Documentation for Software Architecture Patterns

Map-Reduce Pattern

Overview

The Map-Reduce pattern is used for analyzing vast amounts of data in a distributed, parallel manner. It splits large data sets, performs parallel processing, and combines the results efficiently.

Context

Businesses need to process enormous volumes of data (at petabyte scale) rapidly for analysis.

Problem

Efficiently sorting and analyzing ultra-large data sets that can be parallelized and distributed across multiple processors.

Solution

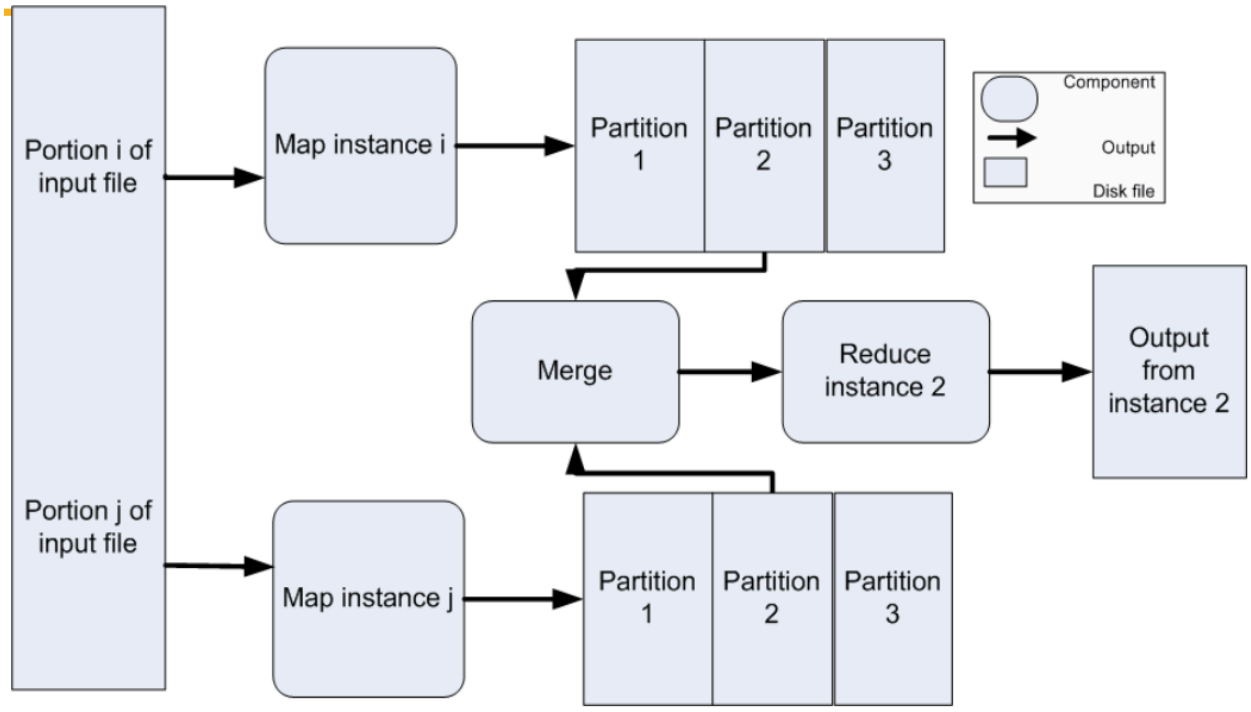
The Map-Reduce pattern consists of three main parts:

1. **Infrastructure**: Handles software allocation to hardware nodes, data sorting, and error recovery.
2. **Map Function**: Filters and extracts relevant data.
3. **Reduce Function**: Aggregates the results from the map functions.

Use Case Example

In search engines, Map-Reduce can be used to index web pages. The map function extracts keywords, while the reduce function aggregates the frequency of these keywords.

Diagram: Map-Reduce Example



Multi-Tier Pattern

Overview

The Multi-Tier pattern organizes a system's architecture into logical tiers, each containing specific groups of components.

Context

Commonly used in distributed systems where infrastructure needs to be organized into separate, logical layers for better modularity and scalability.

Problem

Efficiently splitting a system into independent execution structures, each handling specific tasks, while facilitating communication between these tiers.

