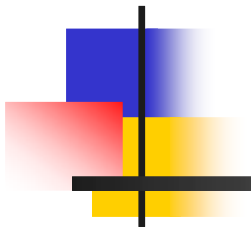
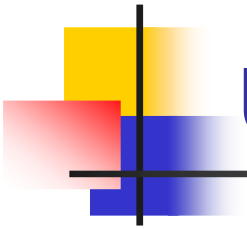


Stock Exchange

Nico Danneberg,
Paul Führing,
Martin Hammitzsch,
Lars Lindner



Überblick

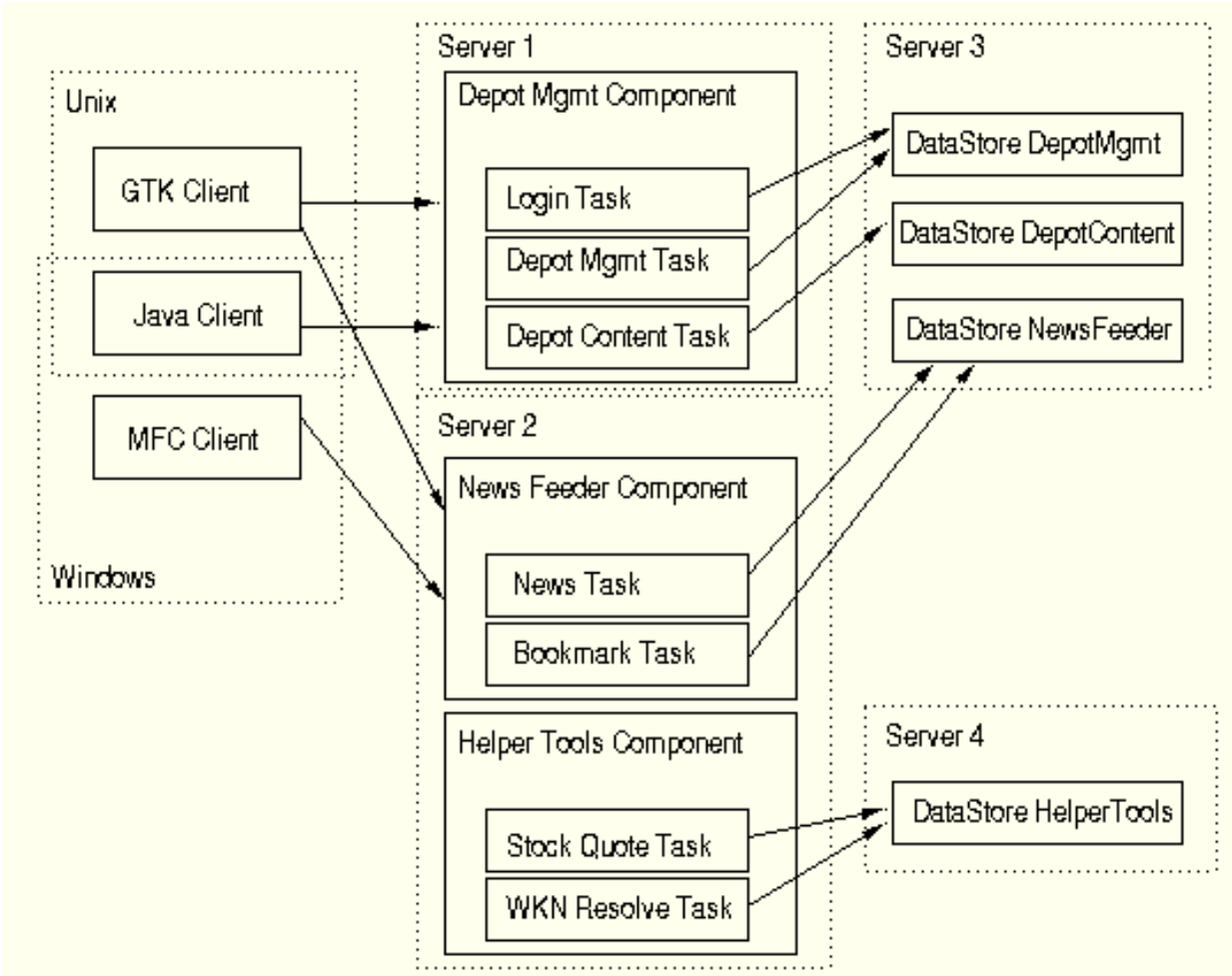
- Börsentool – Stock Exchange (SE)
- Verteilte Beispielkomponenten
  - Stehen bereit auf mehreren Servern
  - Nutzen verschieden verteilte Datenbestände
- Komponente stellt mehrere Funktionen (Tasks) zur Verfügung
- Komponenten durch Client-Applikation angezeigt

