
Komponentenbasierter Taschenrechner mit CORBA

Silke Kugelstadt

Torsten Steinert

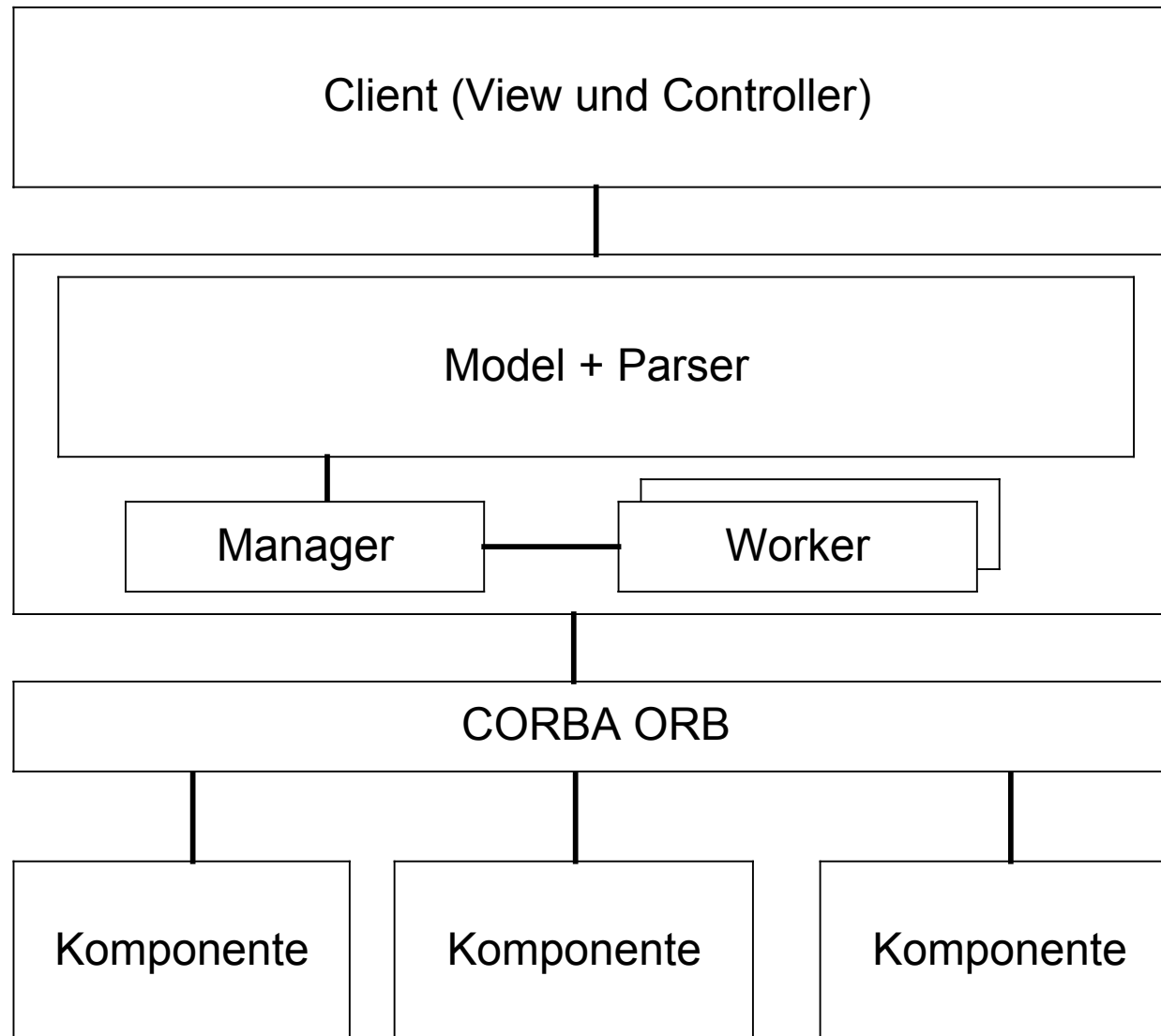
-
- Motivation
 - Demonstration des Taschenrechners
 - Grobarchitektur
 - Implementierung des Clients
 - Implementierung der Komponenten
 - Entwurfsmuster
 - Erweiterungsmöglichkeiten
 - Vergleich mit anderen Komponentenframeworks
 - Live Implementierung einer neuen Komponente

