Überblick

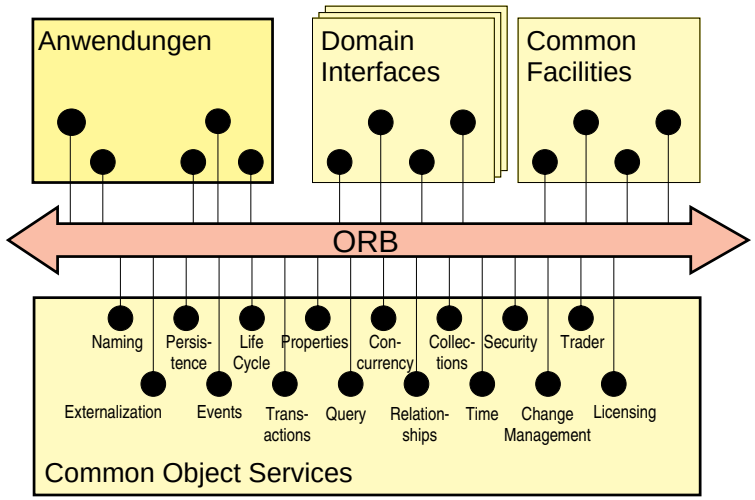
- Einführung in CORBA
  - plattformunabhängige Middleware-Architektur für verteilte Objekte
- Fault-Tolerant CORBA
  - Grundlagen bzw. Basismechanismen
  - Einstellung von Fehlertoleranzeigenschaften
  - Verwaltung von Replikaten
  - Fehlermanagement
  - Recoverymanagement
  - Implementierung

Motivation für CORBA

- Verteilte objektbasierte Programmierung
  - verteilte Objekte mit definierter Schnittstelle
- Heterogenität in Verteilten Systemen
  - verschiedene Hardware-Architekturen
  - verschiedene Betriebssysteme und Betriebssystem-Architekturen
  - verschiedene Programmiersprachen

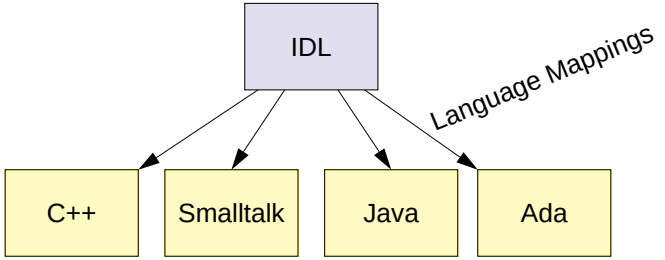
  ⇒ Transparenz der Heterogenität
- Dienste im verteilten System
  - Namensdienst, Zeitdienst,...
- Was ist CORBA?
  - **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
    - plattformunabhängige Middleware-Architektur für verteilte Objekte
    - Sammlung von Standard-Dokumenten verwaltete durch die OMG
    - `http://www.omg.org/spec/CORBA/3.1`

OMA – Object Management Architecture



Interface Definition Language (IDL)

- Sprache zur Beschreibung von Objekt-Schnittstellen
  - unabhängig von der Implementierungssprache des Objekts
  - Sprachabbildung (Language-Mapping) definiert, wie IDL-Konstrukte in die Konzepte einer bestimmten Programmiersprache abgebildet werden
  - Language-Mapping ist Teil des CORBA standards
  - Language mappings sind festgelegt für:
    - Ada, C, C++, COBOL, Java, Lisp, PL/I, Python, Smalltalk
    - weitere inoffizielle Language-Mappings existieren,z.B. für Perl



Interface-Definition-Language

- IDL angelehnt an C++
  - geringer Lernaufwand
- eigene Datentypen
  - Basistypen
  - Aufzählungstyp
  - zusammengesetzte Typen
- Module
  - definieren hierarchischen Namensraum für Anwendungsschnittstellen und -typen
- Schnittstellen
  - beschreiben einen von außen wahrnehmbaren Objekttyp
- Exceptions
  - beschreiben Ausnahmebedingungen

Interface-Definition-Language

- Umsetzung von IDL in Sprachkonstrukte (z.B. Java)

