AspectIX: eine Middleware-Architektur zur Unterstützung nichtfunktionaler Eigenschaften verteilter Anwendungen

Franz J. Hauck — **fjh@acm.org**

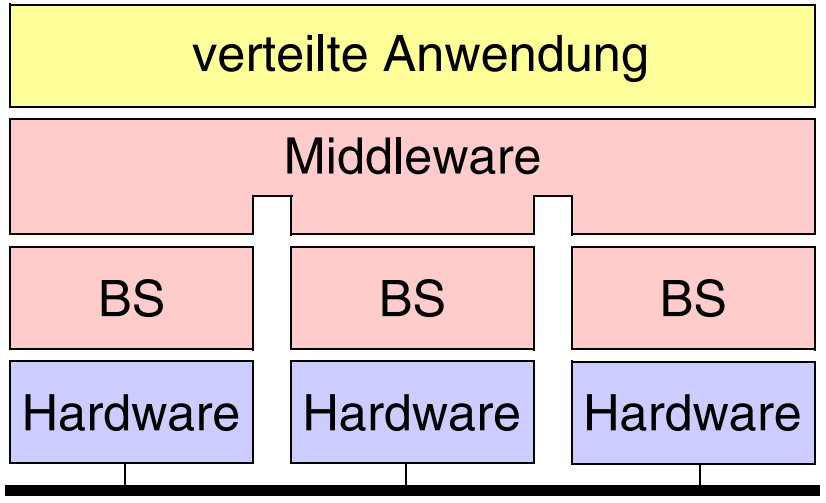
und Rüdiger Kapitza, Hans Reiser, Andreas Schmied

Verteilte Systeme
Universität Ulm

Informatik 4
Universität Erlangen-Nürnberg

1 Motivation

- ★ Systemunterstützung für verteilte Anwendungen
  - ◆ Programmiersprachen
  - ◆ Betriebssystem
  - ◆ Middleware
- ■ Unterstützung durch das System
  - ◆ Programmiermodell
  - ◆ Ablaufsystem
  - ◆ Werkzeuge zur Anwendungsentwicklung
- ■ Unterstützung für nichtfunktionaler Eigenschaften
  - ◆ Fehlertoleranz, Skalierbarkeit, Verlässlichkeit, Sicherheit, Rechtzeitigkeit, Autonomie, Mobilität ...



1 Motivation (2)

- ★ Systemunterstützung für nichtfunktionaler Eigenschaften
  - ◆ Integration in Middleware
  - ◆ Wahrung der Verteilungstransparenz
    - z.B. fehlertolerante Anwendungskomponente benutzbar wie fehleranfällige Komponente
- ■ Fokus auf objektbasierte Middleware
  - ◆ insbesondere CORBA-kompatible Systeme
  - ◆ verteilte Objekte
- ★ Ziel
  - ◆ generische Unterstützung für viele nichtfunktionale Eigenschaften

2 Überblick

- Motivation
- Systemarchitektur der AspectIX-Middleware
  - CORBA und nichtfunktionale Eigenschaften
  - fragmentierte Objekte
  - notwendige Middleware-Erweiterungen
- Anwendungsentwicklung
  - Code-Generatoren und -Transformatoren
- Zusammenfassung