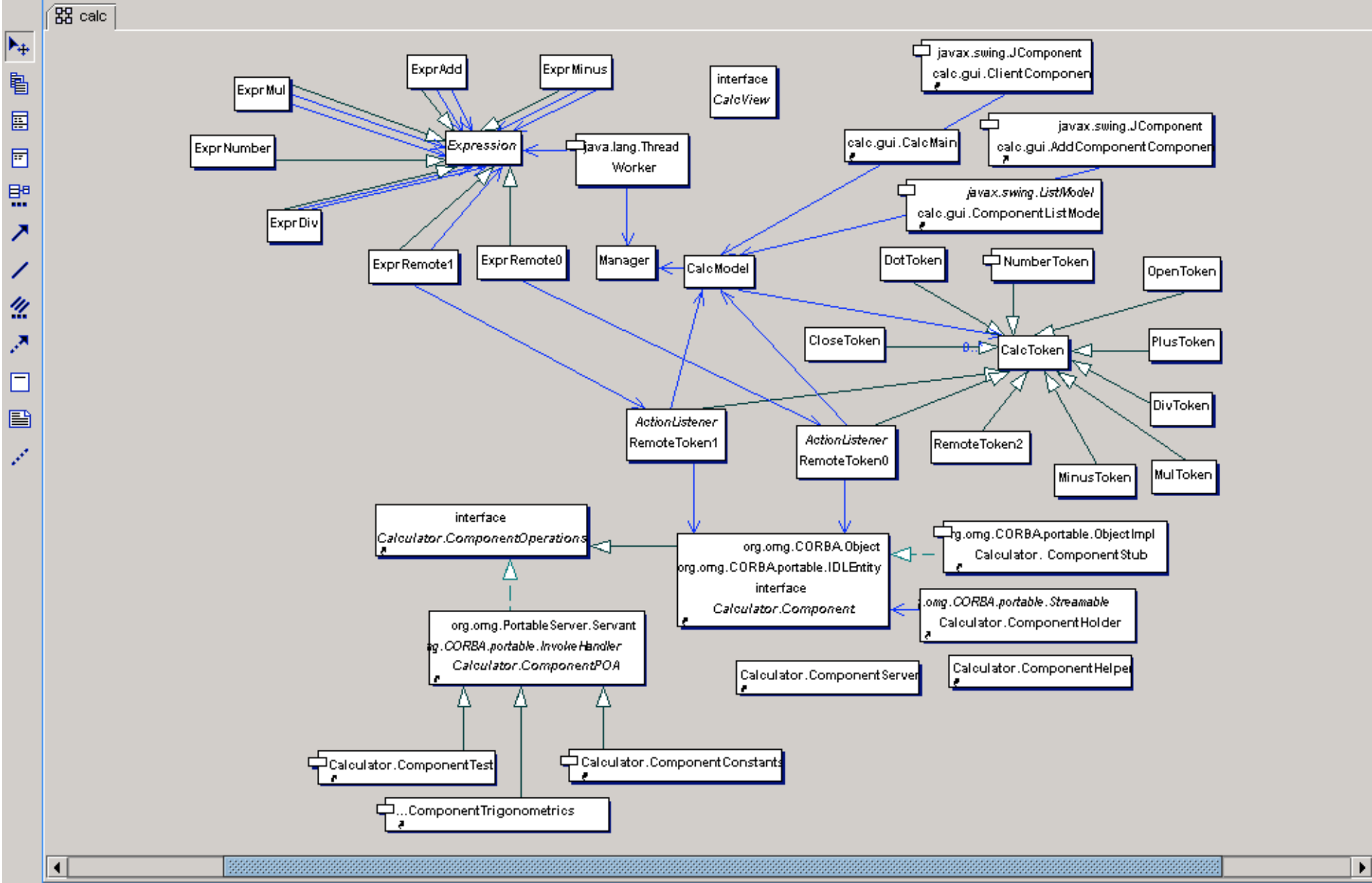
Demonstration

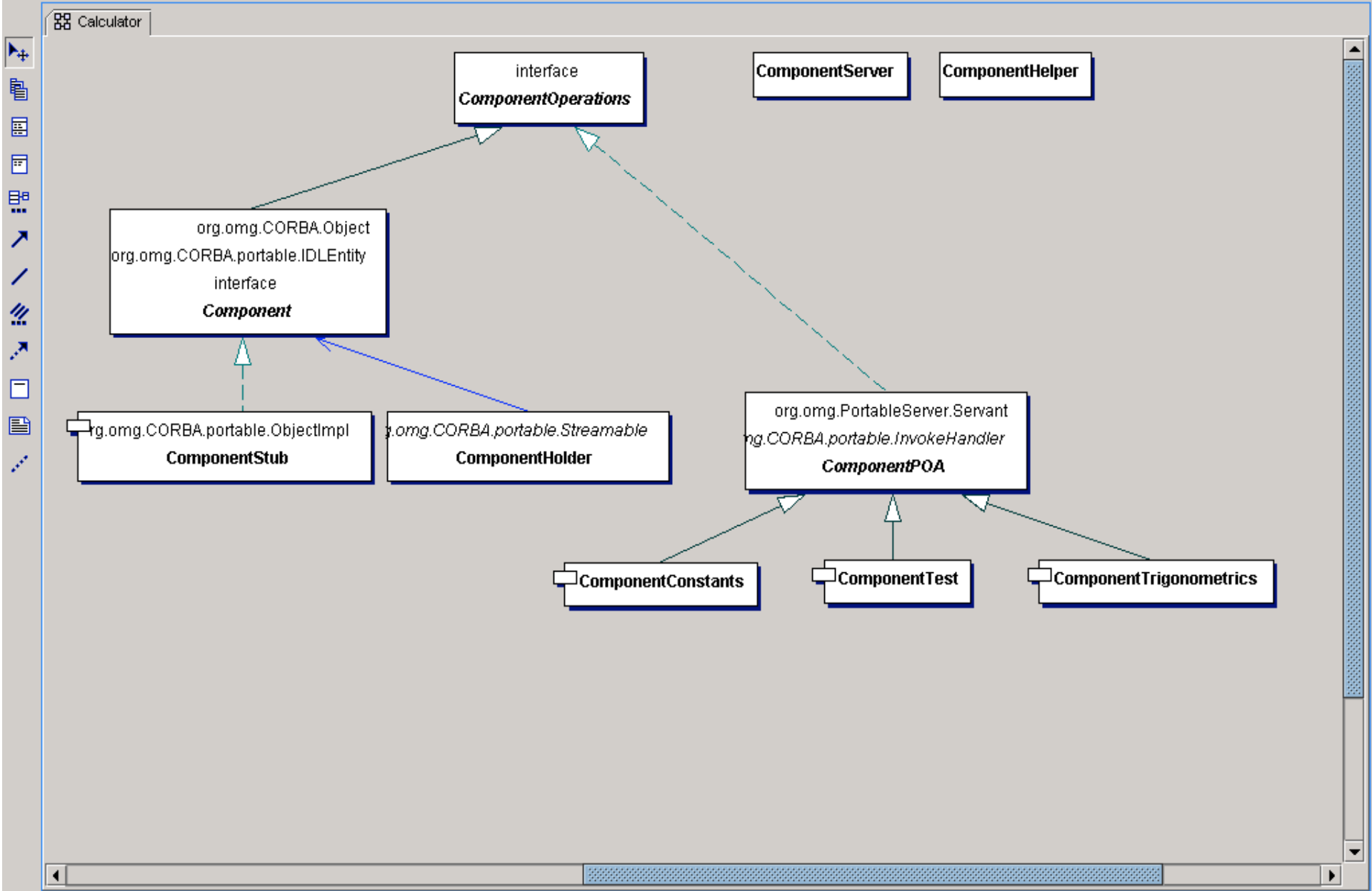
Taschenrechner

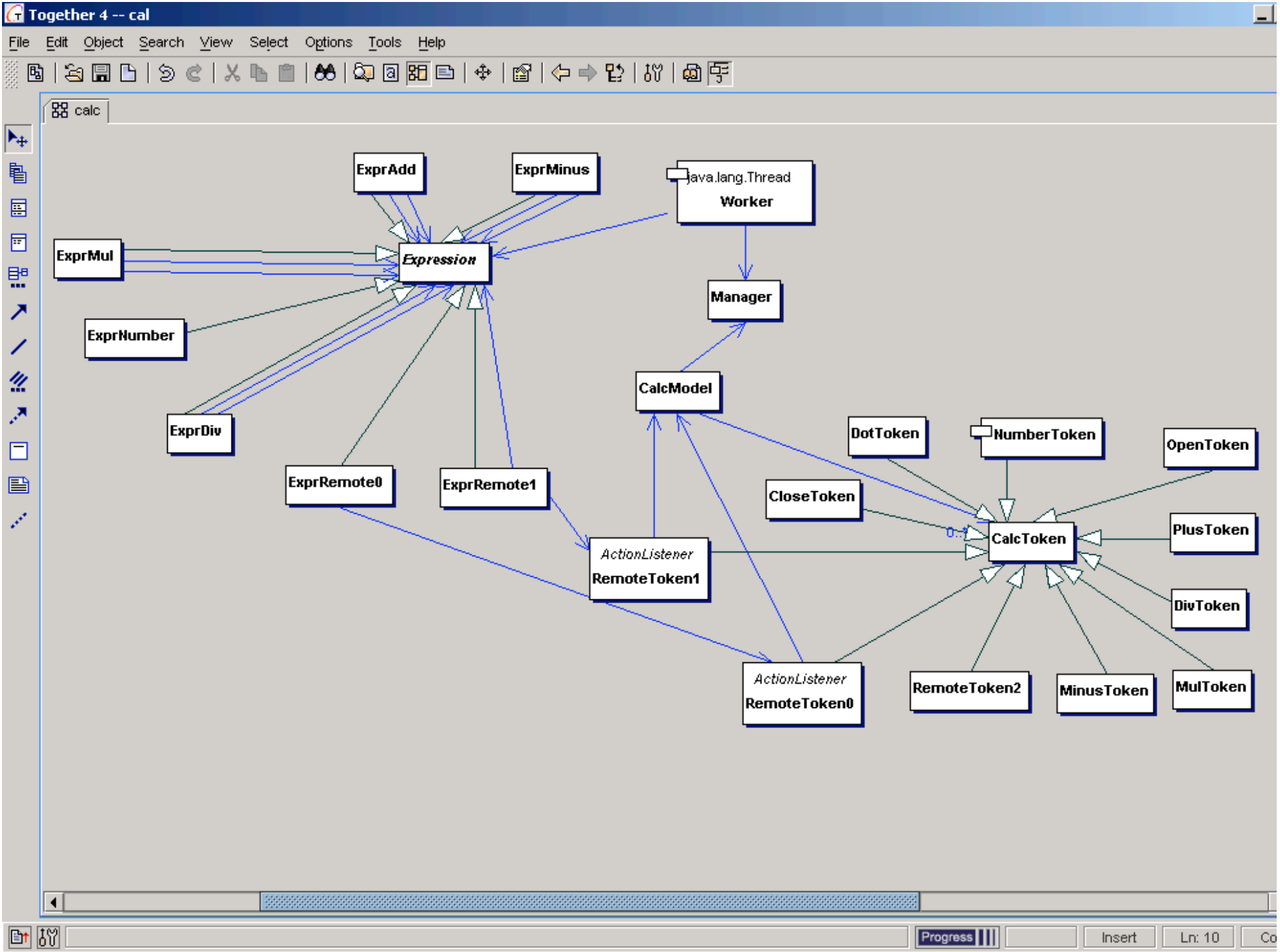
-
- Grundfunktionen
 - Hinzufügen / Entfernen von Komponenten
 - Mehrfachfaden (Multi-Threading)
 - Verteilungstransparenz
 - Einfache Bedienbarkeit
 - Einfache Erweiterbarkeit

