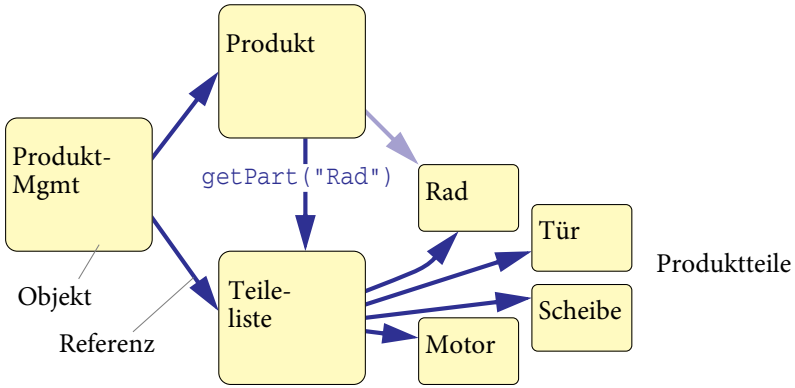
Überblick

Fragmentierte Objekte
- Monolithische verteilte Objekte
- Fragmentierte verteilte Objekte
- FORMI – Fragmentierte Objekte und Java

Übung basiert in Teilen auf Unterlagen von Prof. Franz Hauck (Universität Ulm)

Verteiltes objektbasiertes Programmiermodell

- Anwendungen bestehen aus einer Menge von kooperierenden Objekten



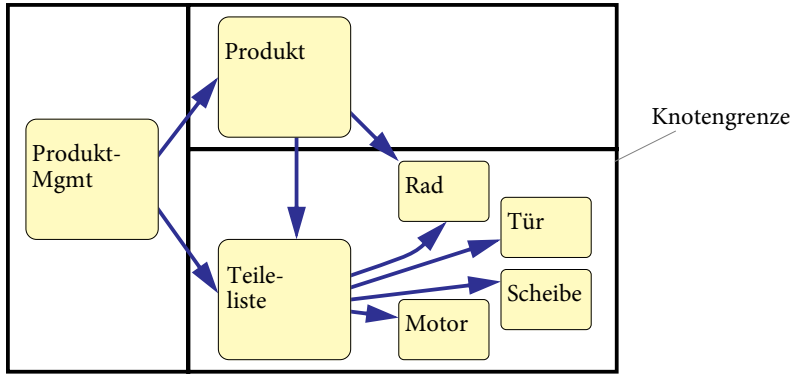
- Interaktion über Methodenaufrufe
- Austausch und Weitergabe von Objektreferenzen

Übergang zur Verteilung

- Objektbasierte Anwendung
  - Beibehalten der Designstruktur
  - Identifikation von Verteilungseinheiten
  - konkrete Verteilung für einen Anwendungslauf
- Wünschenswerte Transparenz
  - zugriffstransparenter Methodenaufruf
  - zugriffstransparente Parameterübergabe
  - ortstransparenter Methodenaufruf

Monolithisches Objektmodell

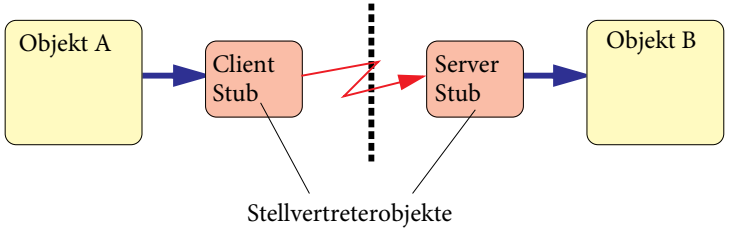
- Vorgehensweise bei Java RMI
  - Beispiel einer Verteilung auf drei Knoten:



  - Ein Objekt **liegt immer vollständig** auf einem Knoten (monolithisches Objekt).
  - Objektreferenzen sind lokal oder entfernt

