

8 Fault-Tolerant CORBA

- 8.1 Überblick
- 8.2 CORBA / FT-CORBA
- 8.3 Grundlagen
- 8.4 Architekturübersicht
- 8.5 Adressierung
- 8.6 Verhalten bei Ausfällen
- 8.7 Replikation
- 8.8 Verwaltung von Replikaten
- 8.9 Fehlermanagement
- 8.10 Recoverymanagement
- 8.11 Implementierung



- Grundlagen bzw. Basismechanismen
- Einstellung von Fehlertoleranzeigenschaften
- Verwaltung von Replikaten
- Fehlermanagement
- Recoverymanagement
- Implementierung



■ CORBA

- CORBA spezifiziert keine Mechanismen für Redundanz
- Zuverlässige Verbindungen basierend auf TCP/IP ermöglichen beschränkte Erkennung von Ausfällen

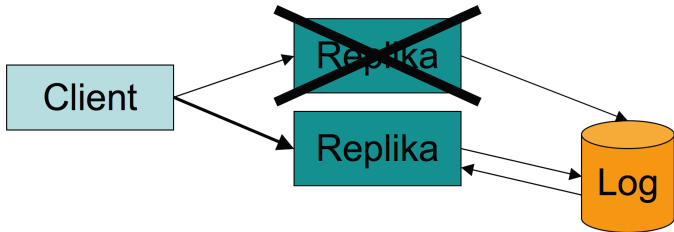
■ Ziele von FT-CORBA

- minimale Modifikation von Applikationen
 - (sowohl auf Client- wie auch auf Serverseite)
- Unterstützung für Fehlertoleranz gegenüber von **fail stop** Fehlern
 - verschiedene Fehlertoleranz-Anforderungen
 - verschiedene Applikationen
 - verschiedene Mechanismen zur Fehlererkennung und -behandlung



Was muss eine Fehlertoleranzinfrastruktur leisten?

- Replikation von Objekten
- Erkennung von Ausfällen
- Logging von Anfragen und Zustandssicherung von Objektzuständen
- Zustandstransfer zur Initialisierung und Reinitialisierung von Objekten
- Transparente Verschattung von Ausfällen



- Objekte bilden die Einheit der Replikation
- Starke Konsistenz
 - Alle Replikate haben den gleichen Zustand
 - Ermöglicht einfache Anwendungsentwicklung
 - Client interagiert mit Objekten
 - Stellt hohe Anforderungen an die Mechanismen der Infrastruktur
 - Alle Mechanismen zur Fehlertoleranz werden durch die Middleware bereitgestellt



■ Objektgruppe

- Menge aller Replikate eines Objektes bilden eine Objektgruppe
- Jede Gruppe verfügt über eine **Object Group Reference (IOR)**
- Zielsetzung
 - Replikationstransparenz
 - Fehlertransparenz

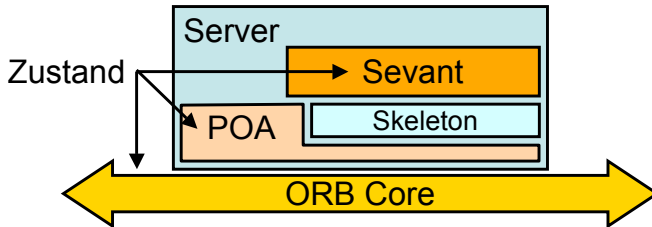
■ Identität

- CORBA-Objekte werden durch ihren Ausführungsort identifiziert
 - schwache Form von Identität
- FT-CORBA benötigt eindeutige und ortsunabhängige Identifikation
 - Objektgruppen werden durch `FTDomainId` und `ObjectGroupId` identifiziert
 - einzelne Replikate durch `FTDomainId`, `ObjectGroupId` und Ort



Grundlagen: Zustand

- Zustand bildet die Menge an Informationen die zur Erhaltung von konsistenten Replikaten nötig ist
- Zustand ergibt sich durch das replizierte Objekt und die Infrastruktur (POA und ORB)