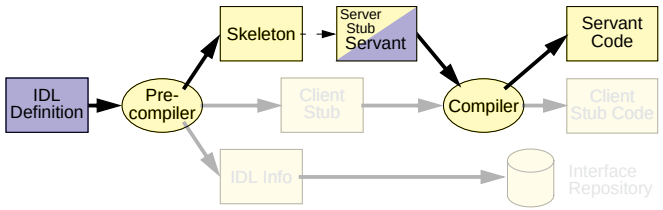
```
// IDL
module MyModule{
 interface MyInterface{
   attribute long lines;
   void printLine( in string toPrint );
 };
};
```

```
//Java
package MyModule;

public interface MyInterface extends ... {
 public int lines();
    public void lines( int lines );
 public void printLine( java.lang.String toPrint );
 ...
};
```

Objekterzeugung

- Ablauf auf Serverseite
  - Beschreibung der Objektschnittstelle in IDL
  - Programmierung des Server-Objekts in der Implementierungssprache
- Stub/Skeleton-Erzeugung

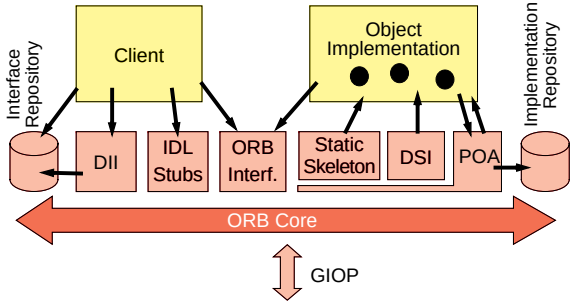


Objekte Binden

- Ablauf auf Serverseite
  - Registrierung des Objekts am **ORB** (bzw. dem Portable-Object-Adaptor (POA))
    - der ORB erzeugt eine Interoperable Object Reference (IOR)
- Binden des Clients an das Server-Objekt
  - Referenz auf Server-Objekt besorgen
    - Rückgabewert eines Methodenaufrufs
    - Ergebnis einer Namensdienst-Anfrage
    - von 'ausserhalb' des Systsems: Benutzer kennt Referenz als String (IORs können in Strings konvertiert werden und umgekehrt)
- Erzeugung des Client-Stub & Methodenaufruf über Client-Stub

ORB Architektur

- **O**bject **R**equest **B**roker – ORB
  - ist das Rückgrat einer CORBA-Implementierung
  - vermittelt Methodenaufrufe von einem zum anderen Objekt
    - ... innerhalb/zwischen Adressräumen/Prozessen von (verschiedenen) ORBs



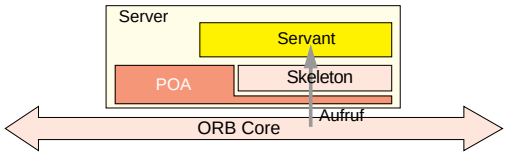
- Zentrale Komponenten eines ORB
  - mehrere interne Komponenten (teilweise nicht für Anwender sichtbar)

Portable-Object-Adaptor (POA)

- Aufgaben des Objektadapters
  - Generierung der Objektreferenzen
  - Entgegennahme eingehender Methodenaufruf-Anfragen
  - Weiterleitung von Methodenaufrufen an den entsprechenden Servant
  - Authentisierung des Aufrufers (Sicherheitsfunktionalität)
  - Aktivierung und Deaktivierung von Servants
- Portabilität von Objektimpl. zwischen verschiedenen ORBs
- Trennung von Objekt (CORBA-Objekt mit seiner Identität) und Objektimplementierung
  - eine Objektimplementierung kann **mehrere** CORBA-Objekte realisieren
  - Objektimplementierung kann bei Bedarf **dynamisch aktiviert** werden
  - persistente CORBA-Objekte (Objekte, die Laufzeit eines Servers überdauern)
  - eine Object Reference beim Client steht für ein bestimmtes CORBA Objekt – **nicht unbedingt** für einen bestimmten Servant!

