Cloud Services & Distributed Computing

Design Challenges of
Non Functional Requirements

Non-Functional Requirements

Our objective:

To understand the non-functional requirements of a distributed system.

These are concerned with the **quality** of the system

Certain requirements are common to many distributed systems:

- Resource Sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

Resource Sharing

- Ability to use any hardware, software or data anywhere in the system

- Cost effective for sharing expensive resources

- Security implications due to many users accessing the common resources
  - How do control/restrict access? Resource manager components
  - Resource manager controls access, provides naming scheme and controls concurrency
  - Client-server v n-tier architecture (distributed objects)

Openness

- Openness is concerned with extendibility and improvability of distributed systems

- Well defined and documented interfaces of components need to be published
  - Allows other components know what services are available

- System needs to adhere to recognized standards

- Components achieve openness by communicating using well-defined interfaces
  - supports changing functional requirements as an organisation grows
  - Helps preserve the investment
  - Supports the integration of new components

Concurrency

- Components in distributed systems are executed in concurrent processes
- Components access and update shared resources (e.g. variables, databases, device drivers)
- Integrity of the system may be violated if concurrent updates are not coordinated
  - Lost updates
  - Inconsistent analysis (Non-Repeatable Read)

Concurrency

- Lost Update
  - Lost updates occur when two or more processes select the same data and then update the data based on the value originally selected.
  - Each process is unaware of the other processes.
  - The last update overwrites updates made by the other process, which results in lost data.

1. Session #1 reads Account A, gets 300.
2. Session #2 reads Account A, gets 300.
3. Session #2 updates Account A to 400 (+100) and commits.
4. Session #1 updates Account A to 350 (+50) and commits.

  - In this scenario, because Session #1 does not know that another session has already modified the account, the update by Session #2 is overwritten ("lost").

Scalability

- Adoption of distributed systems to
  - accommodate a growing load (e.g. more users)
  - respond faster (this is the hard one)

- Usually done by adding more and/or faster hosts or processors

- Components should not need to be changed when scale of a system increases

- Design components to be scalable!
  - They shouldn’t have to be revisited as things scale up

Fault Tolerance

- Hardware, software and networks fail!
  - For lots of reasons

- Distributed systems must maintain availability even at low levels of hardware/software/network reliability
  - Huge improvement over centralised systems

- Operations that continue even in the presence of faults are referred to as fault-tolerant

- Fault tolerance is achieved by
  - redundancy (replication)

Transparency

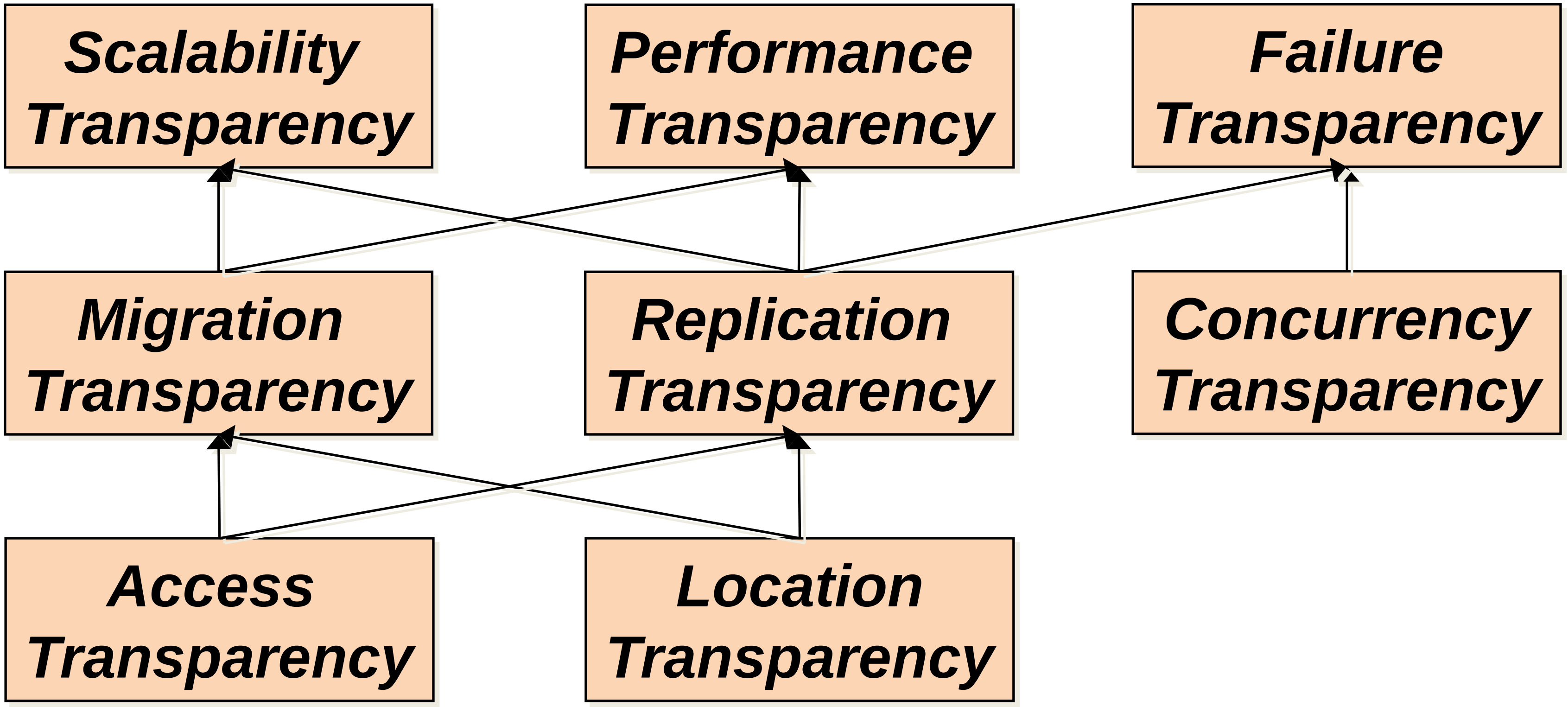
- Non-functional requirements can be satisfied through the various forms of transparency within distributed systems.