- Replikation von Objekten erfordert deterministisches Verhalten
 - Aufrufe werden an alle Replikate in gleicher Reihenfolge zugestellt
 - Ausgehend von identischen Ausgangszuständen werden identische Endzustände erreicht
 - Objekt entspricht somit einem Zustandsautomat
- Problemfälle
 - objektexterne Informationen
 - z.B. Zeit, Zufallszahlen oder Aufruf an externen Informationsquellen
 - Koordinierung wenn parallel mehrere Threads ein Objekt verändern
 - z.B. Anforderung von Locks



■ **Active** Replication

- Replikate bearbeiten alle Anforderungen

■ **Passive** Replication

- Es gibt ein aktives Replikat (*master*) welches Aufrufe bearbeitet
- Alle anderen Replikate sind in Wartestellung

■ Anwendungen besitzen unterschiedliche Fehlertoleranzanforderungen

■ Active Replication

- Sehr kurze Verzögerungen im Fehlerfall
- Hoher Aufwand unter anderem durch die erforderliche Gruppenkommunikation

■ Passive Replication

- Längere Verzögerung bei Ausfällen
- Geringer Aufwand zur Laufzeit
- Schnellere Antwortzeit im Normalbetrieb



- Objektreplikation
- Verwaltung der System- und Fehlertoleranzanforderungen pro Objektgruppe
 - Property Manager-Schnittstelle
- Erzeugen von replizierten Objekten
 - Generic Factory-Schnittstelle
 - Replication Manager-Schnittstelle
 - Zustandstransfer
- Erkennung von Ausfällen



- Failover
 - Falls ein Server nicht antwortet wird es entweder erneut versucht oder ein anderer Server wird kontaktiert
 - Aufrufe werden nur einmal durch das replizierte Objekt ausgeführt (muss durch die Serverseite unterstützt werden)
- Adressierung
 - Veraltete Referenzen werden in Kooperation mit dem Server ersetzt
 - Server liefert aktuelle Version
- Ausfall der Verbindung
 - z. B. kein Replikat ist erreichbar
 - Anwendung wird benachrichtigt



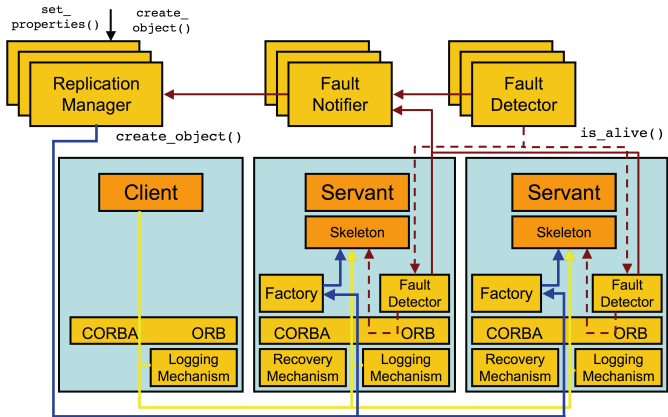
- Infrastruktur kontrolliert Fehlertoleranz
 - Automatische Erzeugung von Replikaten
 - Automatische Erhaltung der Konsistenz
- Applikation kontrolliert Fehlertoleranz (nur in Spezialfällen nötig!)
 - Applikation lenkt die Erzeugung und Platzierung von Replikaten
 - Applikation gewährleistet die Konsistenz
- Zuständigkeit: Fault Tolerance Domain
 - Menge von Rechnern zur Bereitstellung von fehlertoleranten Applikationen
 - Zentrale Komponente bildet der Replication Manager
 - steuert Erzeugung und Platzierung von Replikaten



- Determinismus bleibt Aufgabe des Anwendungsentwicklers
- Keine explizite Unterstützung für
 - Behandlung von Netzwerkpartitionierungen
 - böartige Fehler
 - Softwarefehler und Designfehler
- Interoperabilität
 - Alle Replikate eines replizierten Objektes müssen die gleiche Infrastruktur nutzen

Es ist Aufgabe der Hersteller hier individuelle Lösungen anzubieten!