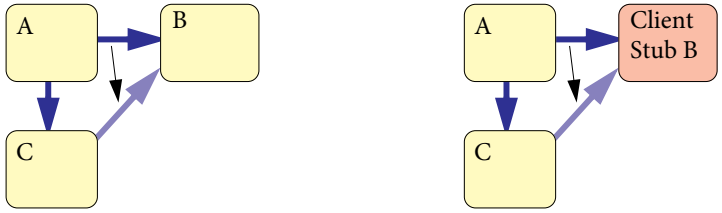
Entfernte Referenzen

- Lokales Stellvertreterobjekt
  - lokale Referenz auf Stellvertreterobjekt entspricht entfernter Referenz (Zugriffstransparenz)
  - Stellvertreter besitzt Stubmethode für jede Objektmethode
    - Marshalling, Unmarshalling
    - Kommunikation mit verteiltem Objekt
  - Stellvertreter kennt aktuelle Kommunikationsadresse des verteilten Objekts (Ortstransparenz) und das Aufrufprotokoll



Referenzweitergabe

- Lokale Weitergabe
  - lokale Sprachreferenz wird immer weitergegeben



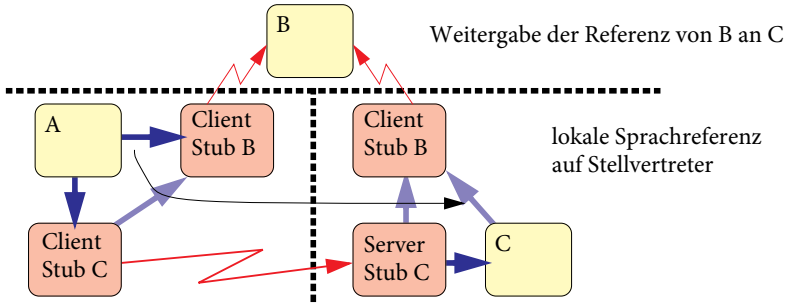
! Semantik für lokale und entfernte Objektreferenzen gleich

Referenzweitergabe – Marshalling

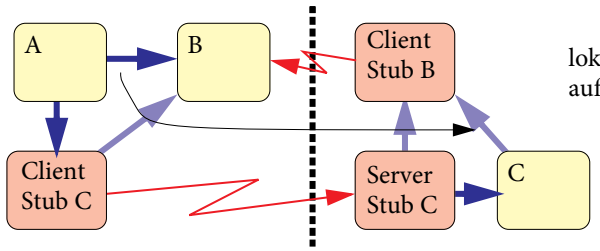
- *Entfernte* Weitergabe eines entfernten Objektes
  1. lokale Referenz auf ein Stellvertreterobjekt
  2. lokale Referenz auf ein lokal liegendes verteiltes Objekt
  - Java RMI: Serialisierung des Stubobjekts
    - enthält eindeutige Kommunikationsadresse und lokal gültigen Objektidentifikator
    - enthält Aufrufprotokollimplementierung
  - allgemein: Marshalling eines global gültigen Objektbezeichners (z.B. eindeutige Kommunikationsadresse und lokale Objekt-ID o.ä.)
- *Entfernte* Weitergabe einer lokalen Referenz auf ein nichtverteiltes Objekt
  - Java RMI: Kopie serialisierbarer Objekte
  - allgemein: systemabhängiges Verhalten: z.B. 'nicht möglich' oder 'wird kopiert'

Referenzweitergabe – Unmarshalling

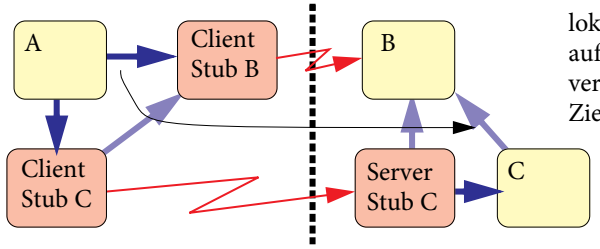
- Entfernte Weitergabe
  - aus übertragener Information muss Objektreferenz abgeleitet werden
  - Erzeugen eines neuen Stellvertreters
  - Referenz auf bereits bekannten Stellvertreter
  - Referenz auf lokal liegendes verteiltes Objekt (optional)
- Beispiele:



