Middleware - Übung

Tobias Distler, Michael Gernoth, Rüdiger Kapitza

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

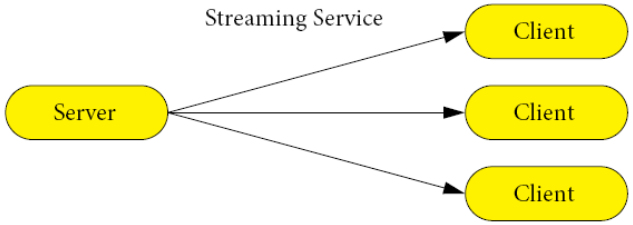
Wintersemester 2009/2010

Überblick

FORMI
- Fragmentierte Objekte
- RMI revisited
- Fragmented Object RMI

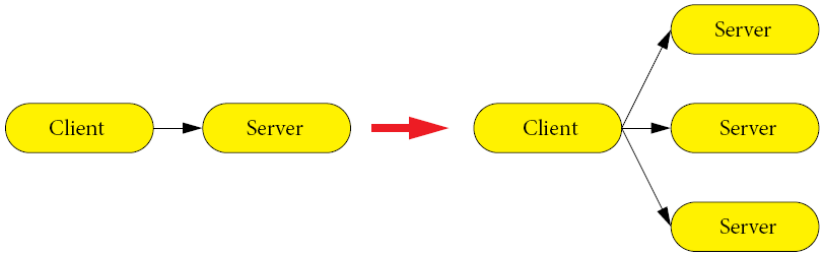
Motivation

- Standard-Middleware für objektorientierte Anwendungen
  - CORBA, .NET-Remoting
  - Java Remote Method Invocation (RMI)
- **Problem**: Vielzahl von Anwendungen mit nicht-funktionalen Anforderungen
  - Weiche Echtzeit
  - Fehlertoleranz
  - Hohe Verfügbarkeit



Motivation

- Java RMI kann erweitert werden
  - Neue Aufrufsemantiken und Transportprotokolle
  - Beispiel: Realisierung von Objektgruppen für Fehlertoleranz



- Mögliche Anpassungen sind jedoch eingeschränkt

→ **Lösung**: Fragmentiertes Objektmodell

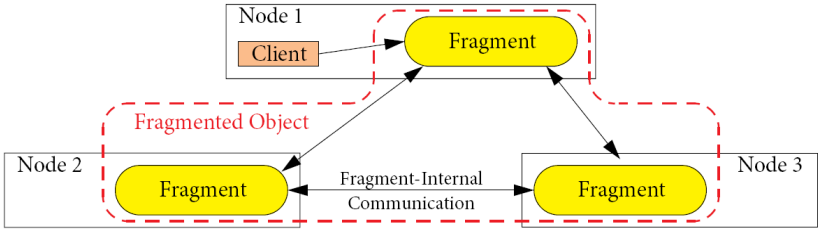
Literatur

Mesaac Makpangou, Yvon Gourhant, Jean-Pierre le Narzul, Marc Shapiro
Fragmented Objects for Distributed Abstractions

Fragmentiertes Objektmodell

Erweiterung des klassischen Stub-basierten verteilten Objekts

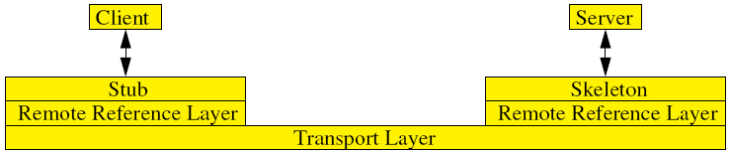
- *Echt* verteiltes Objekt mit eindeutiger ID (→ Fragmente)
- Verteilung von Zustand und Funktion mit offener interner Kommunikation
- Lokale Fragmente bieten die komplette Schnittstelle des fragmentierten Objekts an
- Bietet Unterstützung für dynamische Adaption (z.B. Zustandsmigration)
- Implizite Bindung (Weitergabe einer Referenz erzeugt objektspezifisches Fragment)



Übersicht

- Ziel von Java RMI

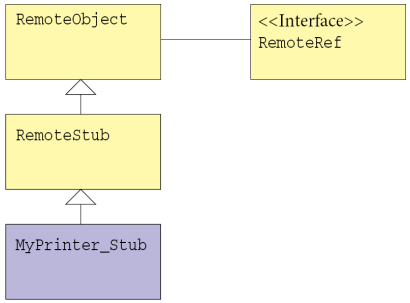
  Erhaltung der Semantik des Java-Objektmodells im Kontext von verteilten Systemen
- Parameterübergabe-Semantik
  - **Call-by-value**
    - Primitive Datentypen
    - Java-Objekte
    - Nicht exportierte RMI-Objekte
  - **Call-by-reference**: Exportierte RMI-Objekte
- Architektur