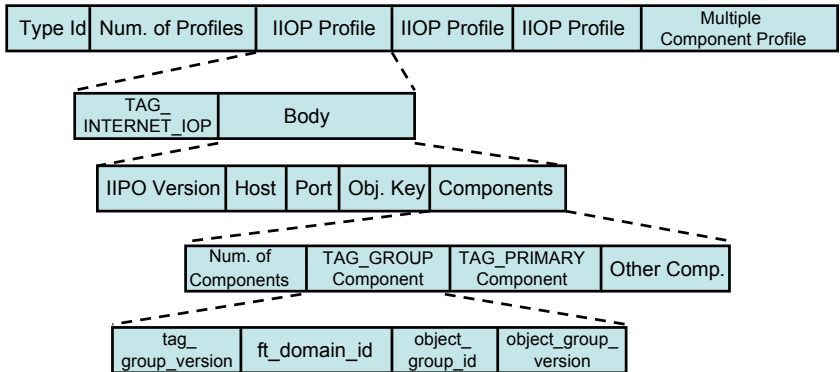




- Interoperable Object Group Reference (IOGR)
 - Eine IOGR besitzt mehrere IOR Profile
 - Jedes Profil enthält Informationen zur Identität des Objektes
 - `FTDomainId` identifiziert die Domäne
 - `ObjectGroupId` identifiziert das replizierte Objekt (innerhalb der Domäne)
 - `ObjectGroupRefVersion` legt die Version der Referenz fest
 - Maximal ein Profil enthält die Komponente `TAG_PRIMARY` zur Identifikation des primären Replikats.
 - Achtung! diese Information muss nicht aktuell sein!
- Was wird adressiert?
 - direkte Adressierung der Replikate
 - Adressierung von Gateways



Interoperable Object Group Reference (IOGR)



- Aktualität von IOGR
 - Problem: Objektreferenzen können veralten
 - z.B einzelne Replikate fallen aus, oder es kommen neue Replikate hinzu
 - Lösung: Version der Client-Referenz wird bei Anfragen übertragen
 1. Versionsinformation aus der Referenz wird als `GROUP_VERSION Service Context` der Anfrage beigelegt
 2. Server entnimmt Anfragen die Versionsinformation
 3. Ist die Version des Clients aktuell wird die Anfrage verarbeitet
 4. Ist die Version des Clients veraltet wird eine `LOCATE_FORWARD_PERM` Exception erzeugt
 5. Ist die Version des Servers (anscheinend) veraltet wird der Replication Manager nach der aktuellen Referenz gefragt



- Wiederholung von Aufrufen bei Ausfällen von Replikaten
- Auf ORB-Transportebene gibt es eine der folgenden Exceptions
 - `COMM_FAILURE`, `TRANSIENT`, `NO_RESPONSE`, `OBJ_ADAPTER`
 - Status ist `COMPLETED_MAYBE`
- Problem
 - Ohne weitere Maßnahmen kann eine Verletzung der at-most-once Semantik vorliegen
- Lösung
 - Eindeutige Identifizierung von Anfragen durch `REQUEST Service Context`
 - `Client Id`, identifiziert den Client
 - `Retention Id`, identifiziert den Aufruf
 - `Expiration Time`, legt fest wie lange Aufrufergebnisse aufbewahrt werden
 - Server ORB erkennt so die Wiederholungen von Anfragen und sendet das bereits ermittelte Ergebnis



■ Problem

- Es kommt zu einem Serverausfall oder Verbindungsproblem während eines Aufrufs
- Die TCP/IP Verbindung wird nicht ordnungsgemäß beendet und der Client bekommt den Abbruch deshalb nicht mit

■ Lösung

- Periodische Testaufrufe (heartbeat messages)
- Client fragt diese via Policy pro Verbindung an und spezifiziert
 - Aufruffrequenz
 - Timeout
- Client-ORB ruft `_FT_HB()` beim Server-ORB auf
- Operation wird vom Skeleton bereitgestellt
- Server-ORB muss dies explizit erlauben
 - Kontrolle von erzeugtem Netzwerkverkehr