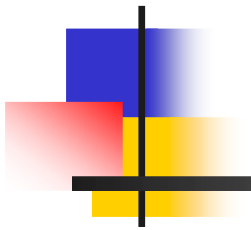
Vorstellung der Applikation

- Komponenten und Tasks
  - Depot Management
    - Login
    - Depots erzeugen, löschen
    - Depotinhalt verwalten
  - News Feeder
    - News anzeigen
    - Bookmarks verwalten
  - Helper Tools
    - Aktienkurs ermitteln
    - WKN Auflösung

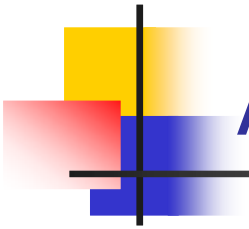
>>Präsentation

Systemübersicht





Analyse

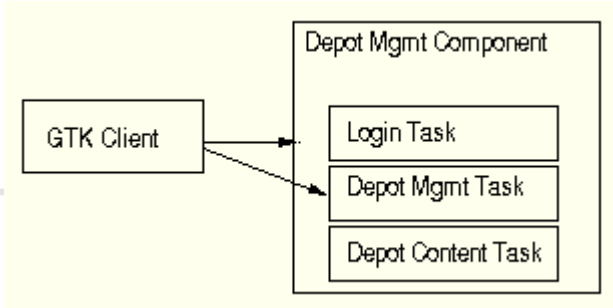


Anforderungen

Identifizierte Subsysteme:

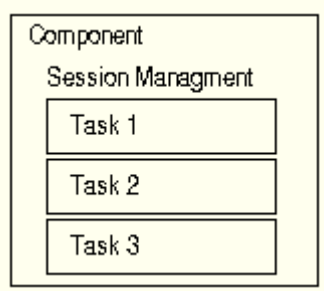
- Client
- Komponente
- Task
- Datenhaltung

Anforderungen – Client



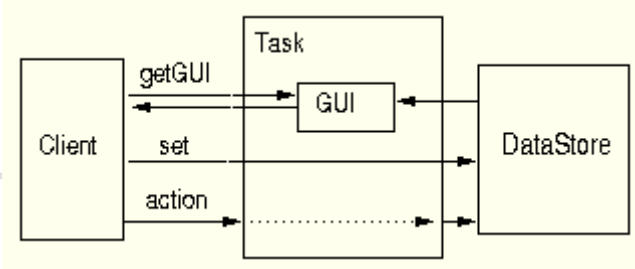
- Client kennt Komponenten
- Dynamisches Hinzufügen von Komponenten
- Verwaltet Session pro Komponente
- Fragt Komponenten nach verfügbaren Tasks
- Ermittelt GUI-Beschreibung des gewählten Tasks und stellt die GUI dar
- Sendet Updates bei veränderten GUI-Elementen zum Task
- Leitet Aktionsereignisse an Task weiter

Anforderungen – Komponente

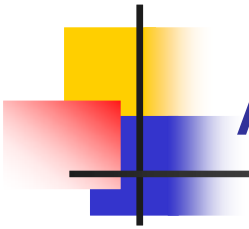


- Komponente kennt Tasks
- Weist Client Session zu
- Startet Tasks bei Clientanfrage
- Verwaltet zugreifende Sessions
- Komponente verwaltet Datenhaltung der Tasks
  - Startet Datenhaltung, wenn von Task benötigt
  - Beendet Datenhaltung, wenn von keinem Task mehr benötigt

Anforderungen – Task

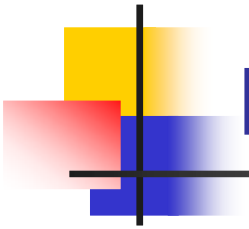


- Task gibt GUI-Beschreibung an Client
- Bearbeitet Aktionsanfragen vom Client
- Leitet von Client generierte Ereignisse für Datenhaltung an diese weiter



Anforderungen – Datenhaltung

- Ermöglicht Lesen und Schreiben von Daten
- Konsistente Benutzung durch mehrere Tasks

