Komponentenbasierter Taschenrechner mit CORBA

Silke Kugelstadt
Torsten Steinert

Inhalt

- Motivation
- Demonstration des Taschenrechners
- Grobarchitektur
- Implementierung des Clients
- Implementierung der Komponenten
- Entwurfsmuster
- Erweiterungsmöglichkeiten
- Vergleich mit anderen Komponentenframeworks
- Live Implementierung einer neuen Komponente

Demonstration

Taschenrechner

Motivation

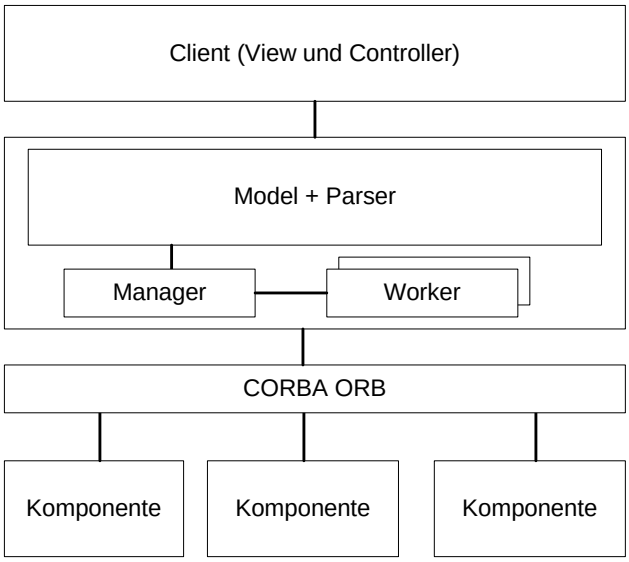
- Grundfunktionen
- Hinzufügen / Entfernen von Komponenten

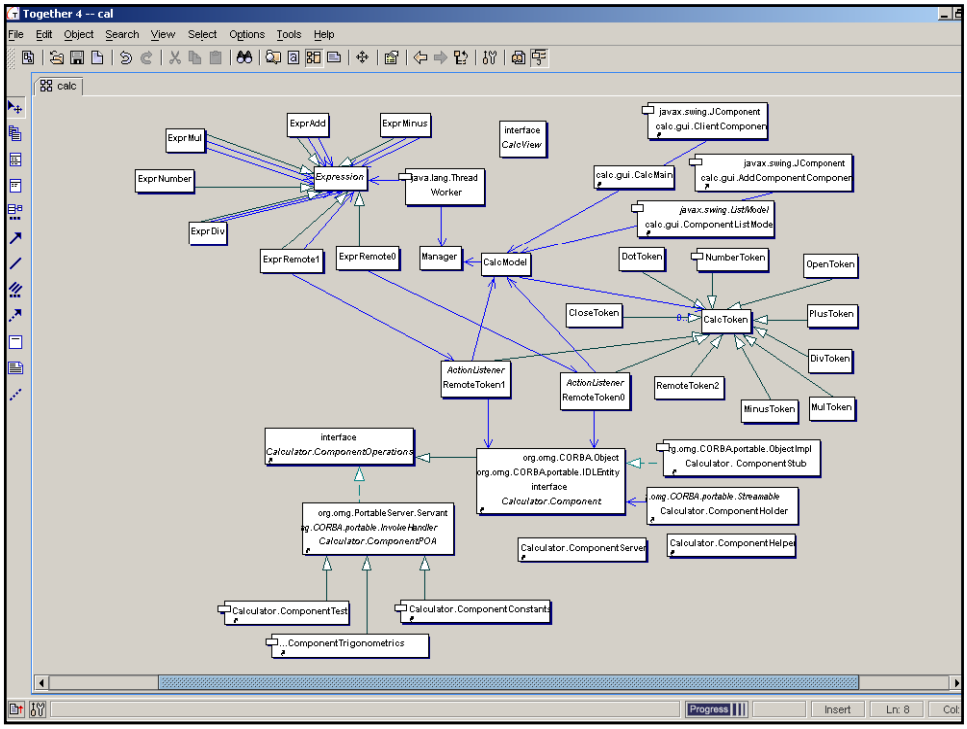
- Mehrfachfaden (Multi-Threading)
- Verteilungstransparenz
- Einfache Bedienbarkeit
- Einfache Erweiterbarkeit