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



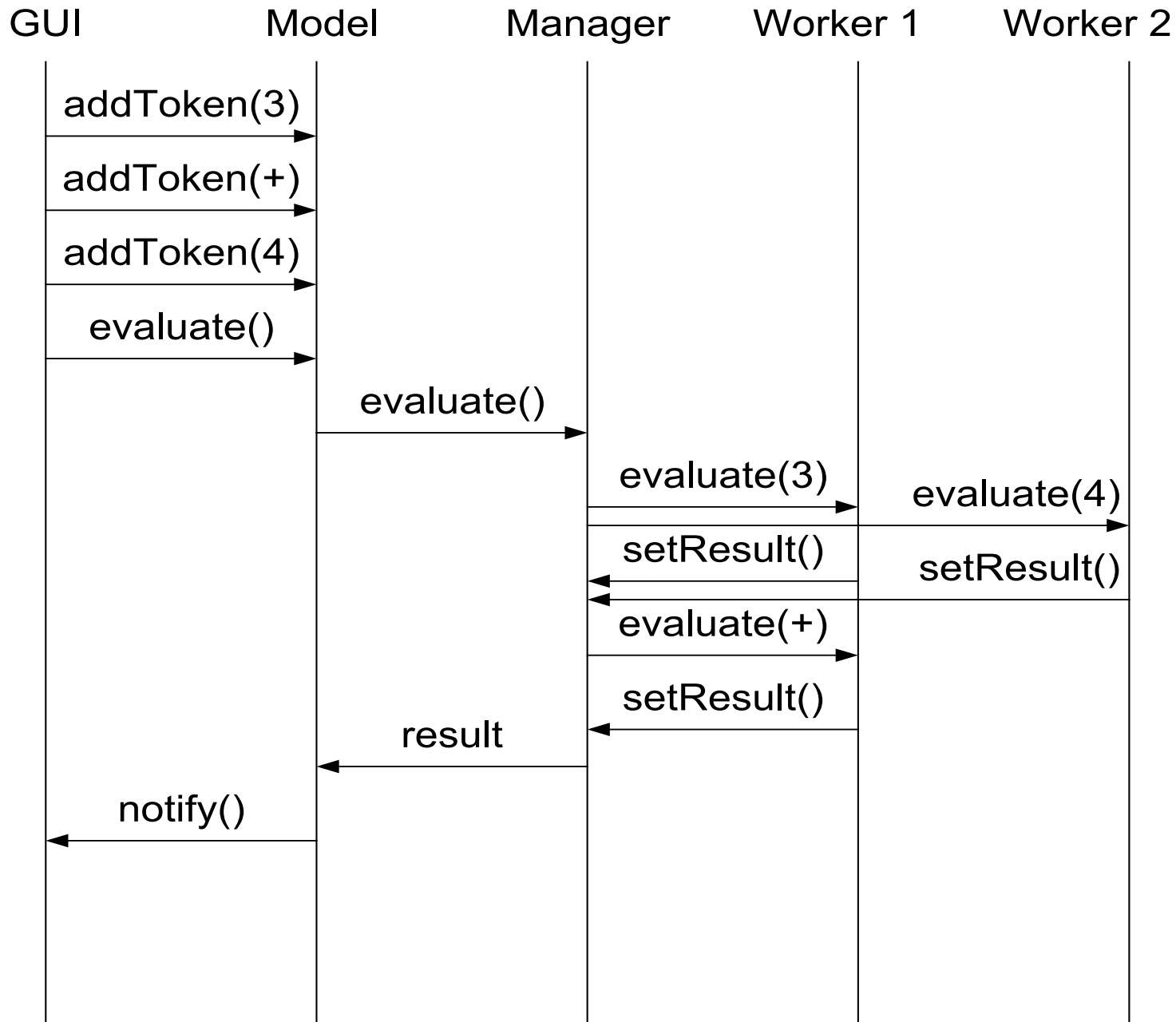




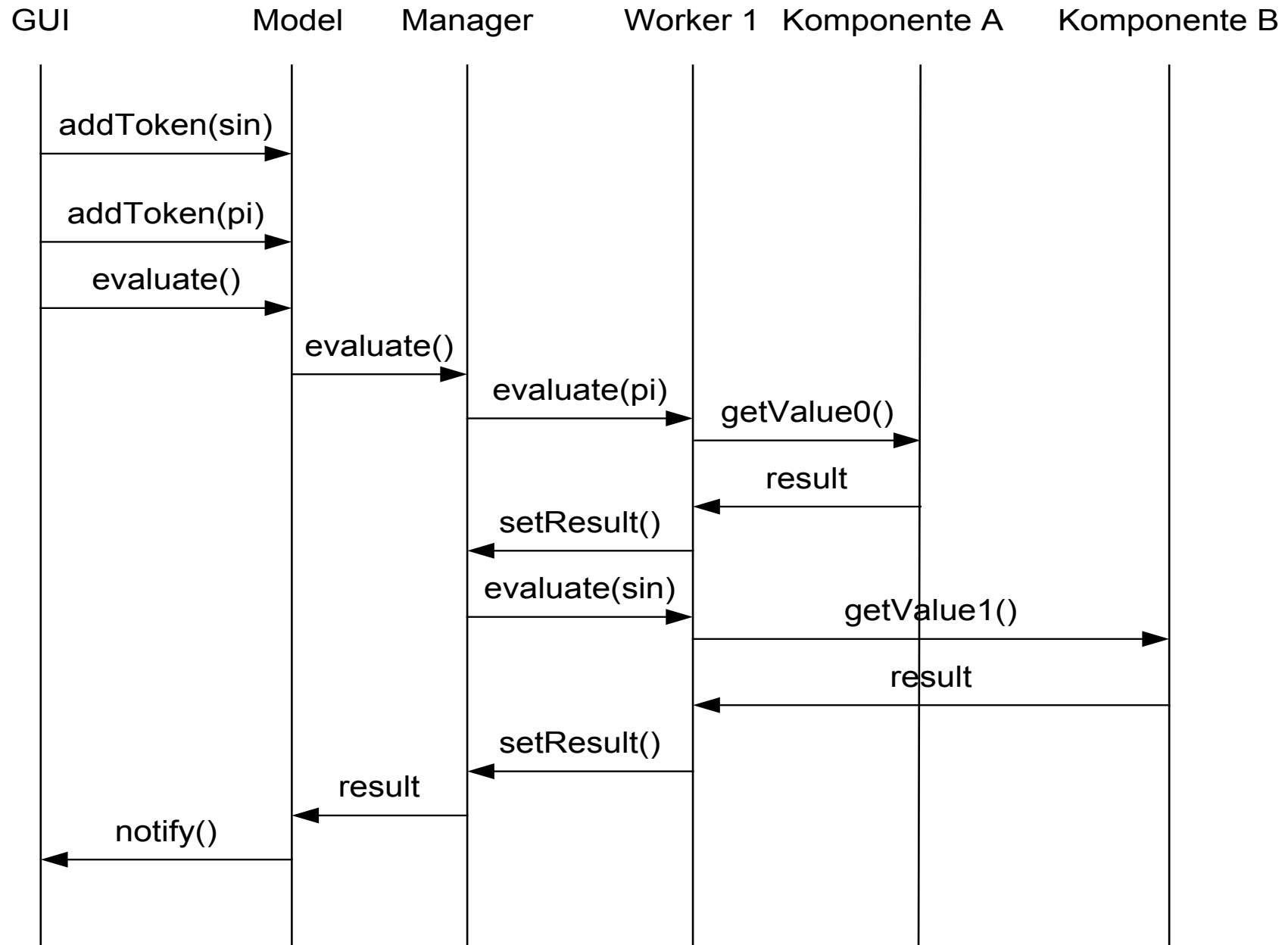


-
- Arithmetische Ausdrücke
 - Eingabe einer Sequenz von Token
 - Umwandlung in einen Ableitungsbaum
 - Auswerten des Ableitungsbaumes

