Assignment2

171250552

刘鹏程

2019 年 10 月 2 日

Availability

	Availability											
Pattern	Fault Detection			Fault Recovery						Fault Prevention		
	Ping/Echo	Heartbeat	Exception	Voting	Active redundancy	spare	Checkpoint/Rollback	Shadow Operation	Passive redundancy	Removal from service	Transaction	Process monitor
Layered	X		X	X	X	X	X	X	X		X	
Broker			X	X	X	X	X	X		X		
MVC	X		X	X	X	X	X		X		X	X
P2P			X			X		X	X	X	X	X
Master-Slave	X			X	X	X			X	X		X
Microkernel		X	X			X	X	X	X	X		X

Patterns' impact on availability

1. Layered:

a. Benefit: use multiple layers can use backup technologies within each layer to avoid the entire system from being unavailability because of an error in one layer
b. Penalty: if the connection components between layers fail, the system would be unavailable. But usually we will use reliable ways to implement layer connections.

2. Broker:

a. Benefit: the clients' failure will not the performance of server and the broker.
b. Penalty: both client and server depend on the broker. Once the broker fails, the system will be unavailable.

3. MVC:

a. Benefit: models\views\controllers can run on different environments to avoid a single component failure, which makes the system unavailable
b. Penalty: add complexity to the system and extend the recovery time

4. P2P (Peer to Peer)

a. Benefit: peers are equally privileged, a component could be easily replaced by another, thus promoting availability
b. Penalty: some bad designed p2p architecture may cause data loss. P2P also add complexity to data/service availability

5. Master-Slave

a. Benefit: an unavailable slave could be easily replaced.
b. Penalty: an unavailable master could be hard to handle with.

6. Microkernel

a. Benefit: failures of plug-ins could be easily tackled
b. Penalty: failures of kernel component are catastrophic

Tactics not associated with most patterns(discuss)

a. Heartbeat:
   Heartbeat detection requires periodic transmission of data and signal, which will largely influence the system performance. So this tactic is more suitable for systems with high reliability requirements, not suitable for system with high performance requirements.

Interoperability

	Interoperability		
Pattern	Locate	Manage Interfaces	
	Discover Service	Orchestrate	Tailor Interface
Layered	X	X	X
Broker	X	X	X
MVC	X	X	X

P2P	X		
Master-Slave		X	X
Microkernel	X		x

Patterns' impact on interoperability

1. Layered:

a. Benefit: different layers exchange information through pre-defined interfaces, each layer can process different kind of information, thus avoiding useless data and improving interoperability.
b. Penalty: add complexity to implement data filtering.

2. Broker:

a. Benefit: the agent acts as a middle component between client and server. The agent is able to interpret different kinds of information, which improves interoperability.
b. Penalty: information interpreting may increase the workload of the agent.

3. MVC:

a. Benefit: models\views\controllers can process different kinds of information on their own through pre-defined interfaces, which improves interoperability
b. Penalty: add complexity to implement data filtering

4. P2P (Peer to Peer)

a. Benefit: peers can design tailored interfaces to implement efficient connections which improve interoperability.
b. Penalty: peers may transfer meaningless information to each other which reduces interoperability.

5. Master-Slave

a. Benefit: the interfaces between master and slave are usually simple and specially tailored, which improves interoperability.
b. Penalty: master may have to handle multiple slaves, and if some slaves fail, they would introduce meaningless information to the master, which reduces interoperability.

6. Microkernel

a. Benefit: support tailored implementation of components through common interfaces, which may improve interoperability.
b. Penalty: may introduce meaningless information which reduces interoperability

No tactics not associated with most patterns

Performance

Pattern	Performance											
	Control Resource Demand						Manage Resources					
	Manage Sampling Demand	Limit Event Response	Prioritize Events	Reduce Overhead	Bound Execution Times	Increase Resource Efficiency	Increase Resources	Introduce Concurrency	Maintain Multiple Copies of Computations	Maintain Multiple Copies of Data	Bound Queue Sizes	Schedule Resources
Layered	X	X		X	X	X			X	X	X	
Broker	X		X		X	X	X	X	X	X		X
MVC		X		X			X	X	X	X	X	X
P2P	X	X		X	X	X	X	X	X	X	X	X
Master-Slave		X			X	X		X	X	X		X
Microkernel	X		X	X			X	X		X	X	

Patterns' impact on performance

1. Layered:

a. Benefit: the components within each layer work more efficiently as they are closed to each other, thus improving performance.
b. Penalty: as the layered architecture is an organized hierarchy, each layer provides service to the layer above and depends on the services provided by the layer below. So the interactions between layers may increase response time which reduces performance.