- Konfigurierbare Eigenschaften
 - Replikation
 - Mitgliedschaft
 - Konsistenz
 - Monitoring
 - Intervall und Timeout
 - Erzeugung replizierter Objekte mittels Factories
 - Weitere Konfigurationseigenschaften
 - Replikatanzahl (initiale bzw. minimale Anzahl)
 - Sicherungspunktintervall



- Stateless
 - statische Daten und nur lesender Zugriff
- Cold Passive Replication
 - Wiederherstellung wird durch Sicherungspunkt und Protokollierung von Aufrufen erreicht
 - langsame Kompensation von Ausfällen
- Warm Passive Replication
 - Aktueller Zustand des primären Replikats wird periodisch an alle anderen Replikate übertragen
 - Protokollierung von Aufrufen
 - schnelleres Recovery im Vergleich zu cold passive
- Active Replication
 - alle Replikate bearbeiten Aufrufe
 - sehr geringe Verzögerung bei Ausfällen

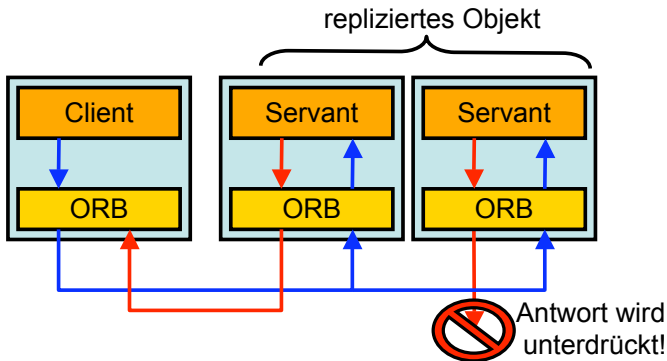


- Normalbetrieb
 - Wenn replizierte Objekte von anderen replizierten Objekten aufgerufen werden, müssen duplizierte Aufrufe und Antworten unterdrückt werden
- Behandlung von Ausfällen
 - Infrastruktur verschattet Ausfälle transparent da mindestens ein Replikat auf Anfragen antwortet
- Behandlung von Beitritten
 - Zustandstransfer zur Integration eines neuen oder temporär ausgefallenen Replikats nötig



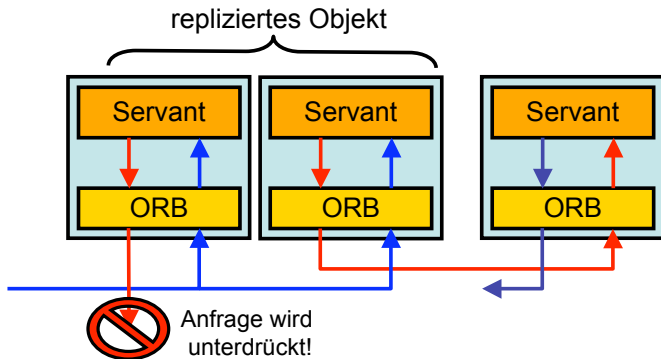
Active Replication

- Unterdrückung von duplizierten Ergebnissen
 - Kann im Prinzip auf Seite des Aufrufers oder der Replikate erfolgen



Active Replication

- Unterdrückung von duplizierten Aufrufen
 - Nötig wenn ein repliziertes Objekt andere Objekte aufruft