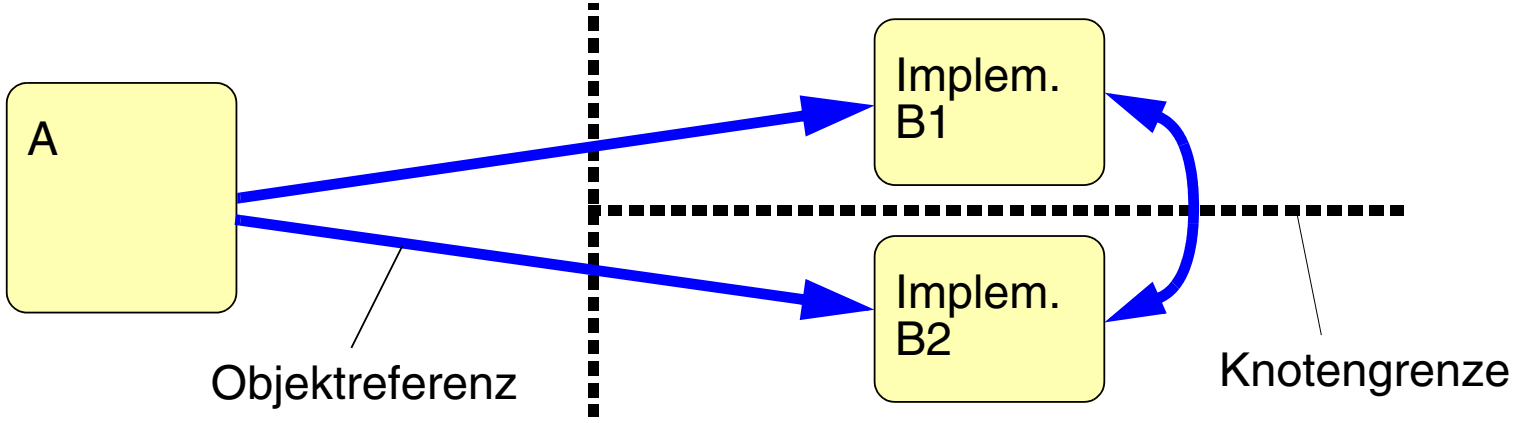
3 AspectIX-Systemarchitektur

3.1 CORBA

- Objektbasierte Anwendung wird auf mehrere Rechner verteilt
  - Bewahren der Anwendungsstruktur
  - verteilungs- und ortstransparente Interaktion (Methodenaufruf)
- ★ CORBA-Middleware sorgt für
  - eindeutige Bezeichnung/Referenzierung der Objekte
  - Lokalisierung der Objekte (Objekt jeweils auf einem Knoten)
  - Kommunikation (RPC-basierte Protokolle)
  - automatische Erzeugung von Stellvertreterobjekten aus Schnittstellenbeschreibung (IDL, Schnittstellenbeschreibungssprache)

3.2 Beispiel: Fehlertoleranz

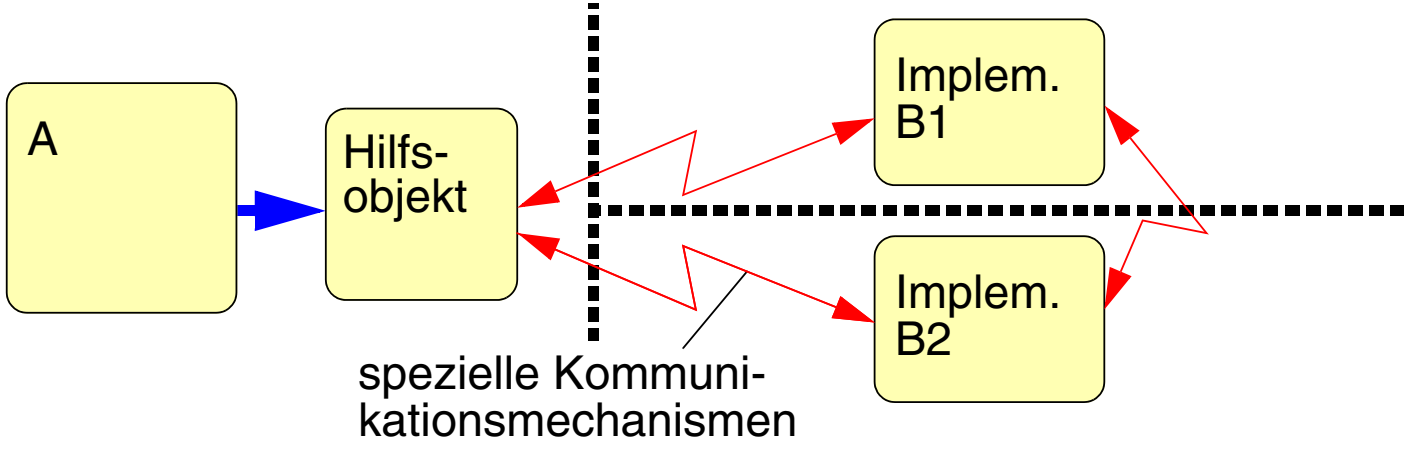
- Fehlertolerantes Objekt besteht aus mehreren verteilten Komponenten
  - redundante Datenhaltung (Replikate)
  - Objektreferenz kann nur auf eine Instanz verweisen
  - fehlertolerantes Objekt: mehrere Referenzen → keine Zugriffstransparenz