Portable-Object-Adaptor (POA)

- Jeder Servant kennt seine POA-Instanz
  - POA-Instanz kennt die an ihm angemeldeten Objekte
  - POA stellt Kommunikationsplattform bereit und nimmt Aufrufe entgegen (für seine Objekte)



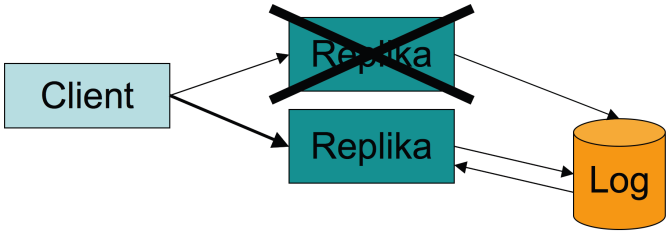
- Mehrere POA-Instanzen (mit unterschiedlichen Strategien) innerhalb eines Servers möglich
- Beispiel: Lifespan policy
  - `TRANSIENT` für Objekte, die Servant nicht überleben
  - `PERSISTENT` für Objekte, die POA und Servant überleben können

Von CORBA zu FT-CORBA

- 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



Grundlagen: Redundanz

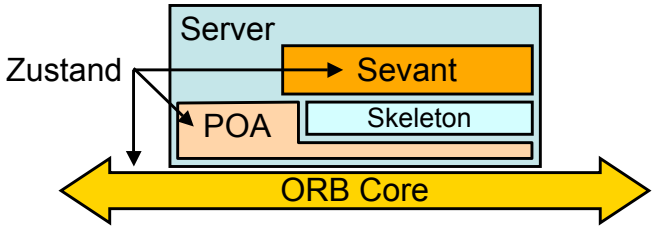
- Objekte bilden die Einheit der Replikation
- Starke Konsistenz
  - Alle Replikate haben den gleichen Zustand
  - Ermöglicht einfache Anwendungsentwicklung
    - Client interagiert mit Objekten
  - Stellt hohe Anforderungen an die Mechanismen der Infrastruktur
    - Alle Mechanismen zur Fehlertoleranz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

Grundlagen: Redundanz

- 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)



Grundlagen: Determinismus

- 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

Grundlagen: Art der Replikation

- **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

Grundlagen: Zuständigkeiten Serverseite

- 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

Grundlagen: Zuständigkeiten Clientseite

- 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 benachrichtig

Grundlagen: Kontrolle und Verwaltung

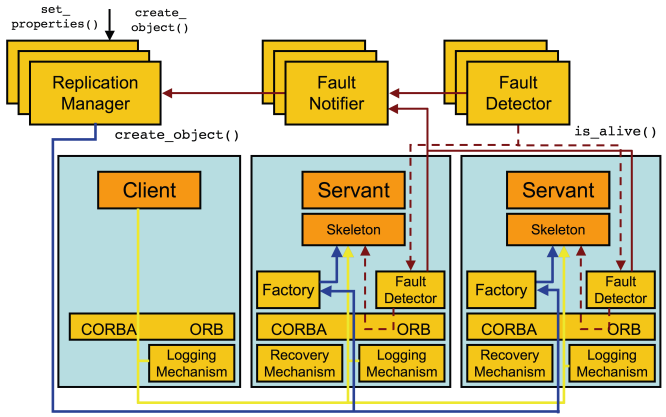
- Infrastruktur kontrolliert Fehlertoleranz
  - Automatische Erzeugung von Replikaten
  - Automatische Erhaltung der Konsistenz
- Applikation kontrolliert Fehlertoleranz (Nur in Spezialfällen nötig!)
  - Applikation lenkt die Erzeugung und Platzierung von Replikaten
  - Applikation gewährleistet die 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

Limitationen des FT-CORBA Standards

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

Es ist Aufgabe der Hersteller hier individuelle Lösungen anzubieten!

Architekturübersicht

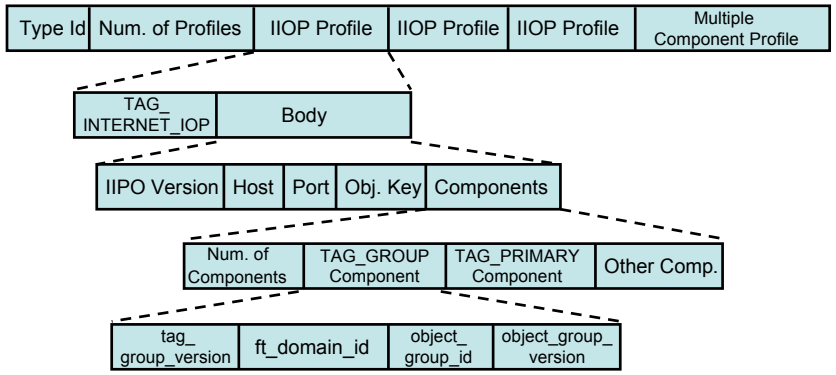


Adressierung

- Interoperable Object Group Reference (IOGR)
  - Eine IGOR besitzt mehrere IOR Profile
  - Jedes Profile setzt 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, muss nicht aktuell sein!
- Was wird adressiert?
  - direkte Addresssieung der Replikate
  - Addessierung von Gateways