Solution

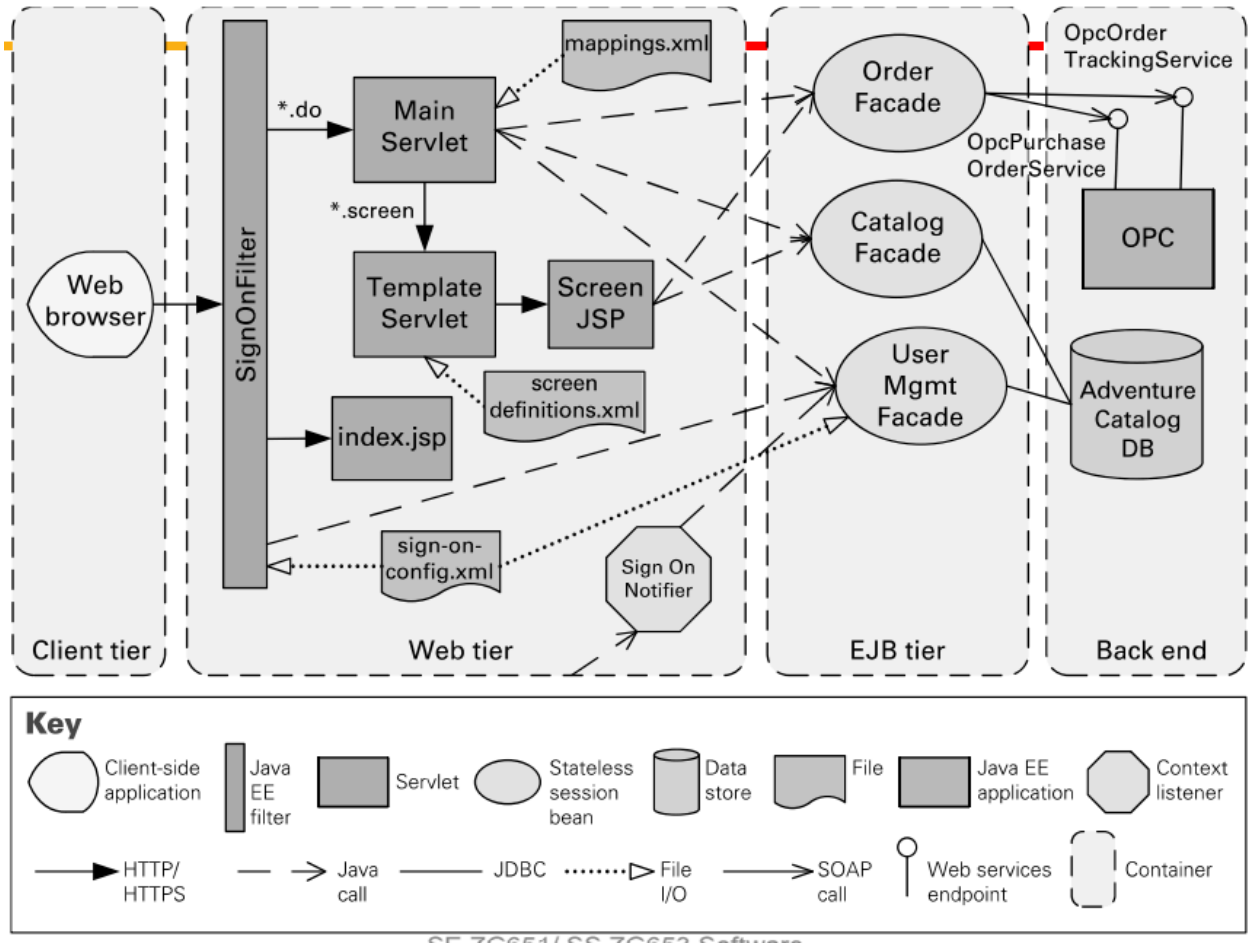
- Components are grouped into tiers that handle specific tasks.
- Tiers communicate with one another through defined interfaces and protocols.

Use Case Example

In e-commerce applications, there are typically three tiers:

1. **Presentation Tier**: Manages the user interface.
2. **Logic Tier**: Processes commands and makes logical decisions.
3. **Data Tier**: Stores and retrieves data from the database.