- Aufrufer sieht Replikate
  - Aufrufer realisierte Fehlertoleranz
    - z.B. B1 antwortet nicht → kommuniziere mit B2

3.2 Beispiel: Fehlertoleranz (2)

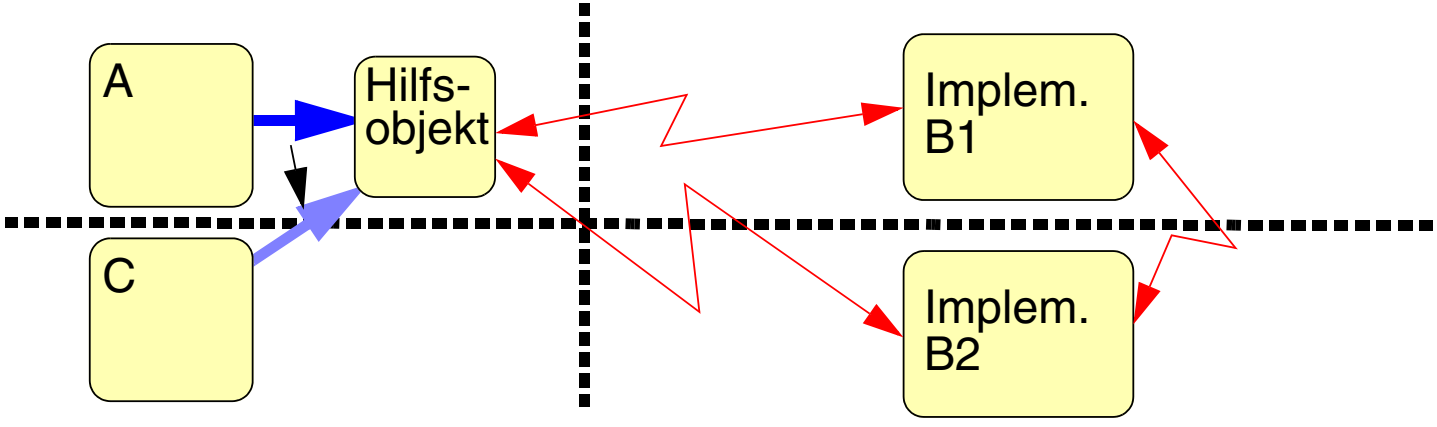
- Einführen von Hilfsobjekten



- transparenter Zugriff auf fehlertolerantes Objekt
- Fehlertoleranz gekapselt im Hilfsobjekt
- proprietäre Protokolle innerhalb des fehlertoleranten Objekts möglich
  - z.B. k-reliable Multicast