Adressierung

Interoperable Object Group Reference (IOGR)



Adressierung

- 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 beigefügt
    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

Verhalten bei Ausfällen

- 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

Verhalten bei Ausfällen

- 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

Einstellung von Fehlertoleranzeigenschaften

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

Replikation

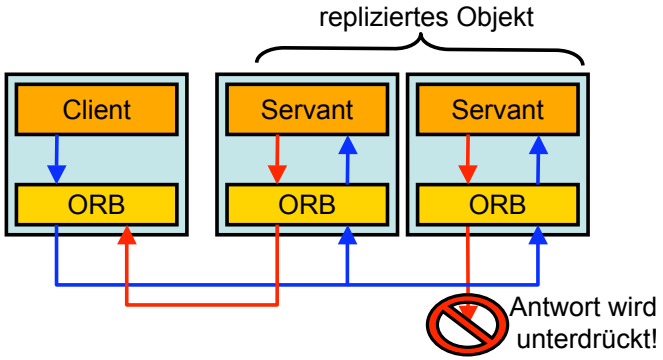
- Stateless
  - statische Daten und nur lesender Zugriff
- Cold Passive Replication
  - Wiederherstellung wird durch Sicherungspunkt und Protokollierung von Aufrufen erreicht
  - langsame Kompensation von Ausfällen
- Warm Passive Replication
  - Aktueller Zustand des primären Replikats wird periodisch 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

Active Replication

- 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

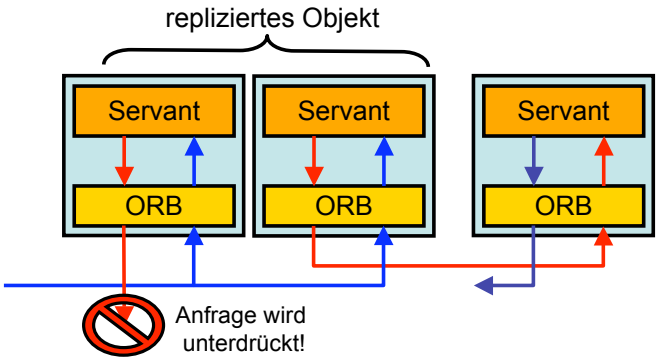
Active Replication

- 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

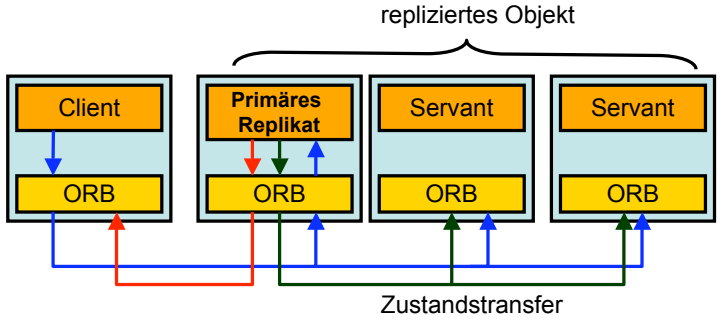


Passive Replication

- 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 dublizieren Anfragen
- Behandlung von Beitritten
  - Zustandstransfer zu neuen oder wiederhergestelltem Replikat bei warm passive Replication

