

Kontomanagementsystem CORBA

Roman Brunnemann, Stephan Müller, Sven Widmer, Xiao Tai Yu

Universität Potsdam
Hasso Plattner Institut

17. Juni 2004



"Ma, why ist this and why ist that? "
"Would you please stop asking those why-questions! "
"Why? "

Unser Vortrag ist für alle "Warum"-Fragen zum Projekt da!

*"Warum haben wir ein Kontomanagementsystem
entwickelt? "*



Motivation

Anforderungsspezifikation

Use Cases

Architektur

Anforderungen

Livepräsentation

Server

Serverseitige Implementation

Lifecycle Service

Möglicher Lifecycle in .NET

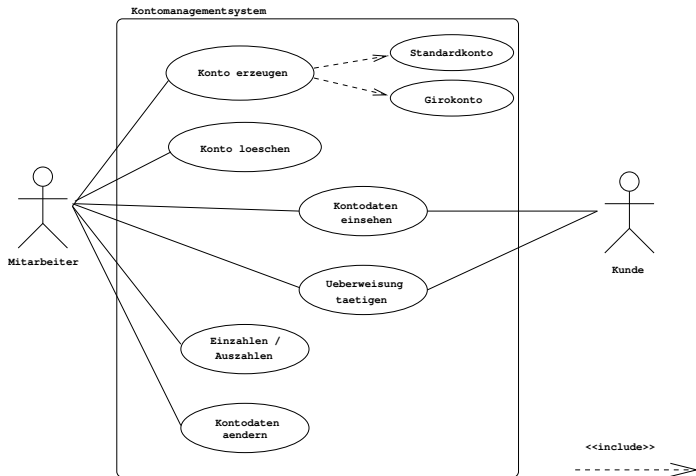
Client

Komponenten

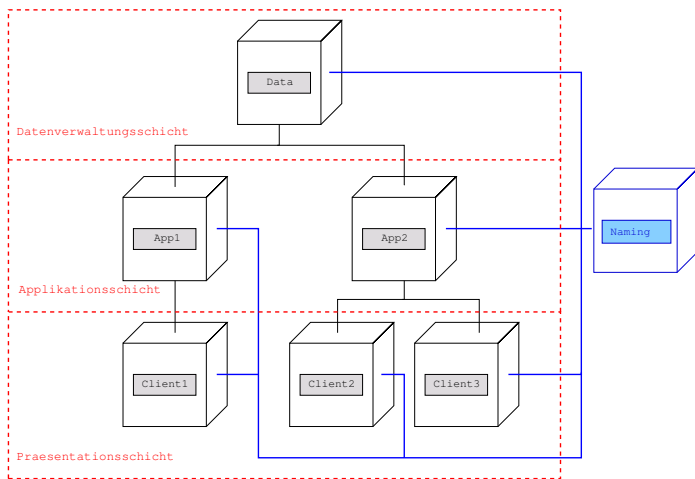
Tests



Use Cases



Verteilung auf Rechnerknoten



Technische Anforderungen

- ▶ Verteilbar / 3 Schichtenmodell
- ▶ Skalierbar
- ▶ Dynamisch erweiterbar
- ▶ Interoperabel
- ▶ Datenkonsistenz (Write Through)
- ▶ Logfile
- ▶ Datenintegrität (Mutual Exclusion)



- ▶ Middleware CORBA - Implementation ORBacus
- ▶ Serverimplementation C++
- ▶ Clientimplementation:
 - ▶ C++ mit QT
 - ▶ Java mit Swing

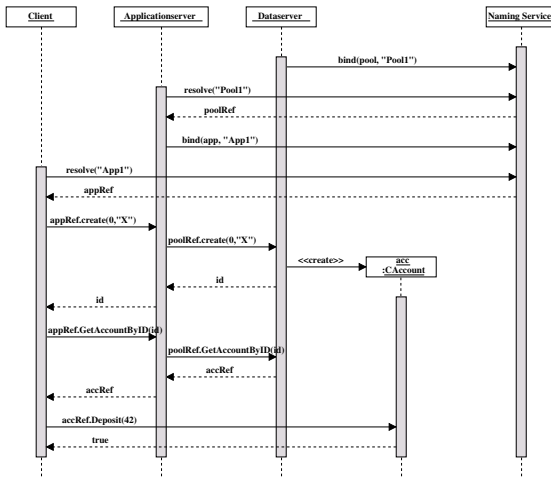


Verwendete Patterns

- ▶ Object Factory
- ▶ Namensservice
- ▶ MVC
- ▶ Lifecycle (Evictor Pattern)