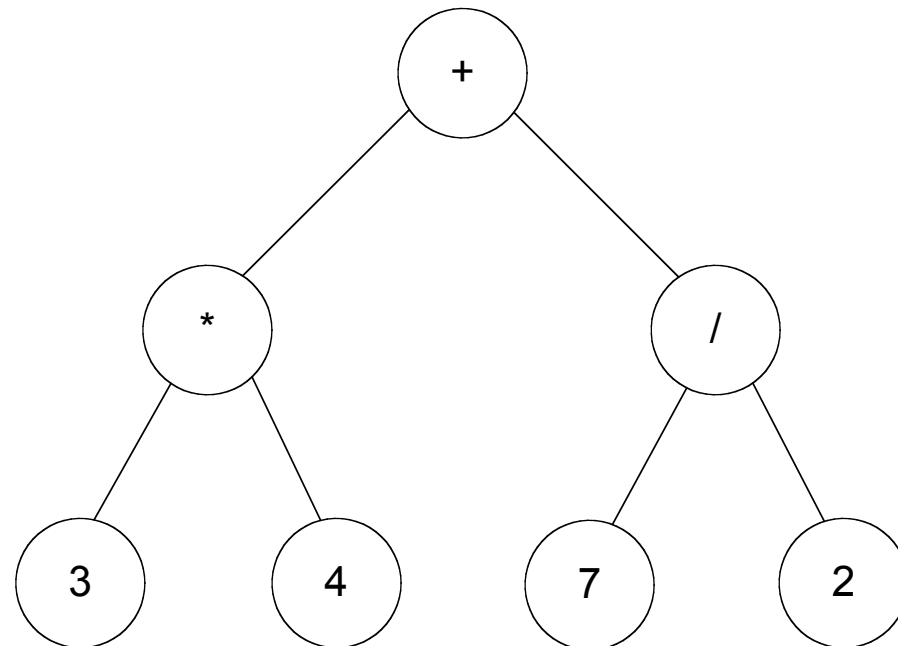
Client : Auswertung lokal



Client : Auswertung remote



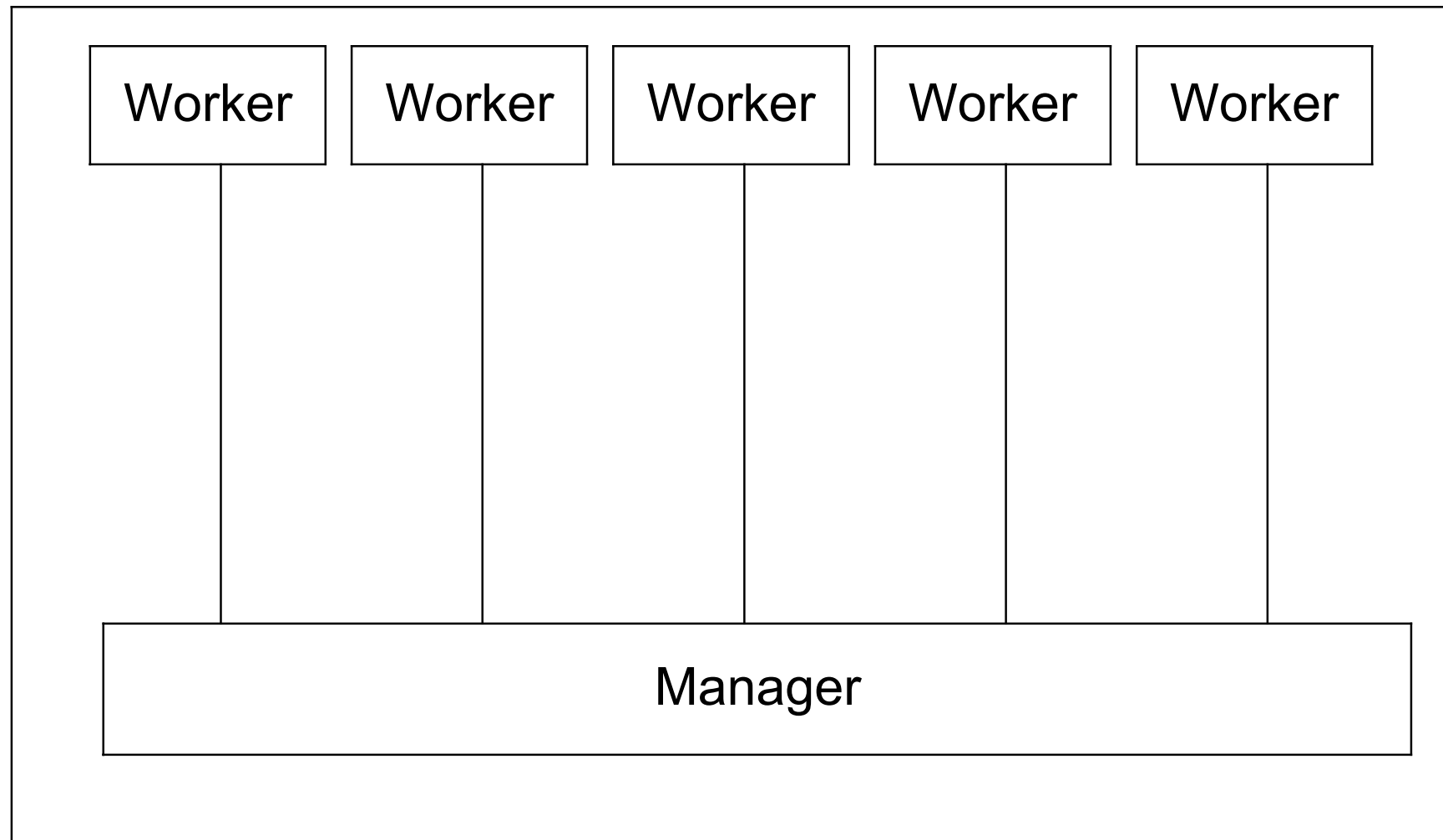
Beispiel : $((3 * 4) + (7 / 2))$



Entwurfsmuster : Master – Worker

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

Master – Worker



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