Referenzweitergabe



lokale Sprachreferenz auf verteiltes Objekt



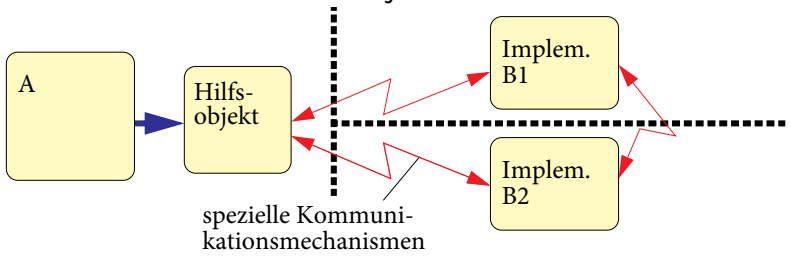
lokale Sprachreferenz auf Stellvertreter verteiltes Objekt am Zielknoten lokal

Probleme des monolithischen Ansatzes

- Fixierung auf RPC-basierte Kommunikation
  - Aufrufe werden transparent bis zum verteilten Objekt geleitet
- Zu starr für Anwendungen mit speziellen Anforderungen
  - Beispiel: Fehlertoleranz durch aktive Replikation
    - Objektreferenz verweist auf mehrere verteilte Einheiten
    - evtl. Multicast-Kommunikation
  - Beispiel: Multimedia-Server
    - stromorientierte Kommunikation evtl. mit Bandbreitengarantie
    - z.B. Live-Radiodienst

Fehlertoleranz durch aktive Replikation

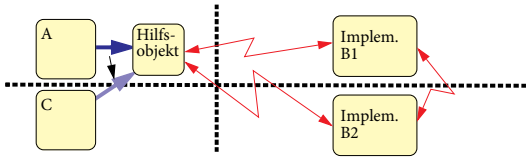
- Monolithischer Ansatz
  - Objektreferenz kann nur auf eine Instanz eines verteilten Objekts verweisen
  - fehlertolerantes Objekt: mehrere Referenzen
    → **keine Zugriffstransparenz**
- Abhilfe: Einführen von Hilfsobjekten



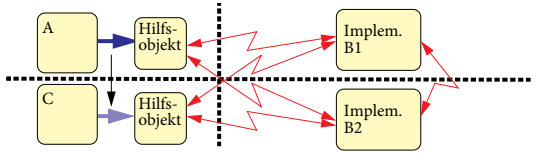
  - transparenter Zugriff auf fehlertolerantes Objekt
  - Fehlertoleranz gekapselt im Hilfsobjekt (Fassade)

Fehlertoleranz durch aktive Replikation

- Fehlertoleranz bei Referenzweitergabe?!
  - Weitergabe der Referenz auf lokales Hilfsobjekt problematisch
  - Verlust der Fehlertoleranz!



- Abhilfe: Erzeugung des Hilfsobjekts
  - bei Referenzweitergabe neues Hilfsobjekt beim Empfänger



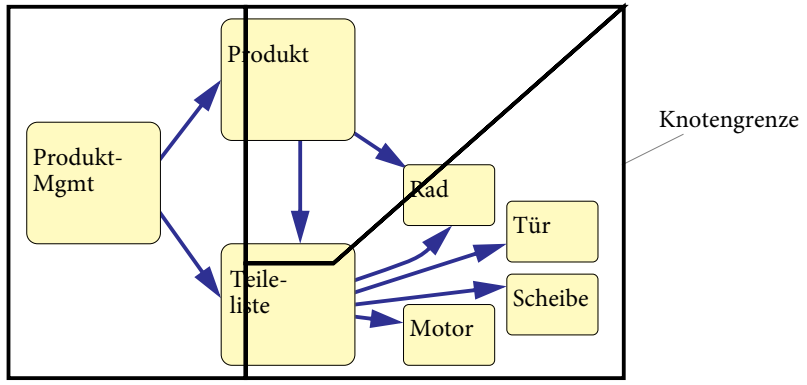
  - Erzeugung eines weiteren lokalen Hilfsobjekts nicht zugriffstransparent
    ! als Ergebnis bzw. Parameters eines Methodenaufrufs nicht transparent nutzbar

Fehlertoleranz durch aktive Replikation

- Mögliche Lösungen
  - Integration in Middleware (vgl. FT CORBA)
    - Einführung fehlertoleranter Objekte mit speziellen Kommunikationsmustern
    - automatische Generierung von speziellen Stellvertretern
    - vgl. auch Erfahrungen aus der Übung Aufgabe 4
- Problem
  - verschiedene Varianten von Fehlertoleranz bedürfen verschiedener Integrationen