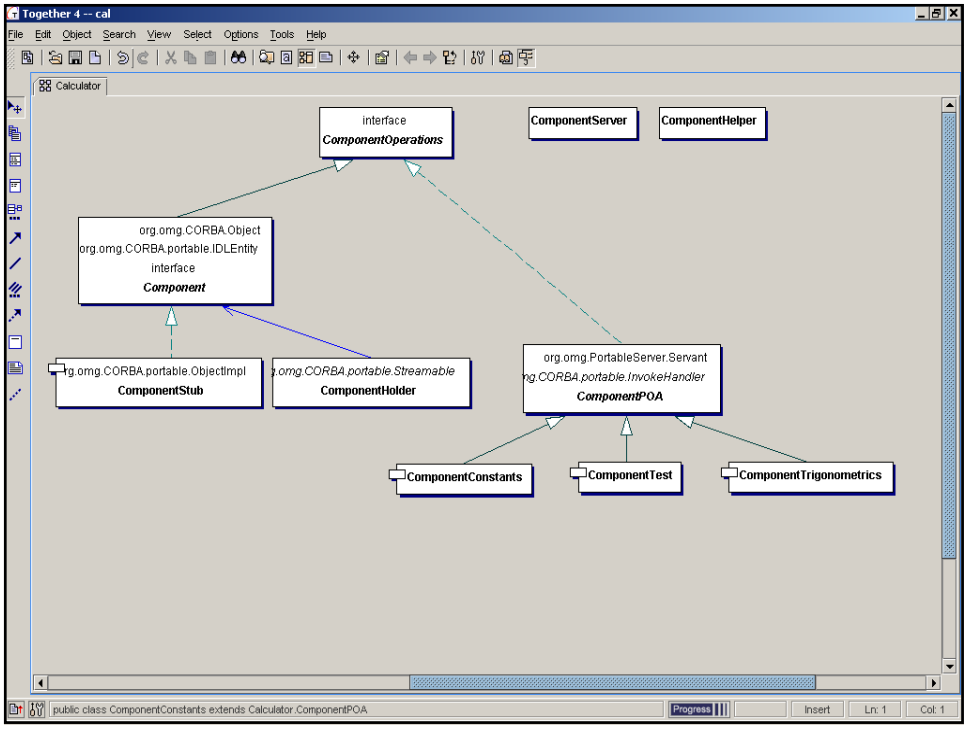
Anforderungsspezifikation

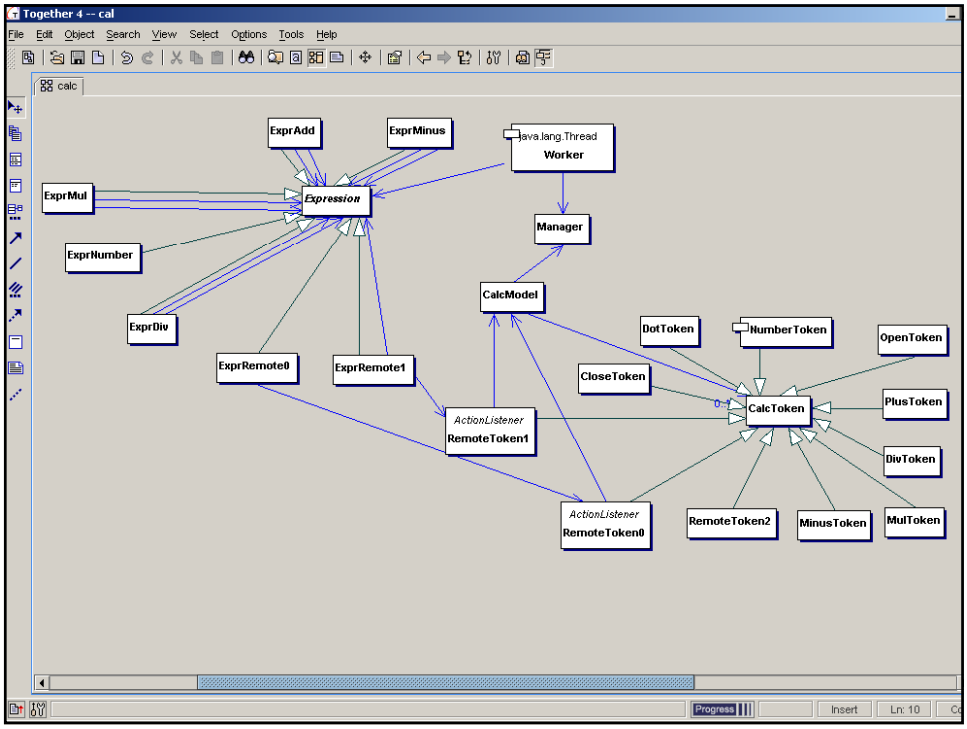
- Taschenrechner mit Grundfunktionen
- Hinzufügen von Funktionseinheiten zur Laufzeit
- Client / Server Architektur
  - Taschenrechner als Client
  - Jede Komponente als Server