Diagram: Multi-Tier Example



Tactics and Interactions (1-10)

Overview

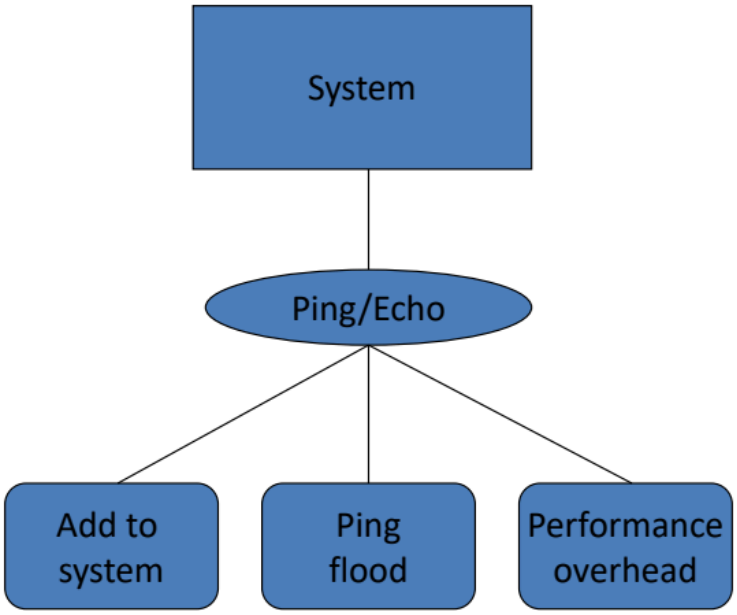
Tactics enhance software patterns by addressing specific quality attributes like performance, modifiability, and reliability. Each tactic introduces new considerations and potential side effects, leading to interactions that require additional tactics to resolve.

Tactics and Interactions Details

1. **Ping/Echo Tactic**

- **Purpose**: Detects faults by sending "ping" messages and expecting "echo" responses.
- **Common Side-Effects**:
  - **Security**: Vulnerable to ping flood attacks.
  - **Performance**: Can introduce latency due to frequent checks.
  - **Modifiability**: Integrating ping/echo mechanisms can be challenging.

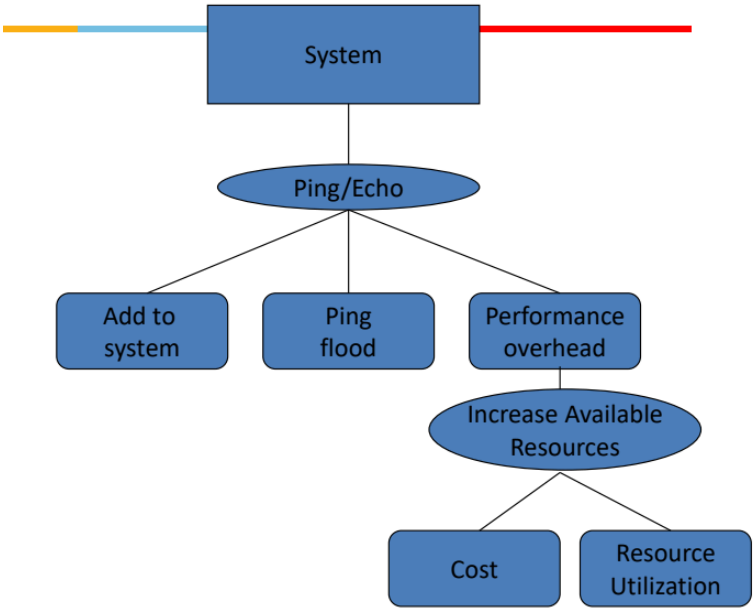
2. **Increase Available Resources**
   - **Purpose**: Enhances system performance by adding more hardware resources (e.g., servers, memory).
   - **Common Side-Effects**:
     - **Cost**: Additional resources require higher financial investment.
     - **Resource Utilization**: Effective use of increased resources must be ensured.
3. **Scheduling Policy**
   - **Purpose**: Optimizes how resources are allocated over time to improve performance.
   - **Common Side-Effects**:
     - **Modifiability**: Adding or changing scheduling policies can complicate the architecture.
     - **Adaptability**: Adjusting the scheduling policy to future needs requires careful planning.
   - **Diagram**: Tactics and Interactions - 3