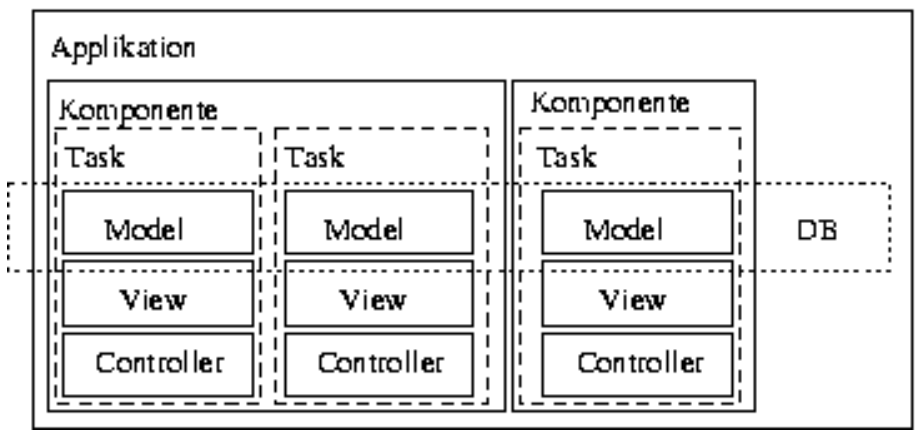


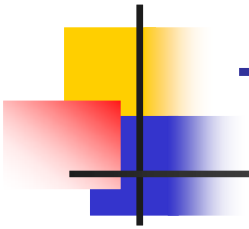
Design Pattern

- Model View Controller (MVC)
- Three-Tier
- Factory
- Sessions

MVC

- Task umfasst
  - Von Komponente zugeordnete Datenhaltung (Model)
  - GUI-Beschreibung (View)
  - Logik (Controller)
- Model realisiert durch Datenhaltung
- View und Controller realisiert durch Task





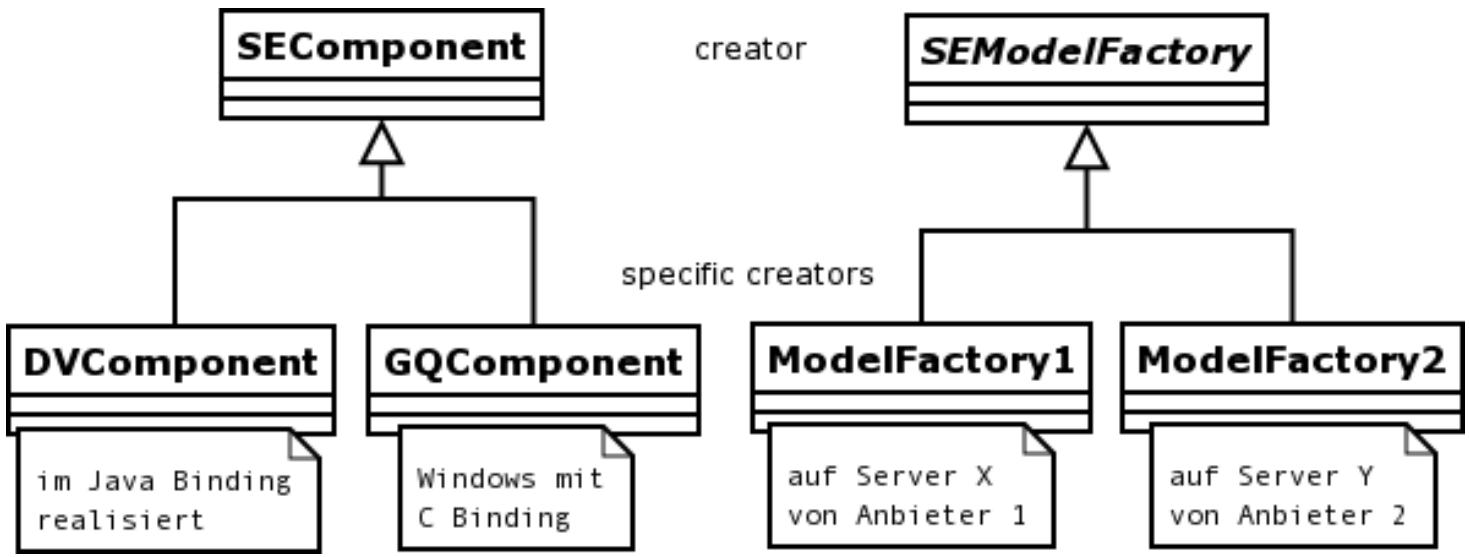
Three-Tier

- Client-Tier durch Client ...
- Application-Tier durch Komponenten ...
- Database-Tier durch Datenhaltung ...