Fragmentierter Ansatz

- Grundidee: **Verteilungsschnitte durch Objekte**



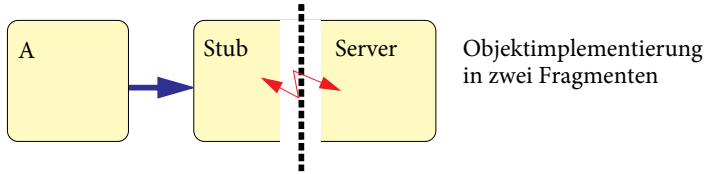
  - Referenzen sind immer lokal
  - verteilte Objekte sind an mehreren Orten präsent (Fragmente)

Referenzen

- Beobachtung
  - Verteiltes Objekt ist überall dort durch Fragment präsent, wo lokale Referenzen auf es existieren
  - es existieren keine entfernten Referenzen
- Zugriffstransparenz
  - immer lokaler Aufruf im lokalen Fragment
  - Kommunikation zwischen den Fragmenten ist gekapselt, d.h. hinter Schnittstelle verborgen
- Ortstransparenz
  - lokales Fragment kennt andere Fragmente
  - Client braucht deren Orte nicht zu wissen

Client-Server-Objekt

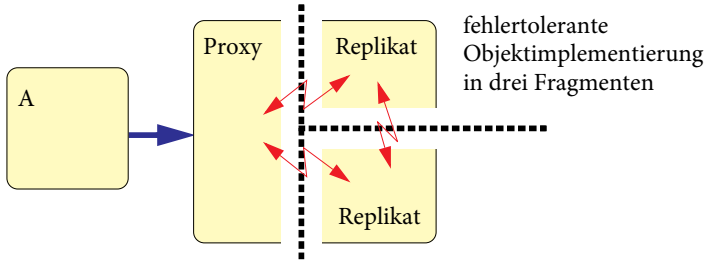
- Zwei Fragmenttypen
  - Serverfragment (einmal vorhanden)
  - Stubfragment (beliebig häufig vorhanden
- Interaktion
  - Stubfragment führt RPC-basierte Kommunikation mit Serverfragment durch (ähnlich Stellvertreterobjekte)



  - identisches Vorgehen wie beim monolithischen Ansatz

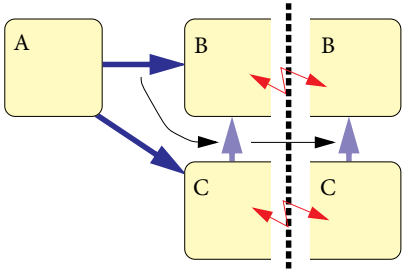
Fehlertolerantes Objekt

- Zwei Fragmenttypen
  - Replikatfragment (mehrfach vorhanden)
  - Proxyfragment (beliebig häufig vorhanden)
- Interaktion
  - Proxy spricht Replikate an
  - Transparenz der Fehler und der Fehlertoleranz



Referenzweitergabe

- Übergabe einer lokalen Referenz an das lokale Fragment



- Kommunikation zwischen Fragmenten
  - Ableiten des Objektbezeichners aus lokaler Referenz
  - Übertragung des Objektbezeichners
  - Erzeugen eines neuen lokalen Fragments auf der Empfängerseite

Erzeugung lokaler Fragmente

- Ableitung des lokalen Fragments aus Objektbezeichner
  - lokales Fragment ist **objektspezifisch nicht schnittstellenspezifisch** (!)
  - direkte Kodierung der Fragmentimplementierung im Objektbezeichner (statisch)
    - z.B. Klassenname oder „Smart Proxy“
  - Befragung eines externen Dienstes, welche Implementierung instanziiert werden soll (dynamisch)
    - z.B. für Anpassung des Objektverhaltens an Vertrauenswürdigkeit, Bandbreiten, Leistungsfähigkeit o.ä. des lokalen Rechensystems
  - evtl. dynamisches Laden von Code notwendig
- Bestehendes lokales Fragment
  - lokale Referenz auf bestehendes Fragment kann verwendet werden
    - interne Tabelle listet Fragmente und deren Objektbezeichner
  - neues Fragment wird erzeugt
    - mehrere lokale Fragmente zum selben Objekt