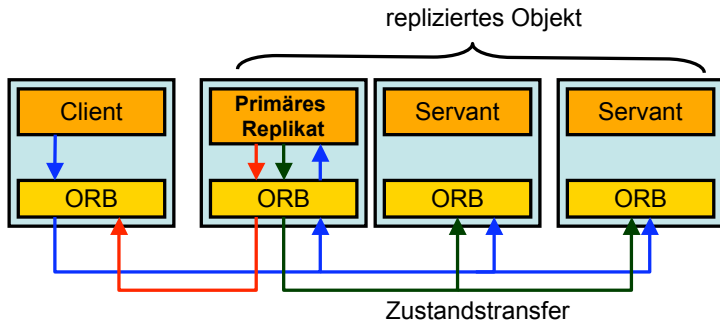


- Normalbetrieb
 - periodischer Zustandstransfer zu allen Replikaten
(nur bei warm passive nötig)
 - Logging von Aufrufen und Sicherungspunkten
- Behandlung von Ausfällen
 - Ausfall des primären Replikats erfordern die Wahl eines neuen Replikats
 - Initialisierung des neuen primären Replikats
(Zustandstransfer bei cold passive Replication)
 - Erkennung von duplizierten Anfragen
- Behandlung von Beitritten
 - Zustandstransfer zu neuem oder wiederhergestelltem Replikat bei warm passive Replication

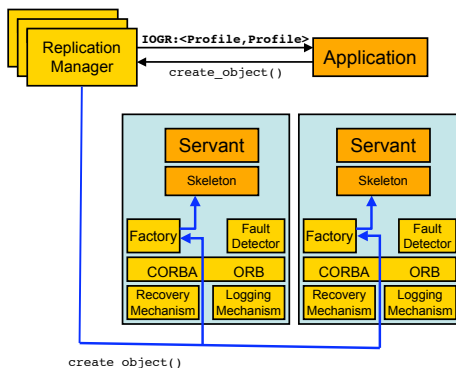


Warm Passive Replication

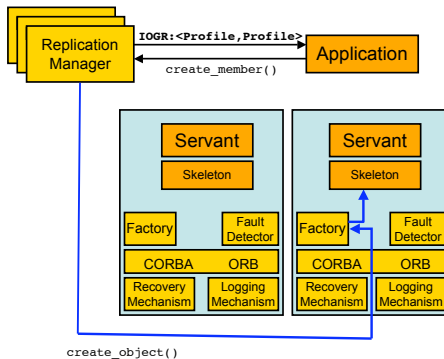
- Protokollierung von Anfragen (durch passive Replikate bzw. die Middleware)
- Periodischer Zustandstransfer



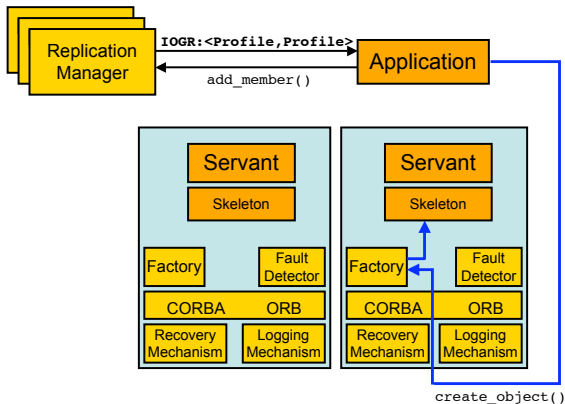
- Infrastruktur kontrolliert Replikaterzeugung
 - Infrastruktur erzeugt Replikate und platziert sie in der Domäne
 - Anwendung ruft hierzu die `create_object()`-Methode des Replication Managers auf



- Anwendung kontrolliert Replikaterzeugung
 - Anwendung erzeugt und platziert Replikate entsprechend ihrer Erfordernisse
 - Anwendung beauftragt den Replication Manager zur Erzeugung einzelner Replikate



- Anwendung kontrolliert Replikaterzeugung
 - Anwendung erzeugt Replikate eigenständig und nutzt den Replication Manager zur Verwaltung



- Infrastruktur kontrolliert Konsistenz durch Mechanismen wie Logging, Zustandssicherung und Recovery
 - Active Replication
 - Nach jedem Aufruf haben alle Replikate den gleichen Zustand
 - Erfordert eine Gruppenkommunikation und total geordnete Zustellung von Nachrichten
 - Passive Replication
 - Nach jedem Zustandstransfer haben alle Replikate den gleichen Zustand
- Anwendung kontrolliert Konsistenz
 - Anwendung stellte alle Mechanismen zur Konsistenzerhaltung bereit