... realisiert

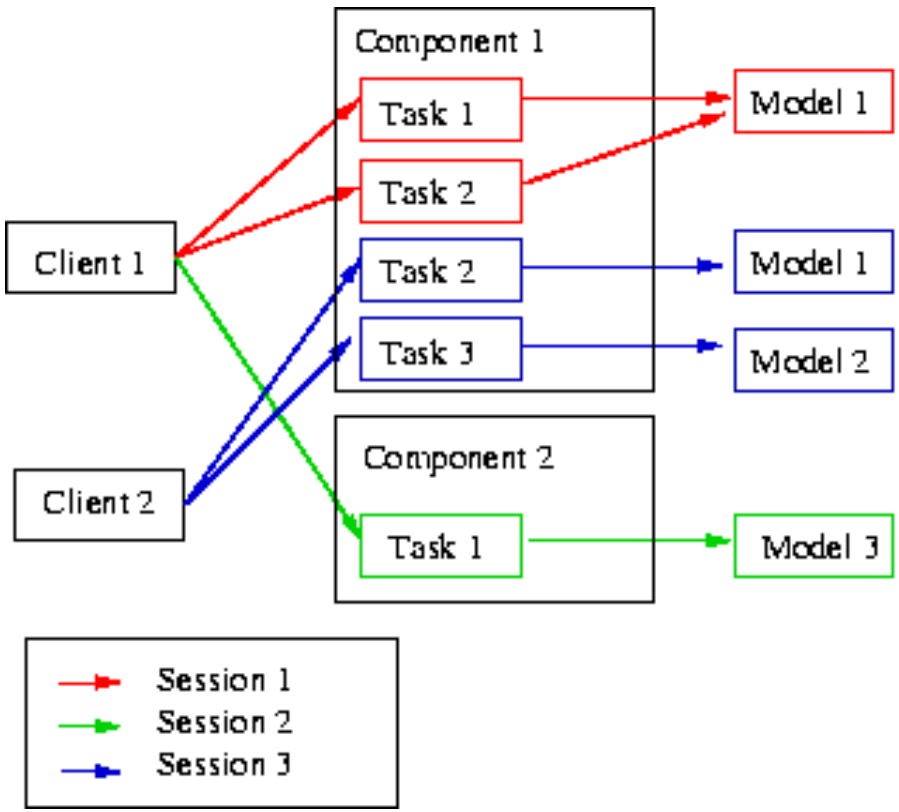
Factory

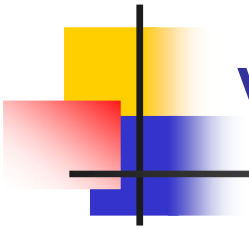
- Jede Komponente ist eine Task-Factory
- Datenhaltung ist eine Model-Factory



Sessions

- Komponente verwaltet Sessiontabelle und ordnet Models den Tasks zu





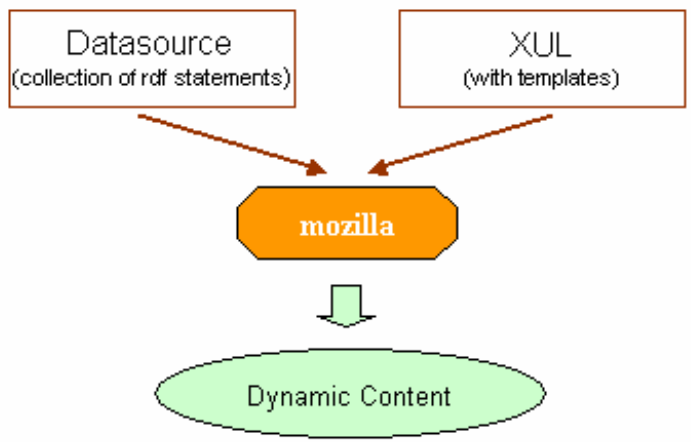
Vergleichbare Lösungen

- Mozilla – XUL/RDF Architektur
- Jax Front
- Glade (RAD-Tool für GNOME)