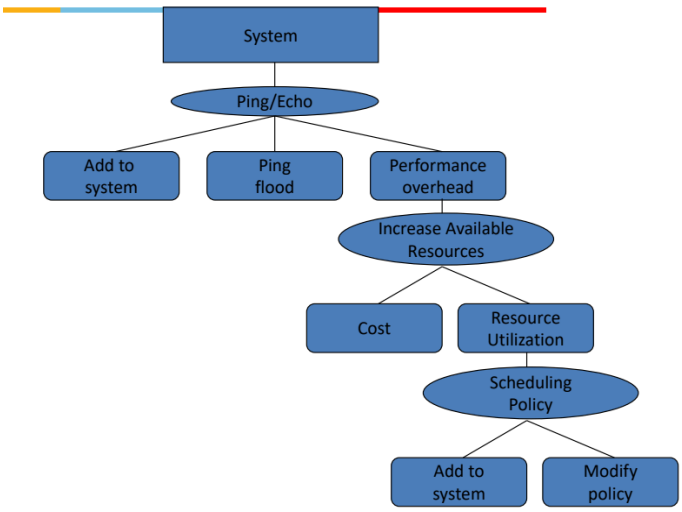
4. **Use an Intermediary**
   - **Purpose**: Introduces an intermediary (e.g., a broker or gateway) to manage communication.

- **Common Side-Effects**:
  - **Performance**: The intermediary may slow down communication.
  - **Modifiability**: Ensuring that all interactions go through the intermediary can be complex.

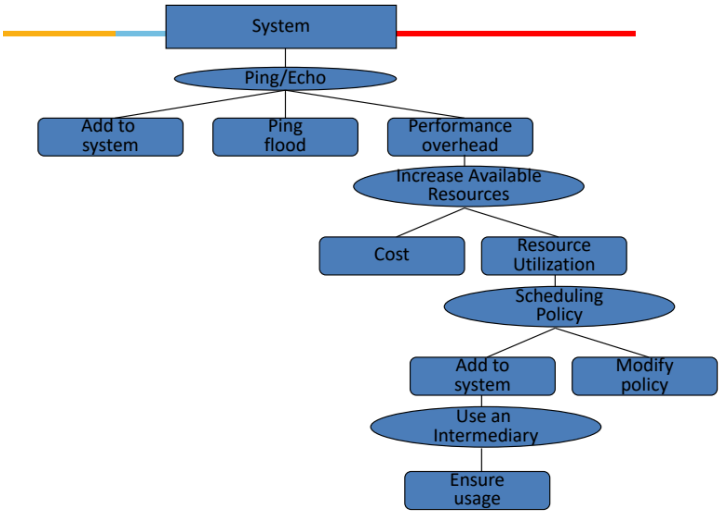
5. **Restrict Communication Paths**
   - **Purpose**: Limits communication paths to enforce security and control.
   - **Common Side-Effects**:
     - **Performance**: Restricting paths can introduce latency.
     - **Scalability**: It may limit the system's ability to scale.
   - **Diagram**: Tactics and Interactions - 5



6. **Maintain Multiple Copies**
   - **Purpose**: Increases availability by having redundant copies of components or data.
   - **Common Side-Effects**:
     - **Consistency**: Keeping all copies updated can be complex.
     - **Performance**: Synchronization of multiple copies can slow down the system.
7. **Load Balancing**
   - **Purpose**: Distributes workloads evenly across resources to prevent bottlenecks.
   - **Common Side-Effects**:
     - **Overhead**: The load balancer itself can become a bottleneck.
     - **Complexity**: Implementing effective load balancing adds architectural complexity.
   - **Diagram**: Tactics and Interactions - 7



8. **Checkpoint/Restart**
    - **Purpose**: Saves the system's state periodically to allow recovery after failures.
    - **Common Side-Effects**:
        - **Performance**: Frequent checkpoints can reduce performance.
        - **Storage Requirements**: Additional storage is needed for checkpoint data.
9. **Use Cache**
    - **Purpose**: Enhances performance by temporarily storing frequently accessed data.
    - **Common Side-Effects**:
        - **Consistency**: Ensuring data consistency between the cache and the main data source can be difficult.
        - **Resource Utilization**: Caching requires additional memory and management.
    - **Diagram**: Tactics and Interactions - 9



10. **Adjustable Granularity**
    - **Purpose**: Changes the granularity of tasks, data processing, or communication to balance performance and complexity.
    - **Common Side-Effects**:
        - **Modifiability**: Adjusting granularity can complicate system modifications.
        - **Performance**: Both coarse and fine granularity can have negative performance impacts, depending on the context.

Summary

- **Patterns** are reusable architectural structures addressing common problems in software design.
- **Tactics** enhance patterns by focusing on specific quality attributes.
- The combination of tactics and patterns helps create robust architectures that meet system requirements.