Stubs und Remote-Referenzen

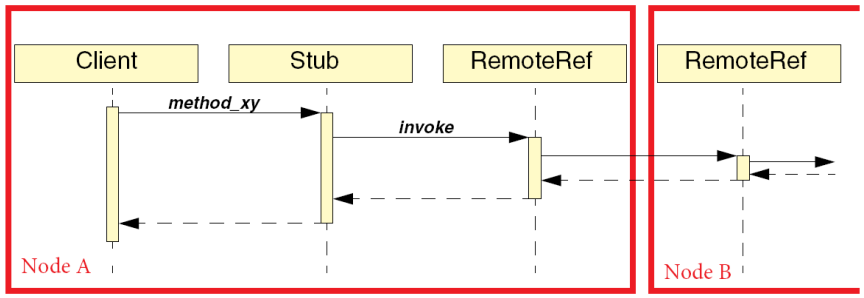
Aufbau von RMI-Stubs

- `RemoteObject`: Basisklasse für Stubs und RMI-Objekte
- `RemoteStub`: Basisklasse für alle Stubs
- `RemoteRef`
  - `invoke()`-Methode für Fernmethodenaufrufe am Remote-Objekt
  - Implementiert `Serializable` → Weitergabe zwischen Rechnern



Fernmethodenaufruf

Entfernter Methodenaufruf in Java RMI



Interne Abläufe

- Aufruf
  - Stub-Objekt wandelt Aufruf in generischen Aufruf um
  - Stub-Objekt ruft `invoke()`-Methode an der `RemoteRef` auf

```
Object invoke(Remote obj, Method method, Object[] params, long opnum)
```

    - `obj`: Objekt, das die Remote-Referenz enthält, z.B. der Stub selbst (`this`)
    - `method`: aufzurufende Methode
    - `params`: Methodenparameter
    - `opnum`: Hash-Wert-Repräsentation der aufzurufenden Methode
- Marshalling
  - Falls RMI-Objektreferenz vom Typ `RemoteStub`, dann serialisiere Referenz
  - Falls RMI-Objektreferenz exportiertes Objekt, erzeuge Stub-Objekt und verfahre wie oben
- Unmarshalling

  Deserialisiere Stub-Objekt

FORMI

- Fragmentierte Objekte
- RMI revisited
- Fragmented Object RMI

FORMI — Übersicht

- Fragmentiertes Objektmodell mit Java RMI

  Design und Implementierung entstanden im Rahmen einer Master-Arbeit
- Anforderungen
  - „Friedliche“ Koexistenz von RMI- und fragmentierten FORMI-Objekten
  - Zugriffstransparenz für Aufrufer

    → kein Unterschied zwischen RMI- und FORMI-Objekt
  - Modelltransparenz bei Parameterübergabe

    → kein Unterschied bei Parameterübergabe
    - Übergabe von FORMI-Objekten an RMI-Objekte
    - Übergabe von RMI-Objekten an FORMI-Objekte
    - Übergabe von FORMI-Objekten an FORMI-Objekte
- Transparente Parameterübergabe

  FORMI-Fragmente müssen sich wie RMI-Stubs verhalten

  → müssen mit RMI-Mechanismen serialisierbar sein
- Literatur

  Rüdiger Kapitza, Michael Kirstein, Holger Schmidt, Franz J. Hauck

  FORMI: An RMI Extension for Adaptive Applications

  `http://middleware05.objectweb.org/WSProceedings/ARM05/a2-kapitza.pdf`

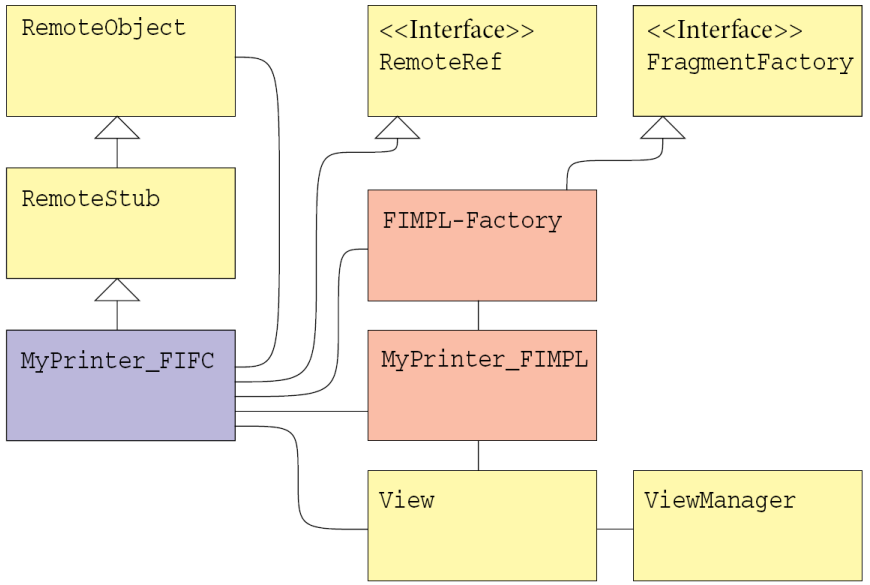
Aufbau eines Fragments

Anforderung an den Fragmentaufbau

- Dynamischer Austausch der Implementierung
  - Indirektionsstufe: Trennung von
    - Fragment-Implementierung (→ `*_FIMPL`)
    - Fragment-Interface (→ `*_FIFC`)
  - Achtung: Als Fragment-Interface wird eine eigene Java-Klasse verwendet (kein `Interface`)
- Dynamische Entscheidung über Fragment-Implementierung bei Parameterübergabe