2. Broker:

a. Benefit: possibly reduce the time for the server to filter meaningless client request as the broker has bear this responsibility.
b. Penalty: the agent/broker itself may be a performance bottleneck and the penalty outweighs benefit.

3. MVC:

a. Benefit: MVC is better than layered in that it requires fewer layer calls, thus improving performance.
b. Penalty: a view may have to make multiple calls to obtain all its display data, so caching data within views is a problem for response time, thus reducing performance.

4. P2P (Peer to Peer)

a. Benefit: if peers scale is large, P2P in many cases will improve data transmission efficiency because the p2p doesn't require a center processor, thus improving performance.
b. Penalty: if peers scale is not large enough, introducing P2P would probably add complexity to the design, and the cost to make such improvement in performance would not be worth it.

5. Master-Slave

a. Benefit: when dealing with the requirement to store and process large amounts of data, master-slave pattern has proven to be reliable and efficient as it distributes data and computing resources to many slave nodes. Obviously, this pattern improves performance.
b. Penalty: applying such pattern to deal with small amount of data is not worthwhile.

6. Microkernel

a. Benefit: each part/component can work on their own without influencing other components, which improves performance.
b. Penalty: increase the workload of the kernel module, thus reducing performance.

Tactics not associated with most patterns

a. Prioritize events
   Prioritizing events is essential for system dealing with multiple events at the same time. In patterns require highly parallel computing, prioritizing events is not that important.

Security

	Security			
	Detect Attacks	**Resist Attacks**	**React to Attacks**	**Recover from**

Pattern																Attacks	
	Detect Intrusion	Detect Service Denial	Verify Message Integrity	Detect Message Delay	Identify Actors	Authenticate Actors	Authorize Actors	Limit Access	Limit Exposure	Encrypt Data	Separate Entities	Change Default Settings	Revoke Access	Lock Computer	Inform Actors	Maintain Audit Trail	Restore
Layered	X		X	X	X	X		X	X			X	X	X	X		X
Broker	X	X					X			X	X					X	
MVC		X	X	X		X					X	X	X	X	X		X
P2P		X	X	X	X	X	X	X	X	X					X	X	
Master-Slave		X	X		X				X		X	X		X	X		X
Microkernel	X			X		X	X	X			X		X	X		X	X

Patterns' impact on security

1. Layered:

a. Benefit: dividing the system into many layers enables the system to distribute logic and data into different layers, thus limiting the exposure of the core part of the system, which improves security.
b. Penalty: add complexity to the system, reduce the performance. Some information on the upper layers may not be secure enough.

2. Broker:

a. Benefit: the agent acts as a central role to deal with attacks. It works efficiently to reject the similar malicious requests from some suspicious groups during a period of time.
b. Penalty: agent/client/server all have to guard against illegal attacks, thus introducing complexity and uncertainty to the system, which is unsecure.

3. MVC:

a. Benefit: The components in the model module can be distributed to limit the damage and promote recovery.
b. Penalty: add complexity to the system.

4. P2P (Peer to Peer)

a. Benefit: basic attack detection and resistance settings can be easily configured to deal with simple connections and requests.
b. Penalty: due to the complexity and uncertainty of different peers, efficient and tailored security method are hard to guarantee.

5. Master-Slave

a. Benefit: it's relatively easy for a master-slave architecture to implement data redundancy, which promotes recovery from attacks.
b. Penalty: usually the slaves are simple machines doing jobs assigned by the masters, so it's hard for slaves to prevent malicious attack.

6. Microkernel

a. Benefit: since most services/plug-ins run in user mode, the security could be ensured by the kernel.
b. Penalty: the security mechanism within each plug-in may have to be tailored, thus intruding uncertainty and complexity, which reduces security.

Tactics not associated with most patterns

a. Encrypt data
As data has been encrypted well through the Internet, we don't have to do extra encryption work. However, in some cases requiring high security, extra data encryption has to be considered to avoid some special malicious attacks.