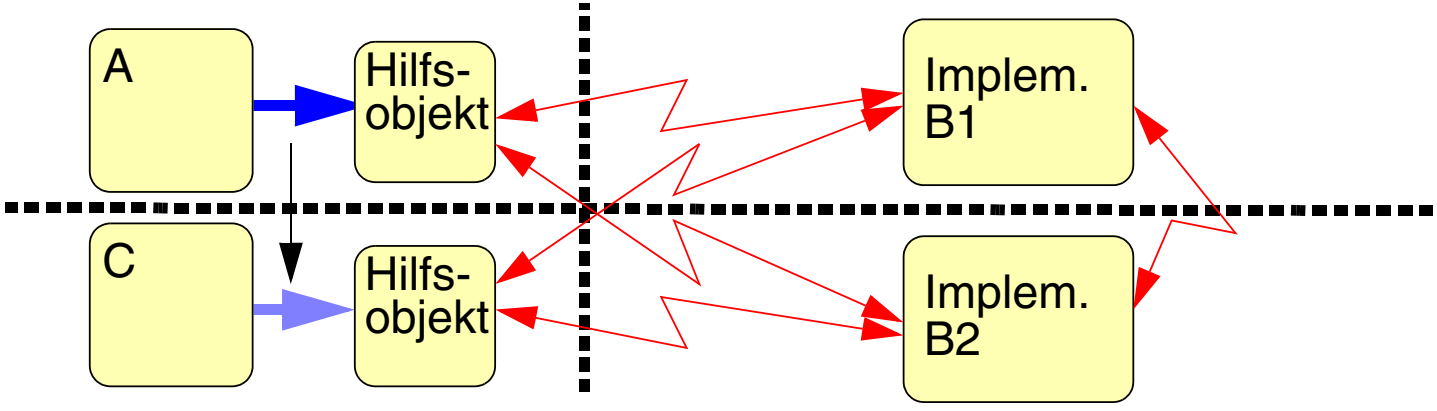
3.2 Beispiel: Fehlertoleranz (3)

- Referenzweitergabe nicht fehlertolerant
  - Referenz auf Hilfsobjekt wird weitergegeben



3.2 Beispiel: Fehlertoleranz (4)

- Erzeugung eines jeweils lokalen Hilfsobjekts



- Erzeugung nicht zugriffstransparent
  - z.B. Referenz als Ergebnis eines Methodenaufrufs nicht sofort nutzbar

3.2 Beispiel: Fehlertoleranz (5)

★ Abhilfe

- Integration in CORBA bzw. Middleware
- FT-CORBA: Objektreferenz kann auf Objektgruppe verweisen
- Hilfsobjekt ist spezielles, automatisch generiertes und transparent instanziiertes Stubobjekt

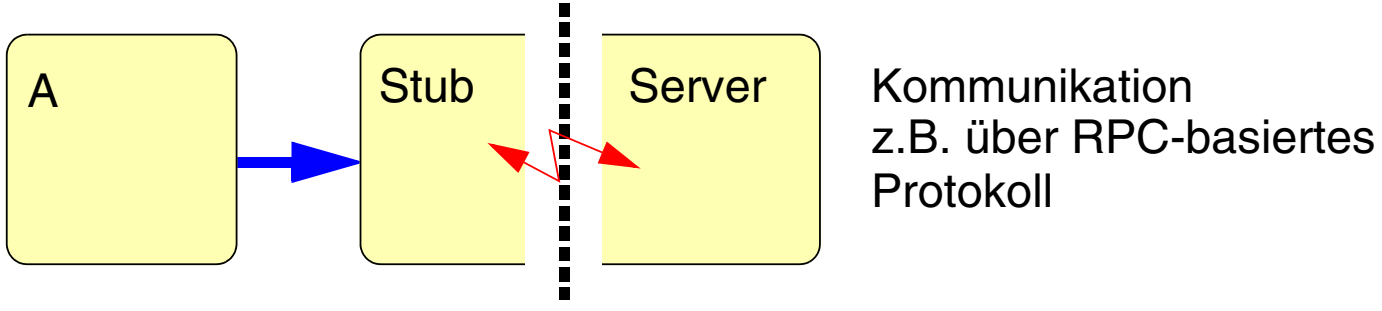
▲ Problem

- Integration von Skalierbarkeit?
- Integration von Multimedia-Datenübertragung?
- Integration von ...