Kommunikation zwischen Fragmenten

- Fragmente eines Objekts sollten sich kennen
  - Kodierung von Kommunikationsadressen im Objektbezeichner (statisch)
  - Ortsdienst (Location Service)
    - liefert über Objektbezeichner Kommunikationsadressen laufender Fragmente (dynamisch)
    - Fragmente registrieren sich im Ortsdienst

Probleme des Fragmentierten Ansatzes

- Komplexeres Programmiermodell
  - Zugriffs- und Ortstransparenz von außen
  - keine automatische Kommunikation innen
  - Entwickler muss Kommunikation selbst implementieren
    - Implementierung mehrere zusammenarbeitender Fragmentimplementierung
    - Initialisierung von Ortsdienst, Diensten für dynamische Code-Laden etc

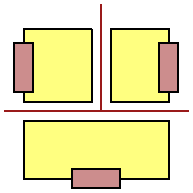
⇒ Unterstützung durch Middleware
- Vereinfachung häufiger Entwicklungsprozesse durch Automatisierung
- Code-Generatoren
  - z.B. für RPC-basierte Fragmentkommunikation

Anforderungen an Middlewaresysteme

- Objektbasierte Middleware mit fragmentiertem Objektmodell
  - weltweit eindeutige Objektbezeichner zum Parametertransport zwischen Fragmenten
    - Erzeugung lokaler Fragmente aus Objektbezeichner
    - evtl. dynamisches Laden von Fragment-Code
  - Erzeugung neuer verteilter Objekte
  - Namensdienst zum Finden von Objekten
  - Ortsdienst zum Finden von Fragmenten
  - Kommunikationsdienste für Interfragmentkommunikation
    - Implementierung von adäquaten Protokollen z.B. RPC-basierte Aufrufprotokolle
  - Code-Generatoren zur Unterstützung der Fragmententwicklung

FORMI

- Fragmentierte Objekte in Java
  - RMI-kompatibel
    - Referenzen auf FORMI-Objekte können transparent wie RMI-Objekte weitergegeben werden
    - z.B. in Registry



- Initiales Design im Rahmen einer Masterarbeit (2004)
- Mittlerweile in der vierten überarbeiteten Variante
- Aktueller Stand: `http://formi.git.sourceforge.net/git`

FORMI

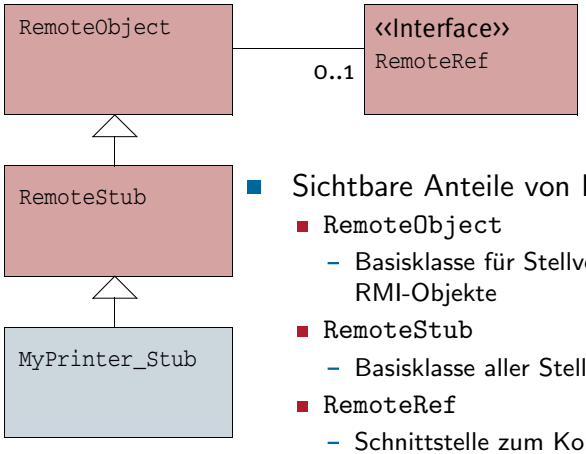
- Anforderungen
  - Koexistenz von RMI und fragmentierten FORMI Objekten
    - Ermöglich den Einsatz von fragmentierten Objekten nur dann wenn erforderlich
  - Erhalt der Zugriffstransparenz
    - FORMI Objekte können an RMI Objekte übergeben werden
    - RMI Objekte können an FORMI Objekte übergeben werden
    - FORMI Objekte können an FORMI Objekte übergeben werden
  - Unterstützung als generische Middlewareschicht in Ergänzung zu RMI
    - dabei werden verschiedene Anwendungszenarien berücksichtigt