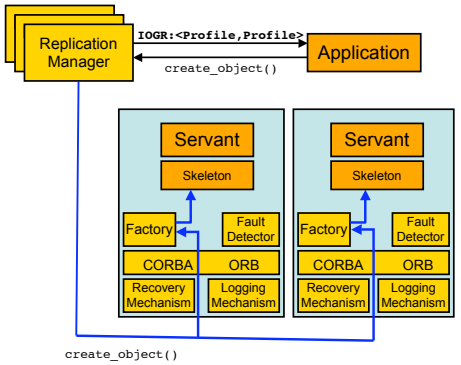
Warm Passive Replication

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



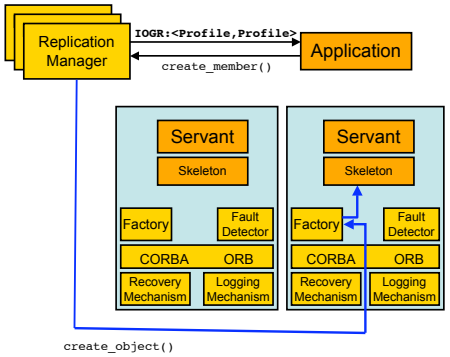
Mitgliedschaft

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



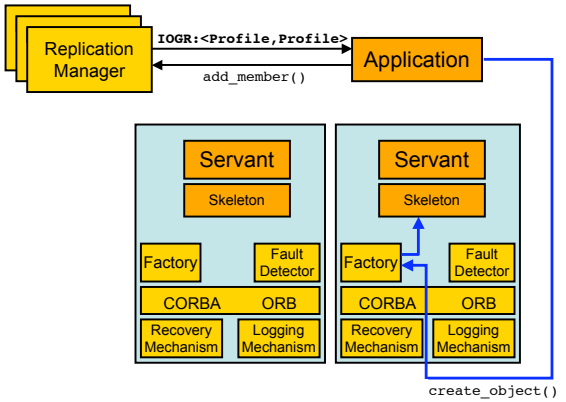
Mitgliedschaft

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



Mitgliedschaft

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



Konsistenz

- 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

Monitoring

- 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

Factories

- 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`)

Verwaltung von Replikaten

- 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

Property Manager Schnittstelle

- 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

Zeitpunkt der Konfiguration

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

Generic Factory Schnittstelle

- Wird vom Replication Manager und von Fabriken implementiert zum Erzeugen von replizierten Objekten

```
typedef Object ObjectGroup;
typedef anyFactoryCreationId;

Object create_object(
   in TypeId type_id,
   in Criteria the_criteria,
   out FactoryCreationId factory_creation_id)
raises(NoFactory, ObjectNotCreated, InvalidCriteria,
          InvalidProperty, CannotMeetCriteria);

void delete_object( in FactoryCreationId factory_creation_id)
raises(ObjectNotFound);
```

Object Group Manager

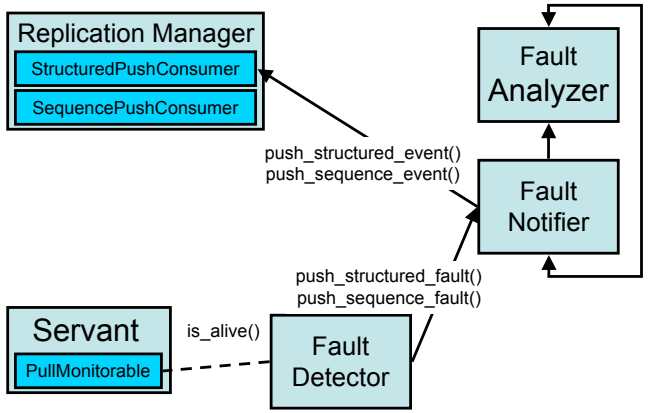
- Operationen der `Object Group Manager` Schnittstelle:
  - `create_member()`
  - `add_member()`
  - `set_primary_member()`
  - `remove_member()`
  - `locations_of_members()`
  - `get_object_group_ref()`
  - `get_object_group_id()`
  - `get_member_ref()`
- Beispiel:

```
ObjectGroup create_member(
   in ObjectGroup object_group,
   in Location the_location, in TypeId type_id,
   in Criteria the_criteria)
raises(ObjectGroupNotFound, MemberAlreadyPresent,
          NoFactory, ObjectNotCreated, InvalidCriteria
           ,...);
```

Fehlermanagement

- 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

Fehlermanagement



Fehlermanagement

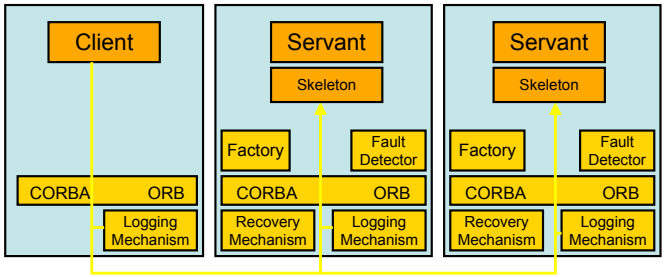
- Fehlerweitergabe durch Fault Detectors
  - einzelne Fehlerreporte (`CosNotification::StructuredEvent`)
  - Sammelreport (`CosNotification::EventBatch`)
  - Fehlertyp: (`ObjectCrashFault`)
    - `Domain_name - FT_CORBA`
    - `Type_name - ObjectCrashFault`
    - `Location - host/process`
    - `TypeId - 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