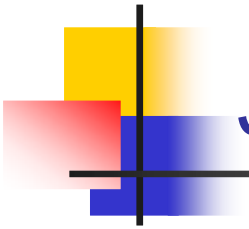
Mozilla – XUL Architektur

Basis für Entwicklung von plattformüber-greifenden Applikationen (www.mozilla.org/docs)

- Vorteile
  - Nutzt XML für GUI (XUL) und Daten (RDF)
  - Verteilung von User-Interface-Komponenten
  - Verschiedene Toolkits (XFPE, Phoenix)
- Nachteile
  - Noch kein Standard



http://www.shalabh.com/creations/xml-in-mozilla/xml-in-mozilla_files/v3_document.htm

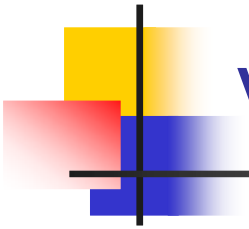


Jax Front

Dynamisches UI basierend auf XML

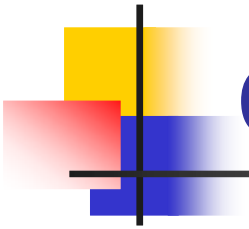
(www.jaxfront.com)

- Vorteile
  - GUI-Spezifikation in XML
  - GUI Darstellung Swing und HTML
- Nachteile
  - Kommerziell



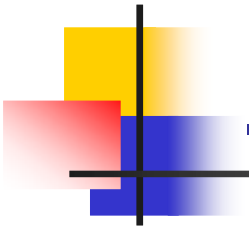
Vergleich der Middleware-Konzepte

- COM+
- .NET
- J2EE
- CORBA



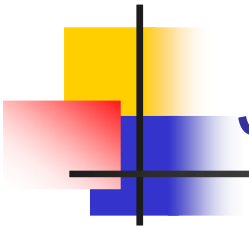
COM+

- Vorteile
  - Effizient (fest implementierte Mechanismen)
  - Relativ sprachunabhängig
- Nachteile
  - Plattformabhängig
  - Fehleranfällig



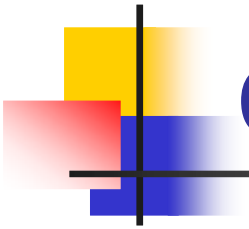
.NET

- Vorteile
  - Guter Support (sehr viele Hilfequellen)
  - Hohe Akzeptanz (sehr bekannt, Standard)
  - Relativ sprachunabhängig und interoberabel
  - Konfigurierbare Protokolle u. Übertragungsformate
- Nachteile
  - Plattformabhängig
  - Einige Dienste laufen über COM+ Services



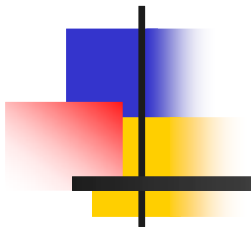
J2EE

- Vorteile
  - Plattformunabhängig
  - Hohe Marktdurchdringung
  - Offener Standard
  - Komponentenunterstützung mit Deployment
- Nachteile
  - Sprachabhängig (aber etabliert und anerkannte Sprache, Einbindung anderer Sprachen über CORBA möglich)