- Granularität der Ausfallüberwachung
 - **Mitglieder**
 - Es wird jedes Mitglied einer Objektgruppe beobachtet
 - **Ort**
 - Es wird ein Objekt als Repräsentant für den Ort beobachtet
 - Wird das beobachtete Replikat beendet wird ein anderes Objekt ausgewählt
 - Fällt das Objekt aus wird der Rechner als ausgefallen betrachtet
 - **Ort und Typ**
 - Es wird ein Objekt pro Ort und Typ kontrolliert
 - Wird das beobachtete Replikat beendet wird ein anderes Objekt vom gleichen Typ ausgewählt
 - Fällt das Objekt aus werden alle Objekte des Typs an diesem Ort als ausgefallen betrachtet



- Erzeugung von replizierten Objekten wird durch eine Sequenz von **FactoryInfo** Strukturen konfiguriert
- Aufbau der **FactoryInfo** Struktur
 - Referenz auf Fabrik
 - Ort der Fabrik
 - Information zur Konfiguration bzw. Eigenschaften der Fabrik (**Criteria**)



- Aufgaben des Replication Manager
 - Verwaltung von Objektgruppen und ihren Eigenschaften
 - Replikationsart
 - Mitgliedschaft
 - Konsistenz
 - usw.
- Technische Realisierung
 - Methoden zur Benachrichtigung über Ausfälle
 - `register_fault_notifier()`
 - `get_fault_notifier()`
 - Erbt von
 - Property Manager – Management von Fehlertoleranzeigenschaften
 - Object Group Manager – Verwaltung von Objektgruppen
 - Generic Factory – Erzeugung von replizierten Objekten



- Ermöglicht Konfiguration von Eigenschaften
 - für alle Objektgruppen
 - für alle replizierten Objekte eines bestimmten Typs
 - für ein bestimmtes repliziertes Objekt zum Zeitpunkt der Erzeugung
 - für ein bestimmtes repliziertes Objekt zu Laufzeit

Spezifischere Definitionen haben höhere Priorität



Eigenschaften	Standard	Typ	Erzeugung	Dynamisch
Replikation	X	X	X	
Mitgliedschaft	X	X	X	
Konsistenz	X	X		
Monitoring	X	X		
Granularität	X	X	X	X
Fabriken		X	X	X
Initiale Replikatanzahl	X	X	X	
Minimale Replikatanzahl	X	X	X	X

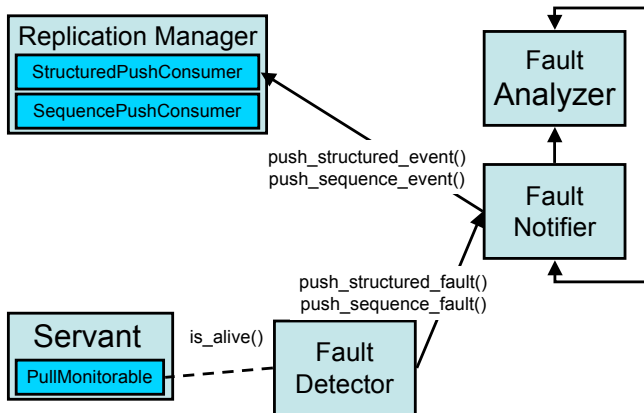


- Operationen der Object Group Manager Schnittstelle:
 - `create_member()`
 - `add_member()`
 - `set_primary_member()`
 - `remove_member()`
 - `locations_of_members()`
 - `get_object_group_ref()`
 - `get_object_group_id()`
 - `get_member_ref()`



- Fault Detector
 - Erkennt Fehler und erzeugt Fehlerberichte
- Fault Notifier
 - Sammelt und verarbeitet Fehlerberichte von Fault Detectors und Fault Analyzers
 - Bildet eine Art Datenbank für alle Fehlerberichte
- Fault Analyzer
 - Spezifisch für jede Anwendung
 - Verarbeitet Fehlerberichte und fasst abhängige Fehler zusammen