→ Einsatz einer Fragment-Implementierungs-Factory (→ `FIMPL-Factory`)

Aufbau eines Fragments



Aufbau eines Fragments

Lösung in FORMI

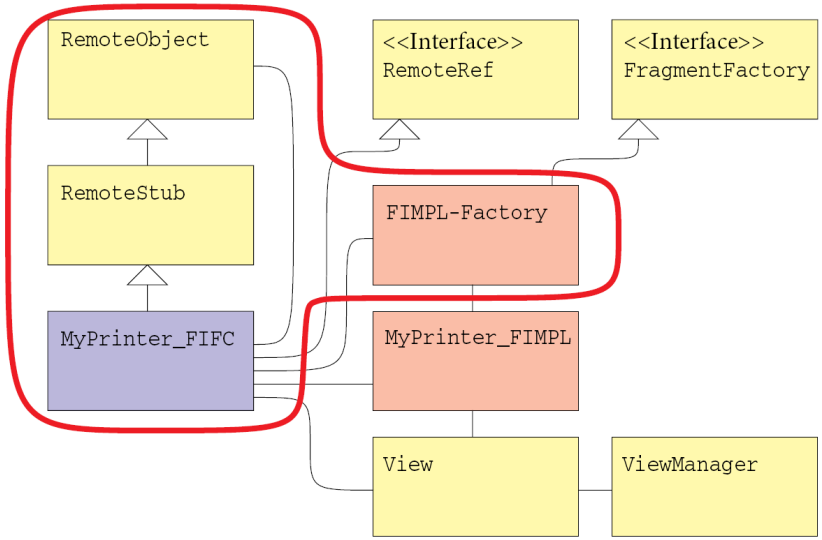
- Stub ⇔ Fragment-Interface
- Fragment-Interface implementiert `RemoteRef` und referenziert sich selbst
  - Vermeidet Marshalling-Probleme
  - `invoke()`-Aufrufe werden in lokale Aufrufe umgewandelt
- Fragment-Interface leitet Aufrufe an Fragment-Implementierung weiter
- Fragment-Interface referenziert eine Factory für Fragment-Implementierungen
  - Factory fällt Entscheidung welche Implementierung verwendet wird
  - Dynamische Entscheidungen möglich
- View-Objekt sorgt für Sharing der Fragment-Implementierung
  - View-Objekt wird in einem View-Manager verwaltet
  - View-Manager enthält Tabelle aller lokalen Fragmente

Marshalling und Unmarshalling

- Marshalling
  - Serialisierung des Fragment-Interface
  - View und Fragment-Implementierung sind als `transient` gekennzeichnet und werden nicht serialisiert
- Unmarshalling
  - Deserialisierung des Fragment-Interface
  - Suche nach vorhandener View
    - Verlinken der View und entsprechender Fragment-Implementierung oder
    - Erzeugung einer neuen Fragment-Implementierung über Factory und Erzeugung eines neuen View-Objekts über den View-Manager

Marshalling und Unmarshalling

Serialisierungsbereich



Erzeugen von FORMI-Objekten

Erzeugen eines ersten Fragments

- Aufruf einer speziellen Funktion mit Übergabe von
  - Fragment-Implementierungs-Factory
  - FORMI-Objekttyp
  - Erste Fragment-Implementierung
- Erstes Fragment für Client als RMI-Stub sichtbar
  - Bei Parameterübergabe: Marshalling
  - Kein Exportieren nötig

Kommunikation zwischen Fragmenten

- Beliebige Kommunikation
  - Java-Sockets stehen zur Verfügung
  - Einsatz von Standard-RMI für aufrufbasierte Kommunikation zwischen Fragmenten
- Finden der Fragmente
  - Zur Zeit kein Ortsdienst verfügbar
    - Kann selbst implementiert werden
    - Verankerung in der Fragment-Implementierungs-Factory
  - Statische Kommunikationsadressen

    Implantieren der Adressen in die Fragment-Implementierungs-Factory
  - Verwendung der Registry als Ersatz für Ortsdienst

    Beispiel: zum Finden von Standard-RMI-Objekten, die anderes Fragment repräsentieren

Unterstützung bei der Objektentwicklung

- Eigener RMI-Compiler: `formic`
  - Aus `rmic`-Implementierung von Sun abgeleitet
  - Erzeugt Fragment-Interfaces (spezielle Remote-Stubs für FORMI)
- Fragment-Implementierung
  - Implementierung der `Remote`-Schnittstelle
  - Unterklasse von `FormiFragment`

  → Keine Code-Erzeugung notwendig
- FORMI-Code
  - Sourcen: `/local/formi/src/`
  - Bibliothek: `formi.jar` in `/local/formi/lib/`