Nanjing University

Software Systems Design & Architecture

Assignment 2

In Software Engineering Institute

by 161250096 Yu Pan

Mail: panyuyuyu@outlook.com

Introduction

The survey mainly focused on eight classical design patterns and their impacts on quality attributes. Obviously, object-oriented design patterns are used to improve the quality of the software. Unfortunately, patterns in practice do not always improve quality attributes, thus providing evidence against common lore. The results can be seen in the following table.

Availability	**Detect Faults**									**Recover from Faults**														**Prevent Faults**				
Pattern	Ping/ Echo	Monitor	Heartbeat	Timestamp	Sanity Checking	Condition Monitoring	Voting	Exception Detection	Self-Test	Active Redundancy	Passive Redundancy	Spare	Exception Handling	Rollback	Software Upgrade	Retry	Ignore Faulty Behavior	Degradation	Reconfiguration	Shadow	State Resynchronization	Escalating Restart	Non-Stop Forwarding	Removal from Service	Transactions	Predictive Model	Exception Prevention	Increase Competence Set
Layered									×	×	×			×					×		×							
Pipes and Filters				×					×										×							×		×
Blackboard		×					×		×		×	×	×			×								×				
Broker		×							×		×	×	×	×							×							×
Model View Controller		×					×		×	×	×			×				×	×			×							
Presentation Abstraction Control		×					×		×	×	×			×					×			×			×			
Microkernel		×												×					×	×		×			×			
Reflection	×						×										×				×				×			

Availability

Pattern	Benefits	Penalties
Layered	Supports fault tolerance and graceful undo	
Pipes and Filters	\	Error handling is a problem
Blackboard	Neutral: Single point of failure, but can duplicate it	Neutral
Broker	Neutral: Single point of failure mitigated by duplication	Neutral
Model View Controller	Neutral	Neutral
Presentation Abstraction Control	Neutral	Neutral
Microkernel	Supports duplication and fault tolerance.	\
Reflection	\	Protocol robustness is necessary

Interoperability	**Locate**	**Manage Interfaces**	
Pattern	**Discover Service**	**Orchestrate**	**Tailor Interface**
Layered	×		
Pipes and Filters	×		×
Blackboard			
Broker			
Model View Controller			×
Presentation Abstraction Control			
Microkernel			
Reflection		×	×

Interoperability

Pattern	Benefits	Penalties
Layered	\	Propagation of calls through layers can be inefficient
Pipes and Filters	One can exploit parallel processing	Time and space to copy data
Blackboard	\	Hard to support parallelism
Broker	Neutral	Neutral
Model View Controller	\	Inefficiency of data access in view
Presentation Abstraction Control	\	High overhead among agents
Microkernel	\	High overhead
Reflection	\	Meta- object protocols often inefficient

Performance	**Control Resource Demand**						**Manage Resources**					
Pattern	**Manage Sampling Rate**	**Limit Event Response**	**Prioritize Events**	**Reduce Overhead**	**Bound Execution Times**	**Increase Resource Efficiency**	**Increase Resources**	**Introduce Concurrency**	**Maintain Multiple Copies of Computation**	**Maintain Multiple Copies of Data**	**Bound Queue Sizes**	**Schedule Resources**
Layered		×	×		×	×	×	×				
Pipes and Filters			×		×	×		×	×		×	
Blackboard	×	×			×	×	×	×				×
Broker	×	×	×		×	×		×		×		
Model View Controller			×			×	X		×		×	×
Presentation Abstraction Control			×			×	×					
Microkernel	×	×	×		×	×				×		
Reflection			×						×			×

Performance

Pattern	Benefits	Penalties
Layered	\	Propagation of calls through layers can be inefficient
Pipes and Filters	One can exploit parallel processing	Time and space to copy data
Blackboard	\	Hard to support parallelism
Broker	Neutral	Neutral
Model View Controller	\	Inefficiency of data access in view
Presentation Abstraction Control	\	High overhead among agents
Microkernel	\	High overhead
Reflection	\	Meta- object protocols often inefficient

Security	**Detect Attacks**				**Resist Attacks**							**React to Attacks**			**Recover from Attacks**	
Pattern	**Detect Instruction**	**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**	**See Availability**
Layered		×			×	×	×										
Pipes and Filters					×	×			×								×
Blackboard		×			×	×	×	×		×			×		×		×
Broker		×	×		×	×	×	×					×		×		
Model View Controller						×				×							
Presentation Abstraction Control						×	×										
Microkernel		×	×		×	×									×		×
Reflection			×												×		×

Security

Pattern	Benefits	Penalties
Layered	Support layers of access	\
Pipes and Filters	\	Each filter needs its own security
Blackboard	Supports access control	Independent agents may be vulnerable
Broker	Supports access control	
Model View Controller	Neutral	Neutral
Presentation Abstraction Control	Neutral	Neutral
Microkernel	Neutral	Neutral
Reflection	Neutral	Neutral

Testability	**Control and Observe System State**						**Limit Complexity**	
Pattern	**Specialized Interfaces**	**Record/ Playback**	**Localize State Storage**	**Abstract Data Sources**	**Sandbox**	**Executable Assertions**	**Limit Nondeterminism**	**Limit Structural Complexity**
Layered	×		×		×	× ×		
Pipes and Filters	×	×	×		×			×
Blackboard	×	×			×	×	×	×
Broker	×	×	×		×		×	
Model View Controller	×		×			×		×
Presentation Abstraction Control	×		×			× ×		
Microkernel	×	×	×		×	×		
Reflection	×		×					

Testability

Pattern	Benefits	Penalties
Layered	The independent module can be tested	Can be difficult to get the layers right and test
Pipes and Filters	\	Parallel processing tests are required

Blackboard	\	Difficult to design test effectively, high development effort
Broker	Can often base functionality on existing services and easy to test	\
Model View Controller	Modules can be tested independently	Complex structure to test
Presentation Abstraction Control	Modules can be tested independently	Complexity; difficult to get atomic semantic concepts right
Microkernel	\	Very complex design and implementation to test
Reflection	Easy to test and structure itself is simple	\

Usability	**Support User Initiative**						**Support System Initiative**	
Pattern	**Cancel**	**Undo**	**Pause/ Resume**	**Aggregate**	**Maintain Task Model**	**Maintain User Model**	**Maintain System Model**	**Limit Structural Complexity**
Layered		×	×		×		×	
Pipes and Filters	×	×	×		×	×	×	×
Blackboard		×		×	×		×	×
Broker		×	×	×	×		×	
Model View Controller		×	×			×	×	×
Presentation Abstraction Control		×	×	×			×	
Microkernel	×	×	×		×		×	
Reflection	×	×	×			×	×	

Usability

Pattern	Benefits	Penalties
Layered	Neutral	Neutral
Pipes and Filters	\	Generally not interactive
Blackboard	Neutral	Neutral
Broker	Location Transparency	\
Model View Controller	Synchronized views	\
Presentation Abstraction Control	Semantic separation	\
Microkernel	Neutral	Neutral
Reflection	Neutral	Neutral

Reference

1. **https://www.rug.nl/research/portal/files/14565482/Chapter%203**