- generische Unterstützung für nichtfunktionale Eigenschaften nicht möglich

3.3 Fragmentierte Objekte

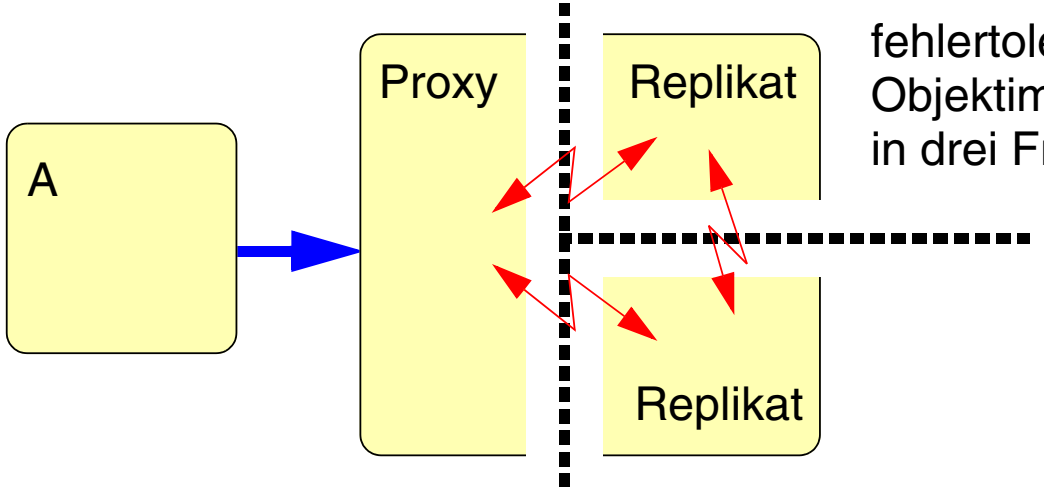
- Verteiltes Objekt
  - besteht aus verteilten Fragmenten
  - überall dort, wo lokale Referenz auf verteiltes Objekt existiert, verweist diese auf ein lokales Fragment
  - es gibt keine Stellvertreterobjekte
  - Fragmente kommunizieren miteinander
- Beispiel: Client-Server-Objekt
  - Serverfragmenttyp (einmal pro verteiltes Objekt instanziiert)
  - Stubfragmenttyp (beliebig häufig pro verteiltes Objekt instanziiert)



3.3 Fragmentierte Objekte (2)

- Beispiel: fehlertolerantes Objekt
  - Replikatfragmenttyp (mehrfach pro Objekt instanziiert)
  - Proxyfragmenttyp (beliebig häufig pro Objekt instanziiert)

  - Proxy spricht Replikate an
  - Transparenz der Fehler und der Fehlertoleranz



3.3 Fragmentierte Objekte (3)

- Weitere Beispiele:
  - Multimedia-Strom-Server: Server-, Proxy-Fragment
    - z.B. Radiodienst
  - Mobiler Agent: mobiles Agentenfragment, Proxy-Fragment
  - skalierbarer hierarchischer Dienst: Fragmente für Knoten in der Hierarchie
    - z.B. DNS oder WWW als fragmentiertes Objekt

3.4 Erweiterungen gegenüber CORBA

- Repräsentation von Objektreferenzen in CORBA
  - IOR, *Interoperable Object Reference*
  - codiert alle wichtigen Informationen über die Objektreferenz
  - im Standardfall: Hostname, Portnummer, lokale Objekt-ID für das IIOP-Protokoll *(Internet Inter ORB Protocol)*
  - Profile erlauben Einfügen beliebiger Informationen
- ★ Instanziierung eines lokalen Fragments
  - falls Referenz lokal bekannt wird und noch nicht bekannt ist
  - Instanziierung ist objektspezifisch (statt typspezifisch im Falle eines Stubs)
- ★ Kommunikationsunterstützung für die Fragmente
  - Fragmente müssen kommunizieren
  - Fragmente müssen sich kennen