Architektur




Together 4 -- cal
File Edit Object Search View Select Options Tools Help
calc
ExprAdd
ExprMinus
ExprMul
ExprNumber
Expression
java.lang.Thread
Worker
ExprDiv
ExprRemote1
ExprRemote0
Manager
interface
CalcView
javax.swing.JComponent
calc.gui.ClientComponent
calc.gui.CalcMain
javax.swing.JComponent
calc.gui.AddComponentComponent
javax.swing.ListModel
calc.gui.ComponentListModel
CalcModel
DotToken
NumberToken
OpenToken
CloseToken
CalcToken
PlusToken
DivToken
ActionListener
RemoteToken1
ActionListener
RemoteToken0
RemoteToken2
MinusToken
MulToken
interface
Calculator.ComponentOperations
org.omg.CORBA.Object
org.omg.CORBA.portable.IDLEntity
interface
Calculator.Component
org.omg.CORBA.portable.ObjectImpl
Calculator.ComponentStub
org.omg.CORBA.portable.Streamable
Calculator.ComponentHolder
org.omg.PortableServer.Servant
org.omg.CORBA.portable.InvokeHandler
Calculator.ComponentPOA
Calculator.ComponentServer
Calculator.ComponentHelper
Calculator.ComponentTest
Calculator.ComponentConstants
...ComponentTrigonometrics
Progress
Insert
Ln: 8
Col:


Together 4 -- cal
File Edit Object Search View Select Options Tools Help
Calculator
interface
ComponentOperations
ComponentServer
ComponentHelper
org.omg.CORBA.Object
org.omg.CORBA.portable.IDLEntity
interface
Component
org.omg.CORBA.portable.ObjectImpl
ComponentStub
org.omg.CORBA.portable.Streamable
ComponentHolder
org.omg.PortableServer.Servant
org.omg.CORBA.portable.InvokeHandler
ComponentPOA
ComponentConstants
ComponentTest
ComponentTrigonometrics
public class ComponentConstants extends Calculator.ComponentPOA
Progress
Insert
Ln: 1
Col: 1