Entwicklungsziele

- Zugriffstransparenz
  - FORMI Fragment verhält sich wie ein RMI Stellvertreterobjekt
    - D.h. ein FORMI Fragment kann wie ein RMI Stellvertreterobjekt serialisiert werden (dies schließt auch die codebase ein)
  - Wird ein FORMI Fragment serialisiert wird auch am Zielort ein FORMI Fragment deserialisiert
    - es muss aber kein neues Fragment dabei entstehen wenn schon eines am Zielort existiert
    - es muss aber nicht den gleichen Objekttyp haben
    - Beide Entscheidungen hängen vom Objekt und der Umgebung ab
- Bedarfsgetriebene Erzeugung von Fragmenten
  - Ein Fragment wird nur dann erzeugt wenn es lokal auch benötigt wird
    - Nützlich zum Beispiel bei der Verwendung eines Namensdienstes

Design

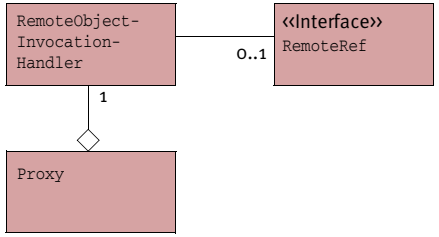
- Aufbau eines lokalen RMI Stellvertreterobjekts (hier die statisch generierte Variante)



  - Sichtbare Anteile von RMI
    - `RemoteObject`
      - Basisklasse für Stellvertreterobjekte und RMI-Objekte
    - `RemoteStub`
      - Basisklasse aller Stellvertreterobjekte
    - `RemoteRef`
      - Schnittstelle zum Kommunikationsprotokoll

Design

- Aufbau eines lokalen RMI Stellvertreterobjekts (hier die dynamische Variante)



- Stellvertreterobjekt
  - Dynamische generierte Instanz ein Unterklasse von `java.lang.reflect.Proxy`
  - `java.rmi.server.RemoteObjectInvocationHandler`
    - Behandelt Aufrufe am Proxy

Design

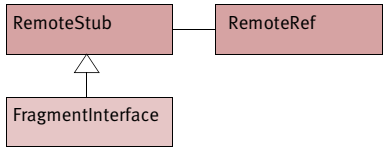
- Abläufe innerhalb von RMI
  - Aufruf einer Methode des Stellvertreterobjekts
    - Umwandlung in einen generischen Aufruf
    - Stellvertreterobjekts ruft `invoke()`-Methode der `RemoteRef`-Instanz auf
  - Marshalling
    - Ist RMI Objektreferenz ist vom Typ `RemoteStub` so erfolgt direkt eine Serialisierung
    - Zeigt RMI Objektreferenz auf ein exportiertes Objekt so wird ein Stellvertreterobjekt erzeugt und serialisiert
  - Unmarshalling
    - Deserialisierung von Stellvertreterobjekten

Aufbau eines FORMI Fragments

- Anforderungen
  - Austausch des Fragmenttyps
    - Austausch soll zur Laufzeit erfolgen können
    - Erzwingt eine Indirektionsstufe damit Referenz auf das Fragment für Klienten konstant bleibt

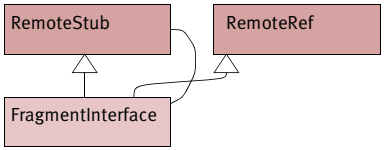
    ⇒ Trennung von *Fragmentschnittstelle* und *Implementierung*
  - Austausch des Fragmenttyps soll zur Laufzeit entschieden werden
    - Es wird eine statische Instanz (*Fragment Manager*) im Fragment benötigt

Aufbau eines FORMI Fragments



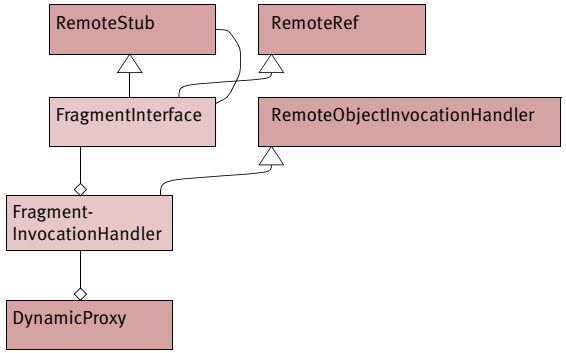
- `FragmentInterface` Basisklasse für alle Framentschnittstellen
  - Schnittstelle zum Klient
  - Erbt von `RemoteStub` und ist damit zu RMI kompatibel