- Fehlerweitergabe durch Fault Detectors
 - einzelne Fehlerreporte (`CosNotification::StructuredEvent`)
 - Sammelreport (`CosNotification::EventBatch`)
 - Fehlertyp: (`ObjectCrashFault`)
 - Domain_name - FT_CORBA
 - Type_name - ObjectCrashFault
 - Location - host/process
 - TypeId - IDL:Library:1.0
 - ObjectGroupId - 4711
 - Sind alle Objekte eines Ortes ausgefallen wird `TypeId` und `ObjektGroupId` nicht im Fehlerreport vermerkt
 - Sind alle Objekte eines Types ausgefallen wird die `ObjektGroupId` weggelassen

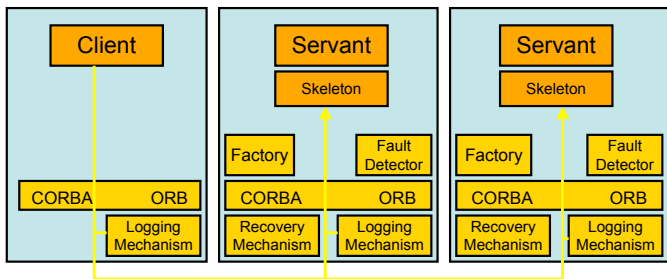


- Fehlerverarbeitung
 - Fehlerberichte werden durch Fault Detectors erzeugt und via *push* an den Fault Notifier weitergeleitet
 - Beliebige Konsumenten registrieren sich beim Fault Notifier und werden über Fehler benachrichtigt
 - Filter reduzieren das Nachrichtenaufkommen



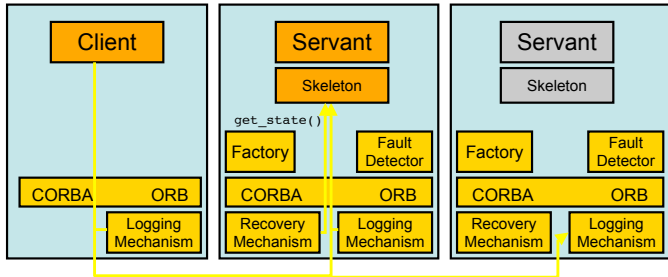
Logging und Recovery

Beispiel: Active Replication



Logging und Recovery

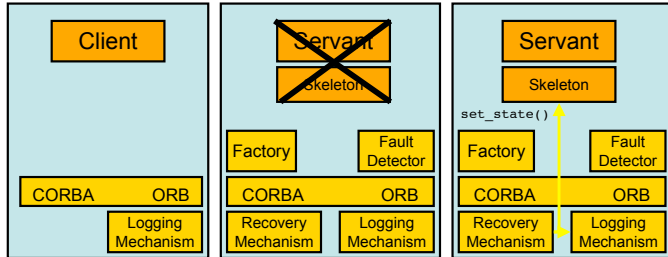
Beispiel: Cold Passive Replication — Normalbetrieb



- Anfragen werden protokolliert durch die Middleware
- Zyklische Erzeugung eines Sicherungspunktes (`get_state()`) der an die Standorte von passiven Replikaten vermittelt wird



Beispiel: Cold Passive Replication — Wiederherstellung



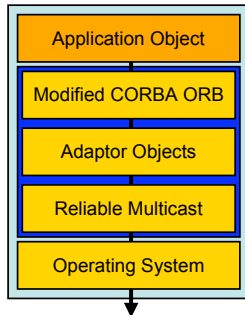
- Bei einem Ausfall wird zunächst der letzte Sicherungspunkt eingespielt und danach bereits bearbeitete aber im Sicherungspunkt noch nicht erfasste Anfragen nochmals ausgeführt
 - Nach Abschluss dieses Vorgangs können neuen Anfragen bearbeitet werden