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

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

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

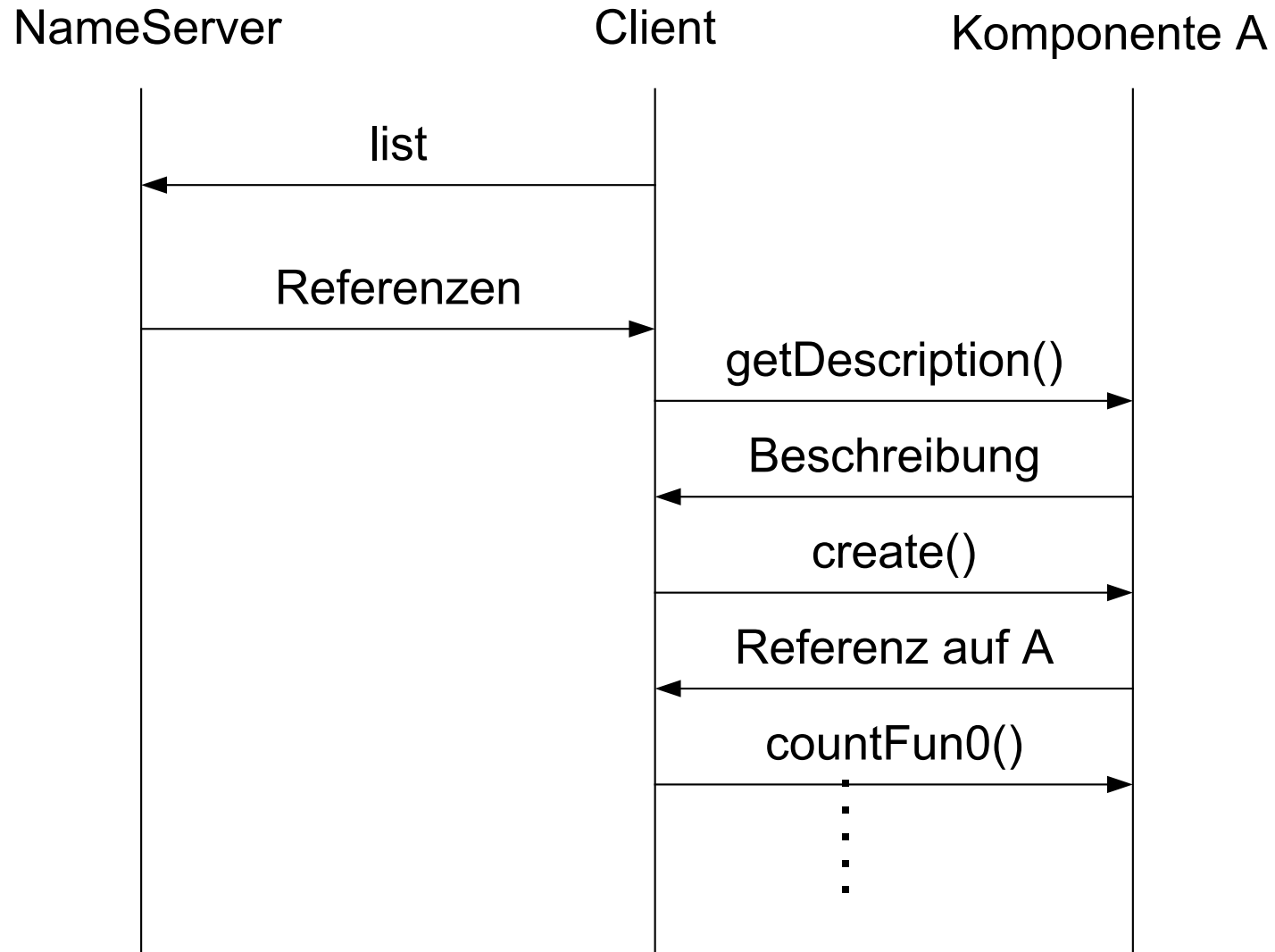
    string getNameFun0(in unsigned long index);
    string getNameFun1(in unsigned long index);
    string getNameFun2(in unsigned long index);