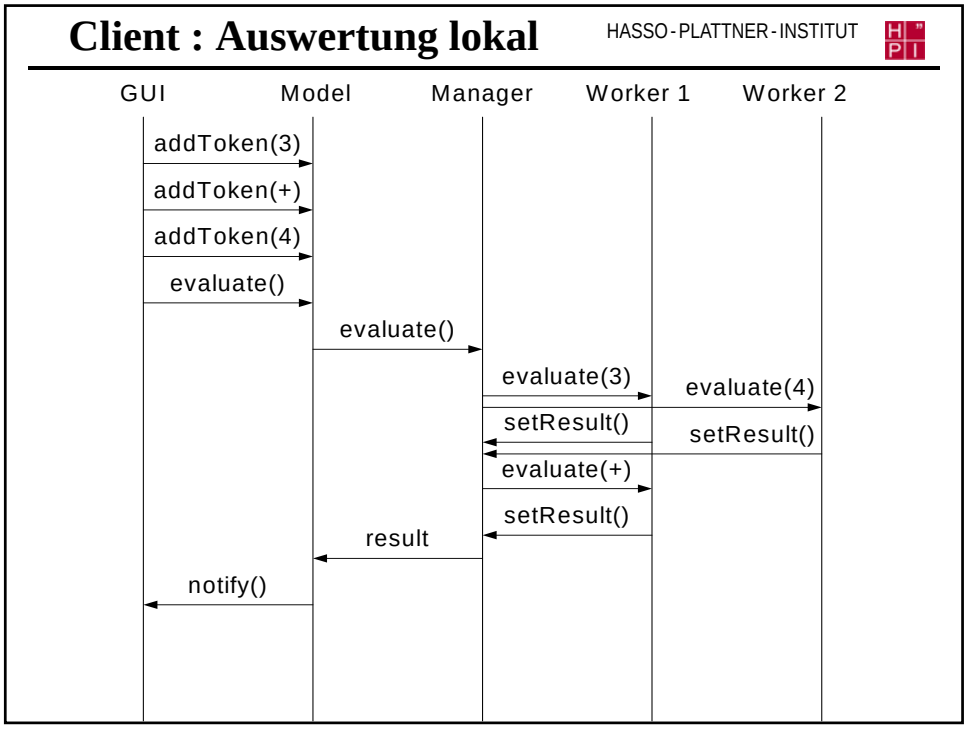
Client

HASSO-PLATTNER-INSTITUT

- Arithmetische Ausrücke
- Eingabe einer Sequenz von Token
- Umwandlung in einen Ableitungsbaum
- Auswerten des Ableitungsbaumes

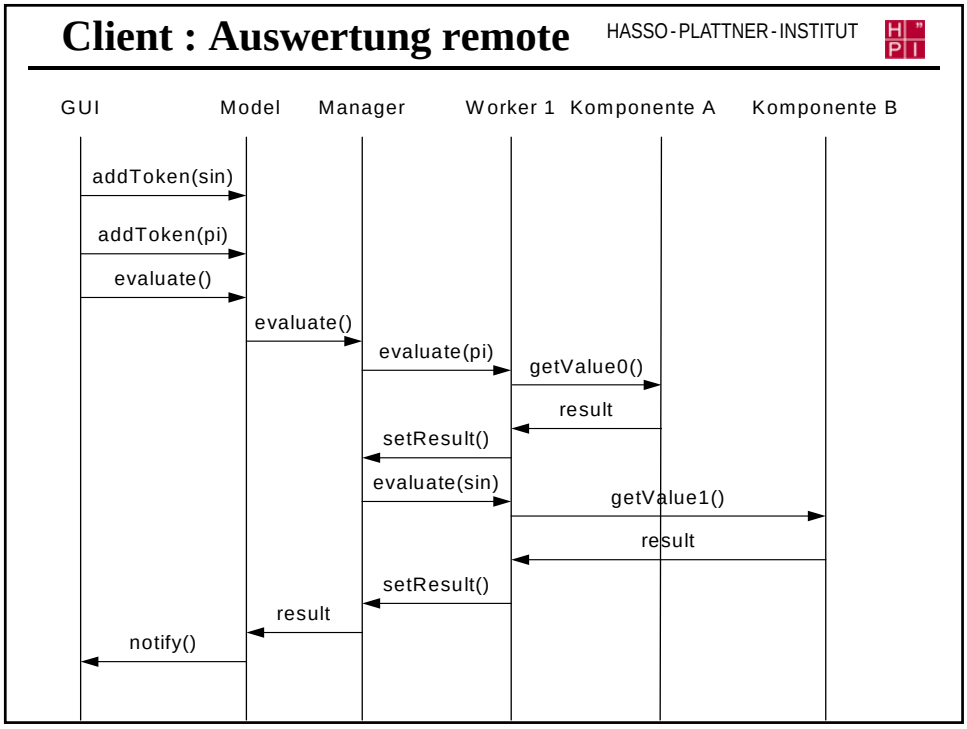
Client : Auswertung lokal

HASSO-PLATTNER-INSTITUT



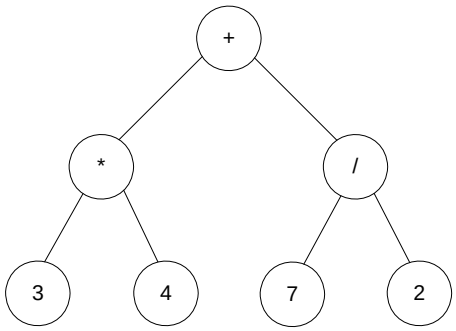
Client : Auswertung remote

HASSO-PLATTNER-INSTITUT



Ableitungsbaum

Beispiel : ((3 * 4) + (7 / 2))



