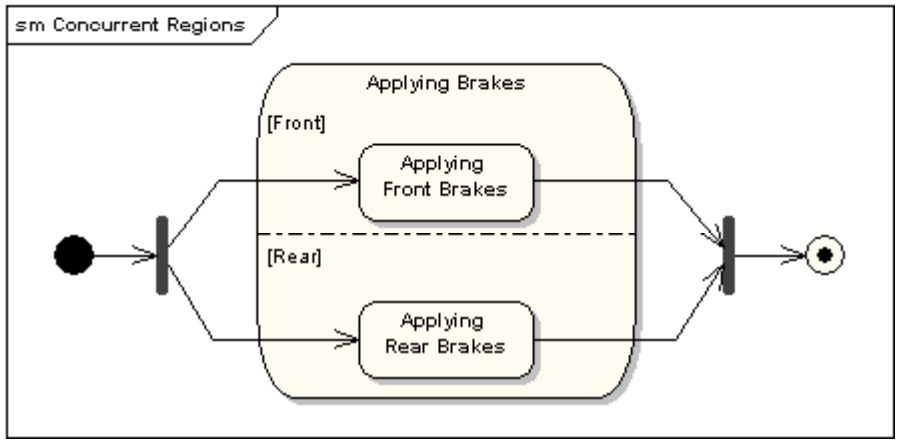
State Machines



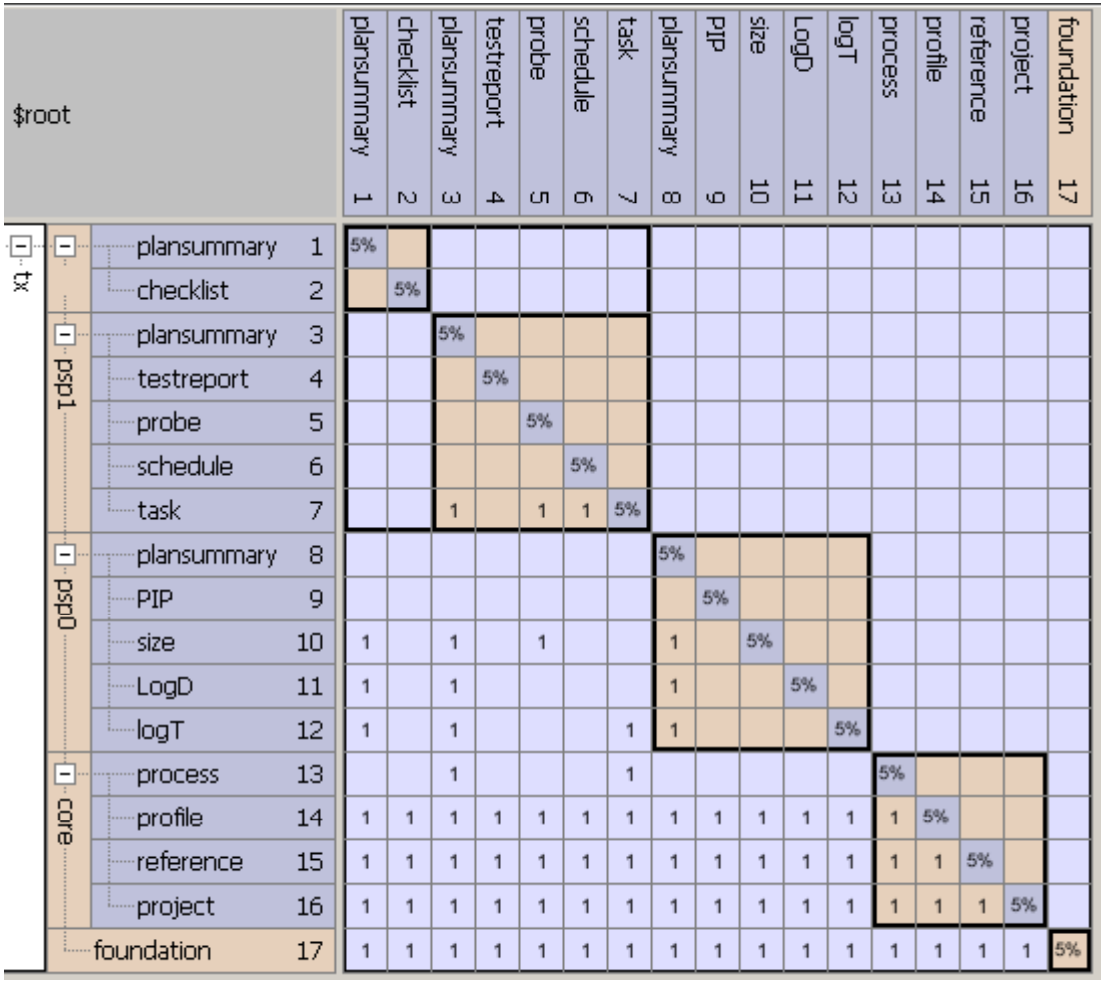
Hierarchical



Concurrent

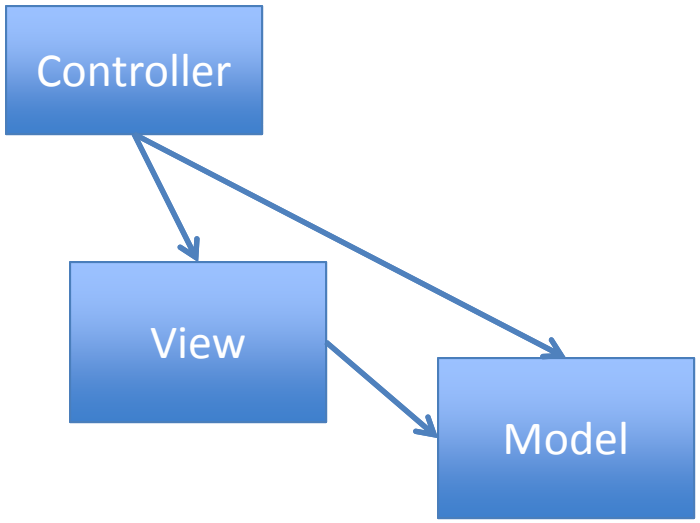

sm Concurrent Regions
Applying Brakes
[Front]
Applying
Front Brakes
[Rear]
Applying
Rear Brakes

Design Structure Matrix



MVC

	model	view	controller
model	x		
view	1	x	
controller	1	1	x



Reading

- Read this SEI tech report:

http://www.sei.cmu.edu/library/abstracts/reports/07tr002.cfm

- http://www.sparxsystems.com/resources/uml2_tutorial/uml2_statediagram.html