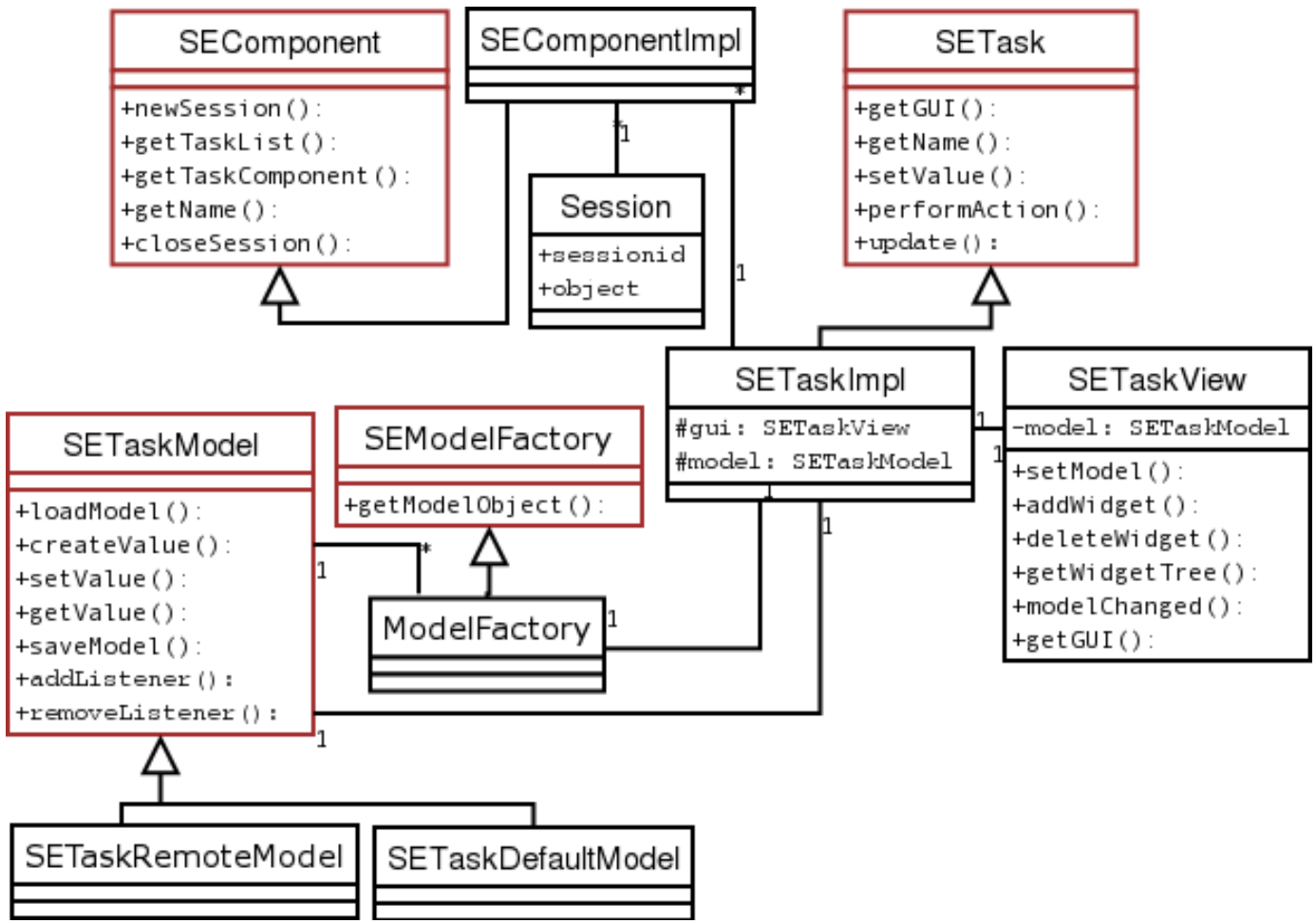
CORBA

- Vorteile
  - Plattformunabhängig
  - Sehr viele Sprach-Bindings verfügbar
  - Wenig Overhead bei Programmierung
  - Offener Standard
  - Etabliert und anerkannt
- Nachteile
  - Erhöhter Programmieraufwand für anspruchvollere Mechanismen