3.4 Erweiterungen gegenüber CORBA (2)

- Codierung des Fragment-Implementierungstyps in der IOR
  - Standard-CORBA: IDL-Typ, für den ein Stub gebraucht wird
  - Statisch: Klassenname oder eindeutige Implementierungsbezeichnung zur Erzeugung eines Fragments
    - z.B. Java ClassName evtl. inklusive einer CodeBase zum dynamischen Laden
    - z.B. DLS-Name für den AspectIX Dynamic-Loading-Service
  - Dynamisch: Entscheidungsbezeichner für externen Entscheidungsdienst
    - z.B. Bezeichner für AspectIX Policy-Decision-Service, der als Ergebnis einen DLS-Namen liefert
    - Entscheidung kann von lokalen Gegebenheiten abhängig gemacht werden, z.B. Vertrauenswürdigkeit, Leistungsparameter
    - Entscheidung kann über die Laufzeit veränderlich sein, z.B. bei Software-Aktualisierungen

3.4 Erweiterungen gegenüber CORBA (3)

- Kommunikationsmittel
  - Middleware stellt Kommunikationsprimitive bereit
    - z.B. AspectIX Communication End Points
- Bekanntmachung der Kommunikationsadressen
  - Standard-CORBA: IIOP-Adresse für Initialisierung des Stubs
  - Client-Server: Kommunikationsadresse für beliebiges Protokoll wird codiert
    - z.B. RTP-Adresse, IP-Multicast-Channel-Adresse
  - Peer-to-Peer: eindeutige Objekt-ID wird codiert; externer Ortsdienst löst Kommunikationsadressen aller Fragmente eines Objekts auf
    - z.B. für mobile Objekte
    - z.B. für fehlertolerante und skalierbare Objekte

4 Anwendungsentwicklung

- ★ Kapselung bestimmter nichtfunktionaler Eigenschaften hinter Objektschnittstelle nun möglich
- ▲ Aufgaben der Anwendungsentwicklung
  - ◆ mehrere Fragmenttypen entwickeln
  - ◆ deren Interaktionen bestimmen und selbst ausprogrammieren
  - ◆ externe Dienste initialisieren
    - statisch, z.B. Namensdienst, DLS, Entscheidungsdienst
    - dynamisch, z.B. Ortsdienst
- ■ Unterstützung der Entwicklung bei häufigen und automatisierbaren Einzelschritten
  - ◆ ADK, Application Development Kit
  - ◆ Code-Generator, -Transformator

4 Anwendungsentwicklung (2)

- Ziel des ADK
  - automatische Fragmentgenerierung für bestimmte nichtfunktionale Eigenschaften
  - Eingabe
    - normale CORBA Servant-Implementierung
    - zusätzliche Informationen, z.B. Angaben über lesende, schreibende Methoden oder über wichtige Datenvariablen (für Fehlertoleranz)
- Aufbau des ADK
  - liest Menge von Java-Klassen ein
  - erlaubt elementare Transformationen auf diesen Klassen
  - komplexe Transformation wird durch hierarchische, modularisierte, wiederverwendbare Transformationsobjekte beschrieben (Weavelets)

5 Status

- Implementierungen in Java
  - ORB-Core
  - Communication-End-Points für IIOP
  - DLS, Dynamic-Loading-Service
  - PDS, Policy-Decision-Service
  - IDLflex, IDL-Compiler (jetzt in ADK integriert)
  - ADK
- Integration der Implementierungen noch nicht vollständig abgeschlossen
- Gegenstand weiterer Forschungsarbeiten
  - ADK
  - Anwendung nichtfunktional Eigenschaften: Fehlertoleranz, Autonomie von Diensten

6 Zusammenfassung und Bewertung

- AspectIX
  - fragmentierte Objekte unterstützen viele nichtfunktionale Eigenschaften in generischer Weise
    - z.B. Fehlertoleranz, Skalierbarkeit, Mobilität, Autonomie, ...
    - nicht ohne Weiteres geeignet für harte Anforderungen an Effizienz, Scheduling u.ä. (Unterstützung des Betriebssystems notwendig)
  - beliebige objektinterne Interaktionsmuster
  - Verteilungstransparenz bleibt gewahrt
  - interoperabel mit Standard-CORBA-Anwendungen
  - ADK kann Entwicklungsaufwand reduzieren
- ★ Mehr dazu unter **`http://www.aspectix.org`**