Aufbau eines FORMI Fragments



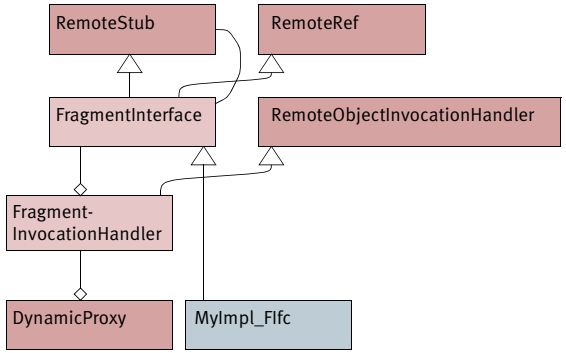
- `FragmentInterface` Basisklasse für alle Framentschnittstellen
  - Schnittstelle zum Klient
  - Erbt von `RemoteStub` und ist damit zu RMI kompatibel
  - Realisiert gleichzeitig die `RemoteRef`-Rolle

Aufbau eines FORMI Fragments



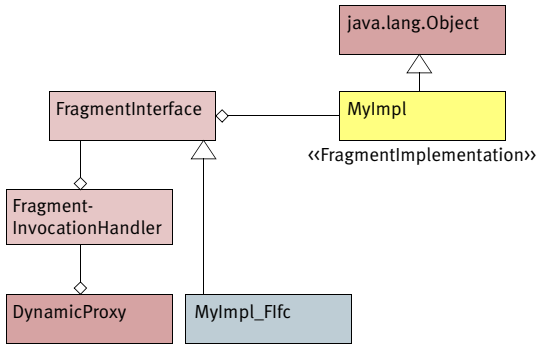
- Für Typ-spezifische Schnittstelle wird ein dynamischer Proxy erstellt
- Spezieller Handler für Konformität zu RMI

Aufbau eines FORMI Fragments



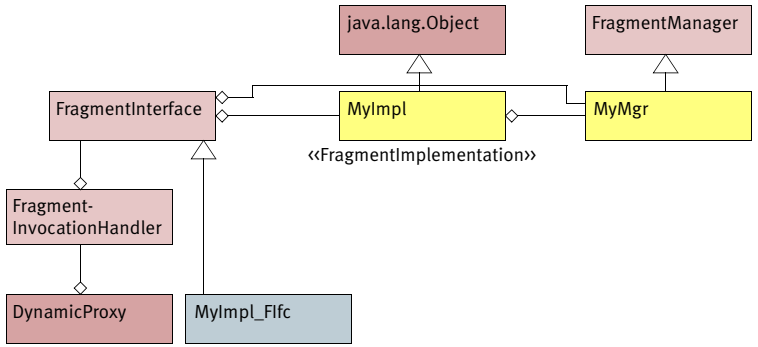
- Alternative: Statisch erzeugtes Fragmentinterface

Aufbau eines FORMI Fragments



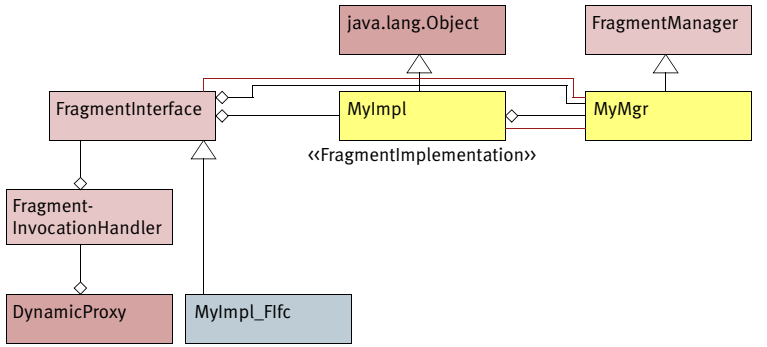
- Fragmentimplementierung
  - Spezifischer, lokaler Code des verteilten Objektes