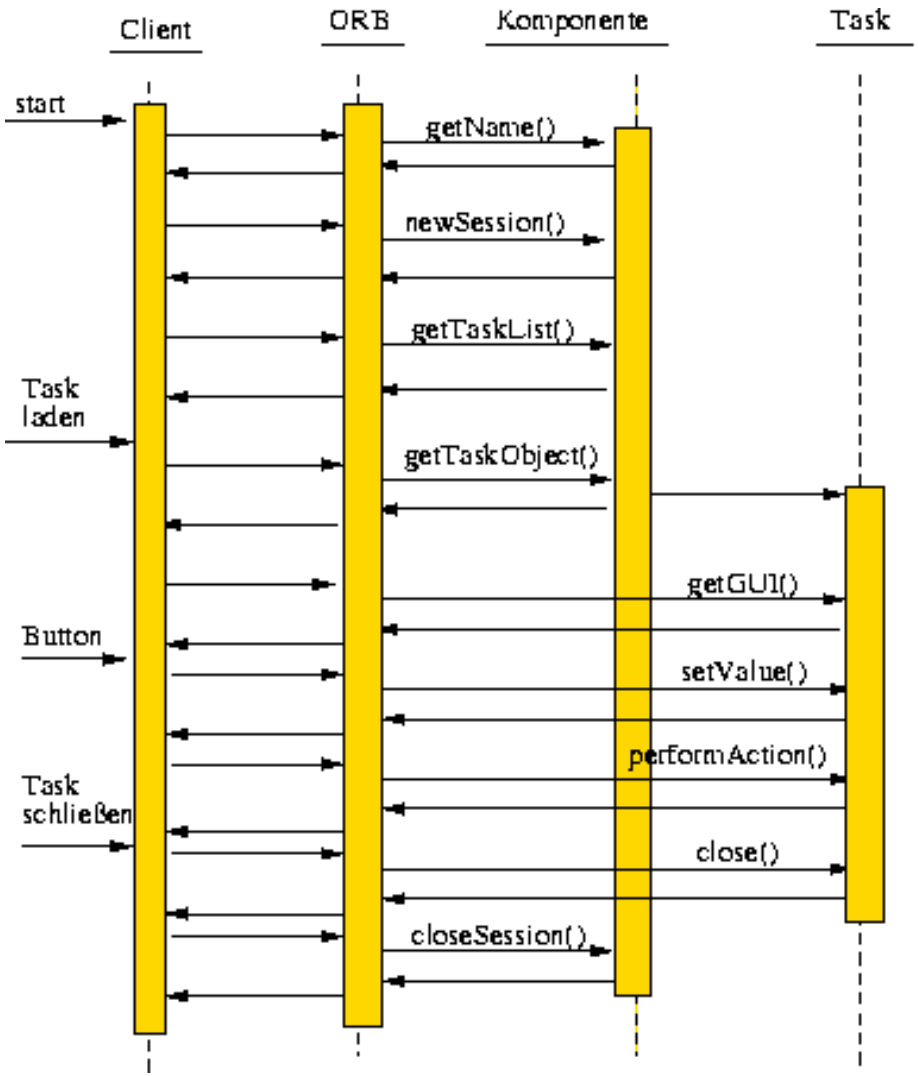


Design

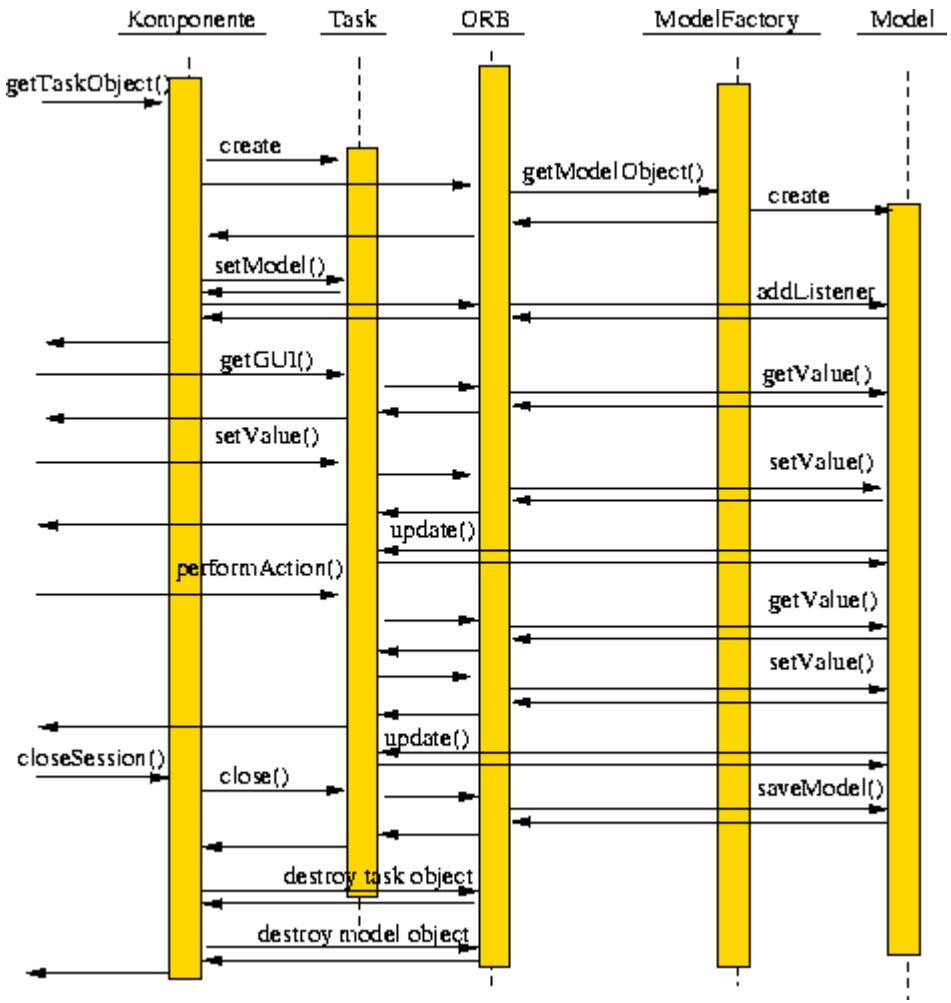
Entwurf – Klassendiagramm

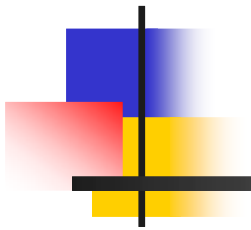


Entwurf – Anfordern eines Tasks

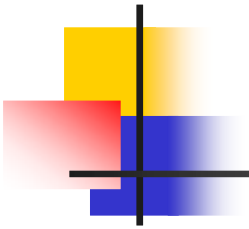


Entwurf – Arbeiten mit Daten





Implementierung



Implementierung

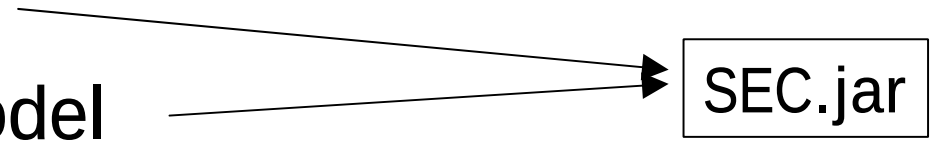
- Wie wurde entwickelt?
- Implementierung der Basisklassen
- Komponentenimplementierung:
  - Entwickeln einer Komponente
  - Entwickeln eines Tasks
- Benutzte Fremdsoftware
- Applikationsdetails
- Wie wurde getestet?

Wie wurde entwickelt?

- Entwicklung in drei Ausbaustufen
  1. Stufe
     - Lokale Datenhaltung
     - Keine Sessions
     - Task nicht nach MVC
  2. Stufe
     - Designerweiterung nach MVC
  3. Stufe
     - Sessionverwaltung
     - Modelfactory mit Remote Models

Implementierung der Basisklassen