Fehlermanagement

- 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

- Schnittstelle zum Erzeugen von Sicherungspunkten

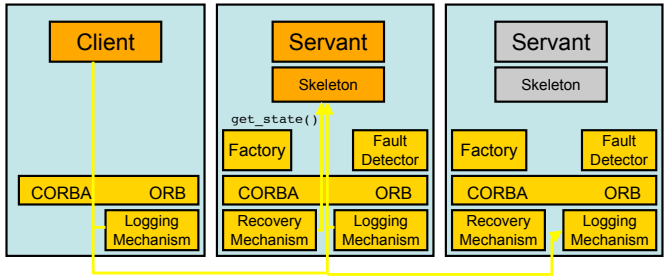
```
interface Checkpointable {
   State get_state()
      raises(NoStateAvailable);
   void set_state(in State s)
      raises(InvalidState);
};
```

- Schnittstelle zum Erzeugen von partiellen Sicherungspunkten

```
interface Updateable : Checkpointable {
   State get_update()
      raises(NoUpdateAvailable);
   void set_update(in State s)
      raises(InvalidUpdate);
};
```

Logging und Recovery

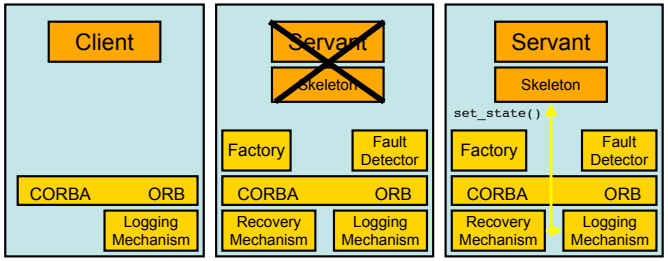
Beispiel: Cold Passive Replication — Normalbetrieb



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

Logging und Recovery

Beispiel: Cold Passive Replication — Wiederherstellung



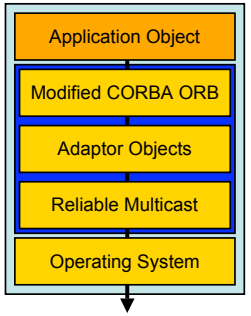
- Bei einem Ausfall wird zunächst der letzte Sicherungspunkt eingespielt und danach bereits bearbeitete aber im Sicherungspunkt noch nicht erfasste Anfragen nochmals ausgeführt
  - Nach Abschluss dieses Vorgangs können neuen Anfragen bearbeitet werden

Implementierungsvarianten

- 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 replizieren Diensten modifiziert

Integrativer Ansatz

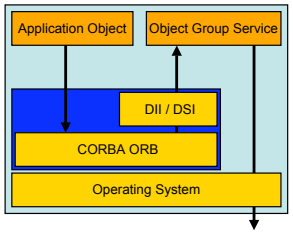
- ORB wird durch eine Gruppenkommunikation ergänzt
- IIOP wird durch ein proprietäres Protokoll ausgetauscht



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

Dienst-Ansatz

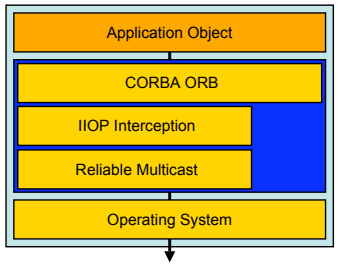
- Anwendung kommuniziert über lokale Dienste mit dem replizierten Objekt



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

Interceptor-Ansatz

- 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

Zusatzanforderungen bei der Anwendungsentwicklung

- 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

Zusammenfassung

- 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

Literatur


Object Management Group
CORBA/IIOP Specification (Chapter 23, Fault Tolerant CORBA)
OMG Technical Committee Document formal/04-03-21, 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


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


Priya Narasimhan
Fault-Tolerant CORBA: From Specification to Reality
Computer , vol. 40, no. 1, pp.110-112, Jan. 2007