SERVICE LOOSE COUPLING

62

62

Motivation

- Software maintenance and reuse
- Fault tolerance and scalable execution
- Governance and trust
  - Cf Sony Rootkit
- Avoid technological lock-in
  - IT economics: suppliers want lock-in!
- Decoupled Contract

63

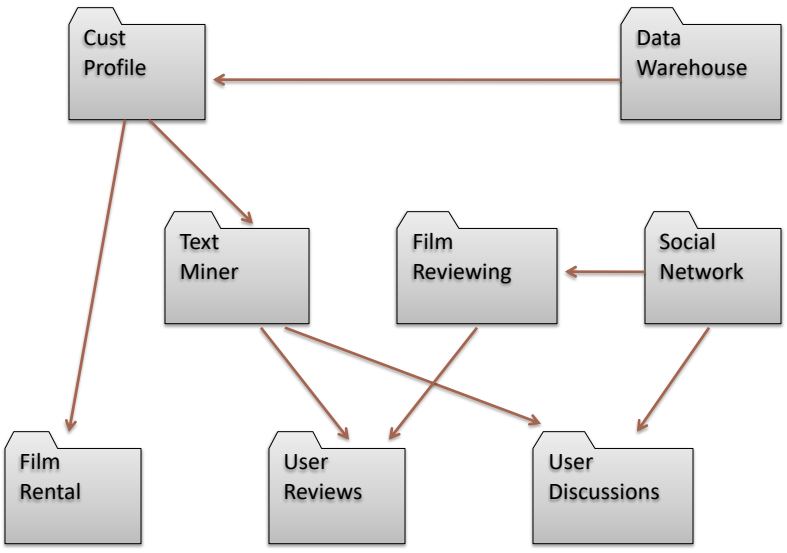
63

Loose Coupling Patterns

- Key patterns: Service Façade
  - Encapsulate and abstract internal services
  - Coarsen grain of interaction

64

64



65

65

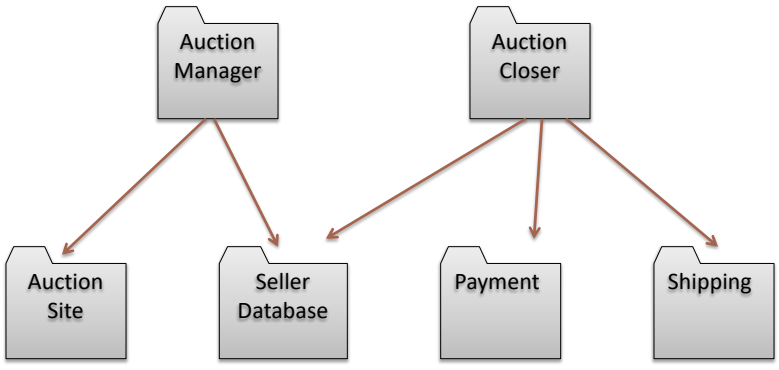
Loose Coupling Patterns

- Key patterns: Service Façade
  - Encapsulate and abstract internal services
  - Coarsen grain of interaction
- Concurrent Contracts

66

66

Concurrent Contracts



67

67

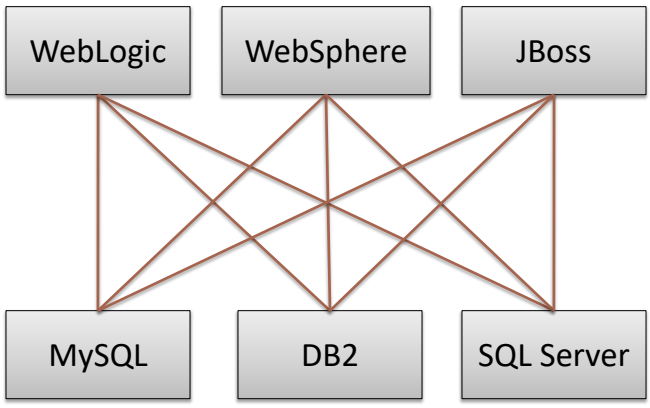
Loose Coupling Patterns

- Key patterns: Service Façade
  - Encapsulate and abstract internal services
  - Coarsen grain of interaction
- Concurrent Contracts
- Legacy Wrapper
  - DDD: Anti-corruption layer
  - Typically combines façade and adapter patterns

68

68

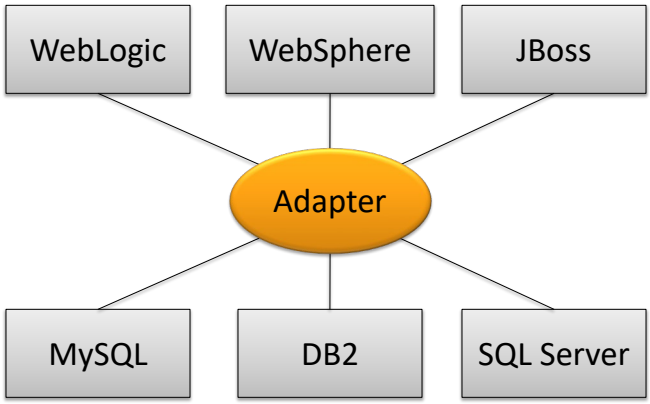
Legacy Integration



69

69

Legacy Adapter



70

70

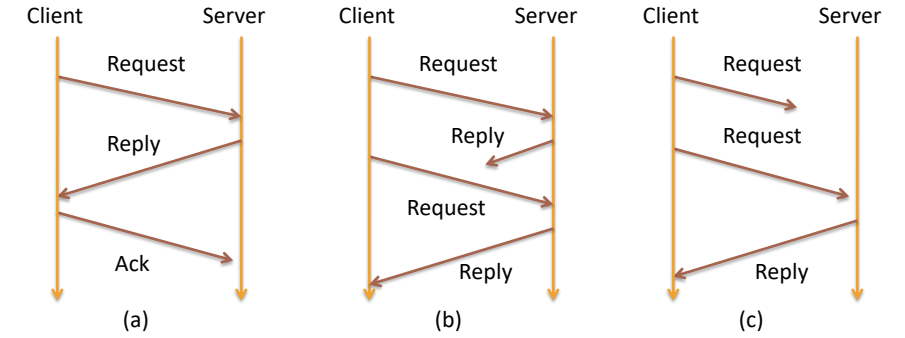
Cost of Loose Coupling

- Cost of transitioning between layers
  - Remapping network buffers between layers
  - Direct controller-to-application communication
  - Filter fusion
- Reliable failure-handling
  - RPC stack: tight coupling for reliability

71

71

Tight Coupling in RPC Stack



Req #: 17 | Req #: 17 | Req #: 17 | Req #: 17 | Req #: 17 | Req #: 16

Cost of Loose Coupling

- Cost of transitioning between layers
  - Remapping network buffers between layers
  - Direct controller-to-application communication
  - Filter fusion
- Reliable failure-handling
  - RPC stack: tight coupling for reliability

```
interface AddQ {
  int getRequestNum();
  void append(int reqNum, Object data);
}
```