Aufbau eines FORMI Fragments



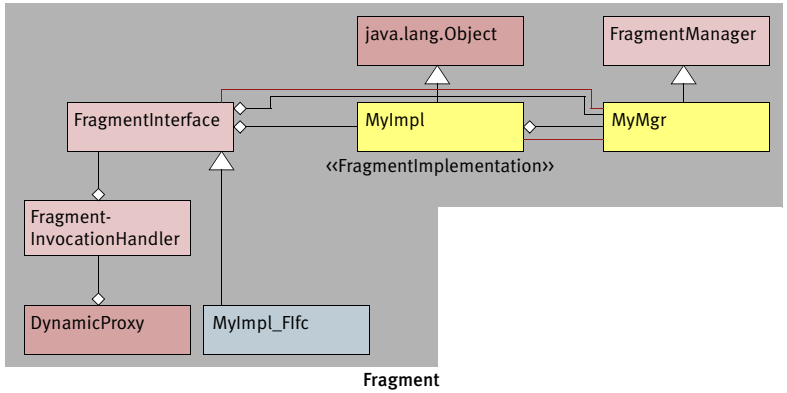
- Fragment Manager
  - Implementiert `Factory`-Muster zur Erzeugung von Fragmentimplementierungen
  - Koordiniert den Austausch von Implementierungen

Aufbau eines FORMI Fragments



- Fragment Manager
  - hat Zugriff auf alle FIfcs und FImpls
  - verlinkt durch `WeakReferences` ermöglicht *garbage collection* wenn nicht mehr von Klients benötigt

Aufbau eines FORMI Fragments



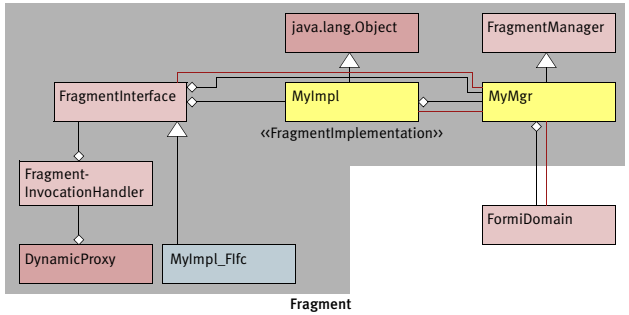
- Fragment Manager
  - Verwaltet eindeutigen Identifikator (UUID) für das FO
  - Verwaltung von Kommunikationsinformationen
  - Logik für die Auswahl, Erzeugung und Initialisierung von FImpls

Serialisierung von FORMI Fragmenten

- Es werden das Fragmentinterface und der Fragment Manager serialisiert
  - Fragment Manager ist besonders wichtig da er Informationen über Kommunikationsadressen verwaltet
- Fragmentimplementierungen sind explizit von der Serialisierung ausgeschlossen (`transient`)

Deserialisierung von FORMI Fragmenten

- Alle serialisierten Objekte werden wieder ausgepackt dabei wird zuerst der Frament Manager bearbeitet
- Zur gemeinsamen Nutzung von Fragmenten gibt es eine zentrale Verwaltungseinheit vom Typ `FormiDomain` in jeder FORMI JVM
  - diese wird automatisch bei Bedarf erzeugt (**Singelton**-Muster)
  - kennt alle fragmentierten Objekte an Hand ihrer UUID
  - kennt alle Fragment Manager zu einem Objekt und kann neue Fragment Referenzen an eine Standardimplementierung zuweisen



Erzeugung von FORMI Objekten

- Zwei Ansätze vorstellbar
  - Erzeugung einer initialen Fragmentimplementierung
  - Erzeugung eines initialen Fragment Managers
- Beispiel: Erzeugung einer initialen Fragmentimplementierung
  - Im Verlauf der Erzeugung wird der Manager erstellt und etwaige initiale Kommunikationsparameter an den Manager übergeben
  - Erhalt einer Referenz auf das fragmentierte Objekt durch die `getReference()`-Methode des Fragment Managers

Kommunikation zwischen Fragmenten

- Beliebige interne Kommunikation möglich
  - Java Sockets, JGroups, RMI, etc.
- Lokalisierung anderer Fragmente
  - Bisher kein dedizierter Ortsdienst vorhanden
  - RMI Registry kann genutzt werden um Fragmente zu finden

Fazit

- FORMI ist eine kleine, generische Middleware für fragmentierte Objekte
  - Es ermöglicht die Verwendung beliebiger Kommunikationsmechanismen
  - Code und Zustand eines Objektes können frei verteilt werden
  - Einfach kombinierbar mit RMI
  - Unterstützt das dynamische Laden von Code wie RMI