Testability

	Testability							
Pattern	**Control and Observe System State**						**Limit Complexity**	
	Specialized Interfaces	Record/Playback	Sandboxing	Abstract Data Sources	Executable Assertion	Localize State Storage	Limit Structural Complexity	Limit Nondeterminism
Layered	X	X	X	X			X	X
Broker	X	X			X			X

MVC	X		X	X	X	X	X	X
P2P			X		X	X		X
Master-Slave	X	X	X			X	X	
Microkernel	X		X	X	X		X	X

Patterns' impact on testability

1. Layered:

a. Benefit: different layers can be tested independently using tools like Stub and Driver, which improves testability.
b. Penalty: integration testing may be hard due to the unpredictable combined effects of different layers.

2. Broker:

a. Benefit: the broker alone is relatively easy to test offline which improves testability.
b. Penalty: as the broker provides large amounts of interfaces to deal with different servers and clients. Meanwhile, it's hard to isolate an instance of the broker system from the real world because the broker usually requires high availability, thus reducing testability.

3. MVC:

a. Benefit: different modules can be tested independently which improves testability.
b. Penalty: the connections between module are difficult to test because of its inherent complexity.

4. P2P (Peer to Peer)

a. Benefit: in some cases, the behaviors of peers could be limited and predictable, which improves testability.
b. Penalty: in most of the cases, different peers tend to act quite differently from each other, the peers' behaviors are unpredictable, which adds complexity to the test work.

5. Master-Slave

a. Benefit: the interfaces between master-slave/master-master are usually well defined and simple, which improves testability.
b. Penalty: because the slave nodes failure should be taken into consideration in the real world, we have to add failure recovery test to see whether the system can behave normally and reliably under extreme circumstances, which reduces testability.

6. Microkernel

a. Benefit: since the different modules could be tested independently, the microkernel has high testability.

b. Penalty: the integration test may be hard to conduct because of different implementation of modules.

Tactics not associated with most patterns

a. Record/Playback
Recording the state that caused a fault and re-creating the fault is costly and consumes processing and storage resources. From my perspective, this tactic could be applied to test work when other quality attributes such as stability, reliability, availability are required

Usability

	Usability						
Pattern	Support User Initiative				Support System Initiative		
	Cancel	Undo	Pause/Resume	Aggregate	Maintain Task Model	Maintain User Model	Maintain System Model
Layered	X	X	X	X		X	X
Broker		X		X	X		
MVC	X	X	X	X	X	X	X
P2P	X	X			X		X
Master-Slave		X	X			X	X
Microkernel	X			X	X	X	X

Patterns’ impact on testability

1. Layered:

a. Benefit: Users will only need to know the interfaces provided by the top layers to use most of the functions provided by the system, which improves usability.
b. Penalty: in some cases requiring users’ comprehensive knowledge about the whole system, it’s hard for users to make full use of the features hidden in the deep layers, which

reduces usability.

2. Broker:

a. Benefit: the clients and the servers will only need to know the interfaces provided by the broker, which increases usability.
b. Penalty: no obvious features about broker pattern would reduce usability.

3. MVC:

a. Benefit: the user will only need to understand the interfaces provided by the view module. The controller and the model module are transparent to the user, which improves usability.
b. Penalty: when something goes wrong with the model and controller module, it's difficult for the user to locate the problem. However, regular users usually care little about where to find the hidden bugs.

4. P2P (Peer to Peer)

a. Benefit: for P2P systems providing simple connections and services between peers, it's easy for users to use the P2P interfaces.
b. Penalty: P2P systems with large scale usually provide complex interfaces and services which require expert knowledge, thus reducing usability.

5. Master-Slave

a. Benefit: the slaves can be automatically configured by the master. Meanwhile, the user may be able to pause and resume the process manually, which improves usability.
b. Penalty: the cluster configuration may be troublesome and time consuming, which reduces usability.

6. Microkernel

a. Benefit: the interfaces provided by the kernel are simple and easy to use, outside components will only have to implement the simple interfaces to provide services, thus improving usability.
b. Penalty: the concrete implementation of the outside components differs a lot from each other, which makes it hard to have a thorough understanding of these components, thus reducing usability.

Tactics not associated with most patterns

a. Pause/resume
   This tactic is suitable for some certain situations where users want to take control of a long running process, which means the users could pause or resume the process anytime they want. However, in many cases, there's no need for stopping the process for a while. Therefore, this tactic is not associated with most patterns in the matrix.