Verarbeitungsmodell - Anlegen eines Kontos



Funktionalitäten des Servers

- ▶ Trennung von Datenhaltung und Applikationsschicht
 - ▶ Applikationsserver übernehmen rechenintensive Aufgaben
- ▶ Multithreadfähige Implementation – Voraussetzung:
 - ▶ Referenzzählung
 - ▶ Gleichzeitiger Zugriff auf ein und das selbe Konto wird verhindert
- ▶ Loggen aller durchgeführten Operationen
- ▶ Eigene Implementation des Lifecycle
 - ▶ Eine vordefinierte Anzahl von Objekten wird im Speicher gehalten



Lifecycle Service - Verwendung des Evictor Patterns

- ▶ Definiert eine Liste aktiver Objekte, die beim Erreichen einer Obergrenze verdrängt werden (hier: LRU).
- ▶ POA unterscheidet nur zwischen Halten bzw. Vernichten von Servants nach dem Zugriff
- ▶ Lösung: Implementation eines Servantmanagers
 - ▶ Vor dem Zugriff auf ein Servant:
 - ▶ Prüfen ob Servant bereits im Speicher, wenn nein, Aktivieren von der Festplatte und ggf. Verdrängen des letzten Elementes der Liste
 - ▶ Nach dem Zugriff:
 - ▶ Ggf. schreiben von Veränderungen auf den Sekundärspeicher (Write-Through Ansatz)



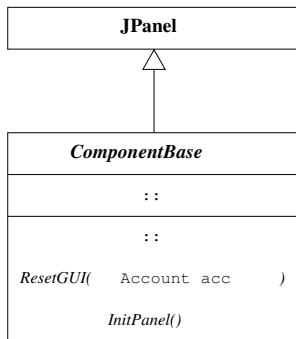
Möglicher Lifecycle Service – .NET Remoting

- ▶ Server Activated Object (SAO), Singlecall
 - ▶ Statusloser Objektzugriff, Lebensdauer auf Methodenaufruf beschränkt
- ▶ Client Activated Object (CAO)
 - ▶ Statusbehaftet, jeder Client erhält eigene Instanz auf dem Server, vordefinierte Lebensdauer
- ▶ Vereinigung der Vorteile beider Aktivierungsarten
 - ▶ Erzeugen der Instanzen durch eine verteilte Object Factory
- ▶ Lifecycle Kontrolle
 - ▶ Lease Time
 - ▶ Client- und serverseitiges Sponsoring



Architektur

► Abstrakte Basisklasse



Dynamisches Nachladen

- ▶ Klassenhierarchie der Kontotypen, dem Client ist nur Basiskontotyp bekannt
- ▶ Kontotyp enthält eigene RTTI
 - ▶ Name von Kontotyp wird ausgelesen und entsprechende Komponente z.B. von einem Webserver nachgeladen

Nachladen von Klassen in Java

```
URLClassLoader ucl = new URLClassLoader(URL);  
component = (ComponentBase)Class.forName( className,  
1, ucl ).newInstance();
```



Testverfahren

- ▶ Testen von Dataserver und Applicationserver mittels eigener Testprogramme
- ▶ Abarbeitung von Testfällen
- ▶ Grenzfälle, Bottom-Up, White-Box
- ▶ Multithread Test auf SMP mit konkurrierenden Zugriffen von verschiedenen Rechnern
- ▶ Clienttest: manuell bzw. mit JUnit



Quellen

- ▶ Henning, Michi and Vinoski, Steve – Advanced CORBA Programming with C++, Addison-Wesley, 1999
- ▶ Stroustrup, Bjarne – Die C++ Programmiersprache, Addison-Wesley, 1999
- ▶ Qt Reference Documentation
- ▶ Dalheimer, Matthias Kalle – Programming with Qt, O'Reilly 1999
- ▶ Rammer, Ingo – Advanced .NET Remoting