- Application developers and users should perceive distributed systems as
  - One system
  - A collection of co-operating components which should be hidden
  - Efficient and cost-effective to construct and maintain (only possible if the complexity of distribution is hidden from the users)

Dimensions of Transparency

Defined in the international standard on Open Distributed Processing (ODP):

> “Distribution transparency is the property of hiding the properties of distribution from end users”.



Dimensions of Transparency

ACCESS Transparency

- Enables local and remote components to be accessed using identical operations.
- Example: File system operations in NFS.
- Example: Navigation in the Web.
- Example: SQL Queries

It is critical in building distributed systems using heterogeneous computer architecture and programming languages.

Dimensions of Transparency

LOCATION Transparency

- Enables components to be accessed without knowledge of their location.
- Example: File system operations in NFS
- Example: Pages in the Web
- Example: Tables in distributed databases

Dimensions of Transparency

MIGRATION Transparency

- A component can be relocated without users or clients noticing it (i.e. allows the movement of information objects within a system without affecting the operations of users or application programs)
- Example: NFS
- Example: Web Pages

Dimensions of Transparency

REPLICATION Transparency

- A replica is a component copy that remains synchronized with its original.
- Users or application programs have no knowledge of the replica.
- Example: Distributed DBMS
- Example: Mirroring Web Pages.

Dimensions of Transparency

CONCURRENCY Transparency

- Users and programmers are unaware that components request services concurrently.
- Enables several processes to operate concurrently using shared information objects without interference between them.
- Example: NFS
- Example: Automatic teller machine network
- Example: Database management system

Dimensions of Transparency

SCALIBILITY Transparency

- Allows the system and applications to expand in scale without change to the system structure or the application algorithms.
- Example: World-Wide-Web
- Example: Distributed Database

Dimensions of Transparency

PERFORMANCE Transparency

- Allows the system to be reconfigured to improve performance as loads vary.

- Example: Distributed make.

Dimensions of Transparency

FAILURE Transparency

- Enables distributed system to conceal faults.
- Allows users and applications to complete their tasks despite the failure of other components.
- Depends on concurrency and replication transparency.
- Example: Database Management System

Transparency

Summary

- Transparency - Perceive distributed systems as One system
- Access - Enables local and remote information objects to be accessed using identical operations
- Location - Enables information objects to be accessed without knowledge of their location
- Migration - A component can be relocated without users or clients noticing it
- Replication - A replica is a component copy that remains synchronized with its original.
- Concurrency - Enables several processes to operate concurrently using shared information objects without interference between them.
- Scalability - Allows the system and applications to expand in scale without change to the system structure or the application algorithms
- Performance - Allows the system to be reconfigured to improve performance as loads vary
- Failure - Enables distributed system to conceal faults.