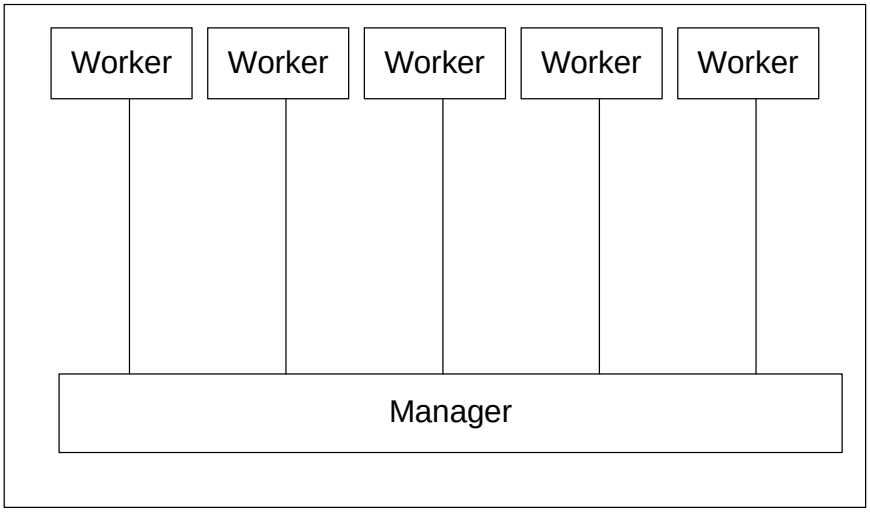
Client : Parallelität

Entwurfsmuster : Master – Worker

- 10 Worker Threads
  - Unabhängige Auswertung einer Expression
- 1 Manager Thread
  - Verteilt die auszuwertenden Expression auf die Worker Threads

Client : Parallelität 2

Master – Worker



Monitor Konzept

- Manager und Worker sind als Monitor implementiert
  - Gegenseitig ausschließender Zugriff (mutual exclusion)
  - Kapselt Status
  - Zugriff nur über veröffentlichte Methoden

Komponenten

- Komponenten stellen neue Funktionen zur Verfügung.
  - 0 – Stellige (Konstanten)
  - 1 – Stellige (z.B. sin)
  - 2 – Stellige (z.B. x ^ y)

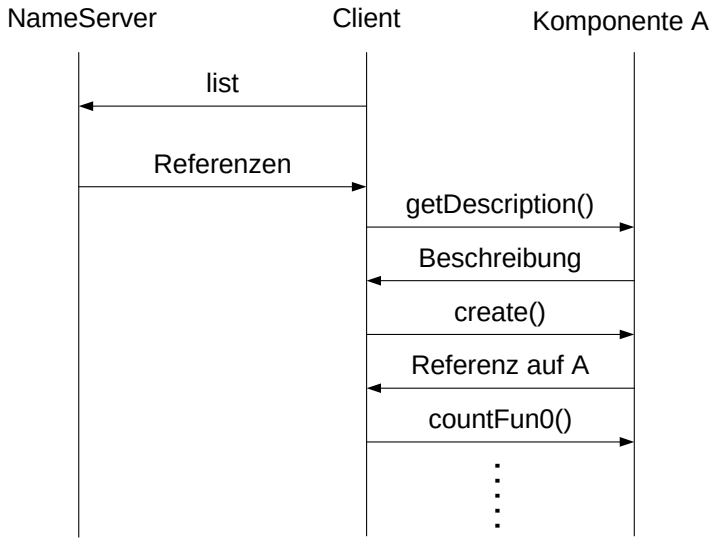
Komponenten : IDL


```
module Calculator {
 interface Component {
  readonly attribute string description;

  readonly attribute unsigned long countFun0;
  readonly attribute unsigned long countFun1;
  readonly attribute unsigned long countFun2;

  string getNameFun0(in unsigned long index);
  string getNameFun1(in unsigned long index);
  string getNameFun2(in unsigned long index);

  double getValue0(in unsigned long index);
  double getValue1(in unsigned long index, in double p1);
  double getValue2(in unsigned long index, in double p1, in double
   p2);

  Component create();
 };
};
```