- Integrativer Ansatz
 - Fehlertoleranz wird direkt durch den ORB realisiert
- Dienst-Ansatz (entstanden vor FT-CORBA)
 - Unterstützung für replizierte Dienste wird als CORBA Service angeboten
- Interceptor-Ansatz
 - Aufrufe werden durch Interceptoren (unterhalb des ORBs) abgefangen und für die Bereitstellung von replizierten Diensten modifiziert



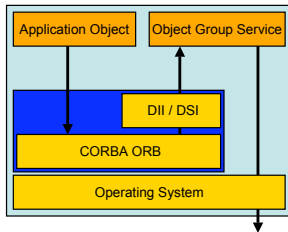
- ORB wird durch eine Gruppenkommunikation ergänzt
- IIOP wird durch ein proprietäres Protokoll ausgetauscht



- Nachteil: Aufwendige Implementierung
- Vorteil: Effizient und transparent für die Applikation und kann FT-CORBA-kompatibel sein



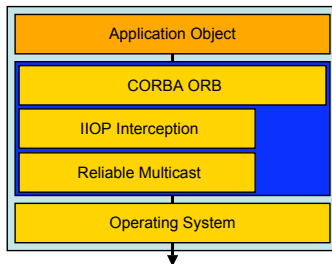
- Anwendung kommuniziert über lokale Dienste mit dem replizierten Objekt



- Nachteile:
 - Indirektion über den ORB
 - Nicht transparent für die Anwendung und nicht FT-CORBA-kompatibel
- Vorteil: Keine Veränderung des ORBs nötig und damit portabel



- Interceptoren fangen Aufrufe ab und vermitteln sie weiter an das replizierte Objekt



- Nachteil: Interceptoren sind oft spezifisch für ein Betriebssystem
- Vorteil: Transparent für die Anwendung

■ Zustandstransfer

- Objekt muss die **Checkpointable**-Schnittstelle implementieren
- Erfordert den gesamten Objektzustand als **ByteArray** bereitzustellen
 - Aufgabe des Entwicklers!
- Sicherungspunkt kann erstellt werden wenn sich das Objekt in einem konsistenten Zustand befindet
 - alle laufenden Aufrufe werden beendet
 - neue Aufrufe werden blockiert

■ Determinismus

- Keine Aufrufe zu externen Informationsquellen
- In der Regel Verlust von mehrfädiger Ausführung



- FT-CORBA bildet einen Standard zur Entwicklung fehlertoleranter Infrastrukturen und Anwendungen
- Durch Middleware-Mechanismen und starke Konsistenz sind replizierte Objekte weitgehend transparent für Anwendungen
- Objektimplementierungen müssen angepasst werden
 - Determinismus
 - Zustandstransfer
- Kritikpunkte
 - Einheit der Replikation: Objekt vs. Prozess
 - Nichtdeterminismus ist Sache des Anwendungsentwicklers
 - Zustand: Objekt + Infrastruktur
 - Zustand der Infrastruktur ist schwer zu ermitteln
 - Konfiguration ist kompliziert
 - Stark konsistente Replikation erfordert aufwendige Gruppenkommunikation



- [OMG, 2004] Object Management Group
CORBA/IIOP Specification (Chapter 23, Fault Tolerant CORBA)
OMG Technical Committee Document formal/04-03-21, 2004.
- [Felber, 2004] Pascal Felber, Priya Narasimhan
Experiences, Strategies, and Challenges in Building
Fault-Tolerant CORBA Systems
IEEE Transactions on Computers, vol. 53, no. 5, pp. 497-511, 2004
- [Baldoni, 2002] R. Baldoni, C. Marchetti, R. Panella, L. Verde
Handling FT-CORBA Compliant Interoperable Object Group References
*WORDS '02: Proceedings of the The Seventh IEEE International Workshop on
Object-Oriented Real-Time Dependable Systems, 2002*
- [Narasimhan, 2007] Priya Narasimhan
Fault-Tolerant CORBA: From Specification to Reality
Computer , vol. 40, no. 1, pp.110-112, Jan. 2007