    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);

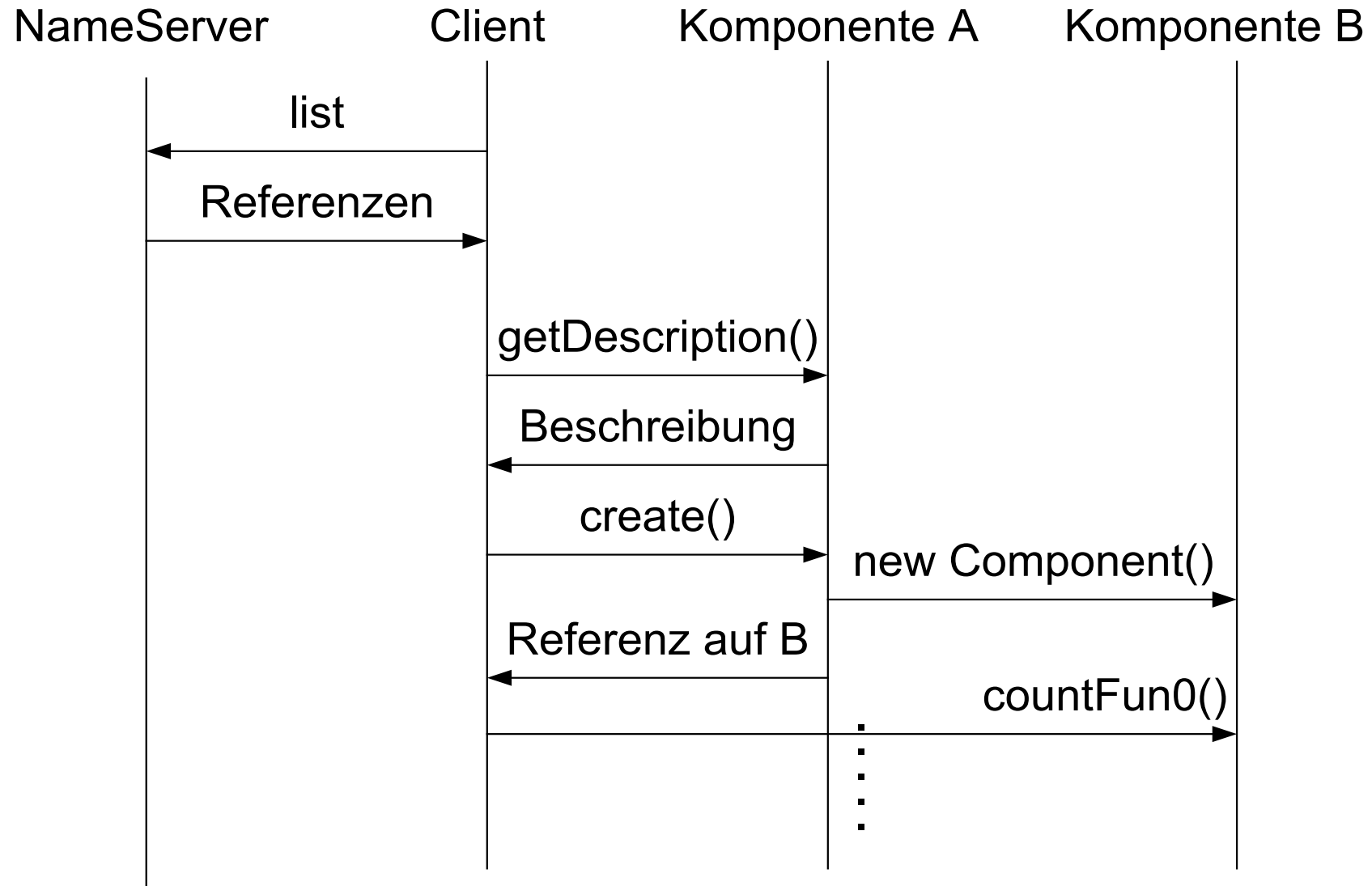
    Component create();
  };
};
```

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

Entwurfsmuster : Singleton



Nicht -Singleton



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

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

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

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

- Von CORBA verwenden wir lediglich den transparenten, entfernten Objektaufruf und den Nameserver.
 - DCOM, .NET etc bieten diese Funktionalität auch an
- Eine Entwicklung mit diesen anderen Frameworks ist also auch denkbar.

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

-
- 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

$E _ E + T$

$E _ T$

$T _ T * F$

$T _ F$

$F _ (E)$

$F _ i$