Einschränkungen

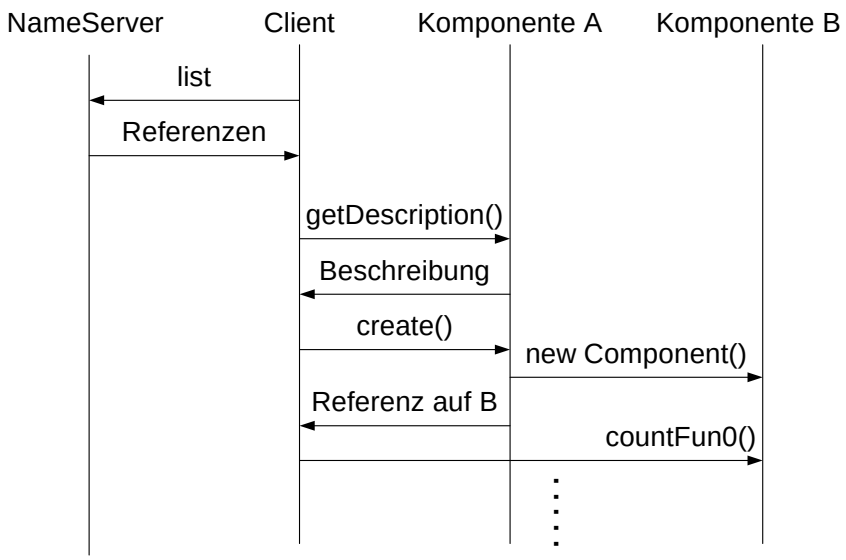
- Funktionen mit mehr als 2 Parameter
- Einstellungen für Komponenten (z.B. rad -> deg bei Trigonometrischen Funktionen)
- Erweiterungen außerhalb des Funktionsparadigmas

Entwurfsmuster : Singleton



Nicht -Singleton

HASSO-PLATTNER-INSTITUT



Verwendete Entwurfsmuster

HASSO-PLATTNER-INSTITUT

- Beobachtermuster
- Model – View – Controller
- Singleton
- Factory
- Proxy
- *Monitor*
- *Manager - Worker*

Tests

- Modultests
  - einzelne Module mit JUnit testen
- Integrationstests
  - gesamte Anwendung mit einem GUI-Tester testen, z.B.
    - Abbot
    - Pounder
    - Jemmy Module

Modultests mit JUnit

- Für jede zu testende Java Klasse eine Testklasse schreiben
- In jeder Testklasse mehrere Testfälle implementieren
- Tests regelmäßig automatisch ablaufen lassen

Integrationstests

- Wenn einzelne Module funktionieren, dann die gesamte Anwendung testen
- Dazu wird ein GUI Testtool benötigt
- Aus Zeitgründen noch nicht implementiert

Frameworkvergleich

- Von CORBA verwenden wir lediglich den transparenten, entfernten Objektaufruf und den Nameserver.
- DCOM, .NET etc bieten diese Funktionalität auch an

$\Rightarrow$ Eine Entwicklung mit diesen anderen Frameworks ist also auch denkbar.

Erweiterungen

- Parser Einschränkungen : Parameter ≤ 2
- Flyweight Muster bei großen Formeln
- Verwendung von POAs um u.a. folgende Dinge zu erreichen :
  - Persistenz
  - Automatisches starten von Komponenten

Literatur

- Gerald Brose et al : „Java Programming with CORBA“
- Gamma et al „Entwurfsmuster“
- Jeff Magee, Jeff Kramer : „Concurrency“
- Schmidt et al : „Pattern – Oriented Software Architecture“
- www.jacorb.org
- www.junit.org

Demonstration

Programmierung einer Komponente

Grammatik

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow i$