- Allgemeinen Komponenten- und Task-Klassen im Java Binding
- Lokales Test-Model



- Test-Komponente inklusive Tasks zur Ausnutzung aller Features
- Spezifische Models
- Spezifische Komponenten (z.B. GetQuote mit SOAP)
- Anleitung …

Komponentenimplementierung

1. Basisbibliothek kopieren (SEC.jar)
2. Konfiguration der Komponente (component.properties)

```
nsname =      DVComponent
name   =      Depot Managment Component
tasks  =      LoginTask,\
              DepotMgmtTask,\
              DepotEditorTask
```

3. Tasks installieren
4. ORB und Komponente starten

```
java -cp .:../SEC.jar SEComponent -ORBInitialPort 1678
```

Taskimplementierung

1. Konfiguration des Tasks (CalcTask.properties)

```
ModelFactory = corbaname::localhost:1678#ModelFactory
ModelName = FileSaveModel
```

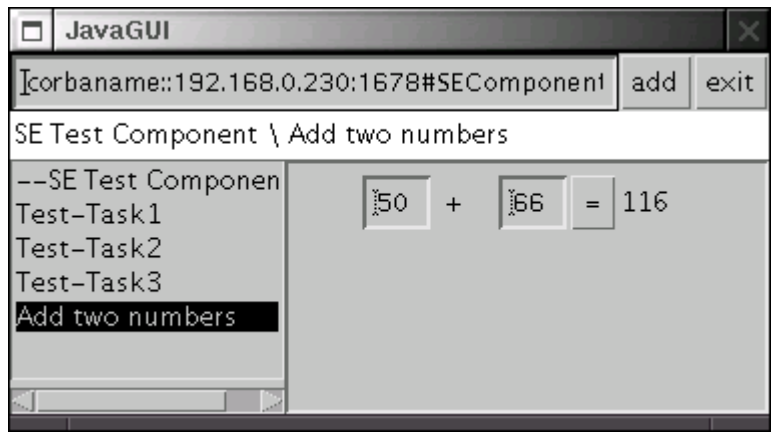
2. GUI-Beschreibung

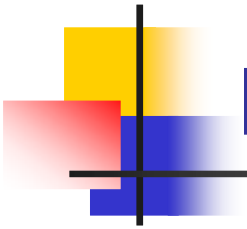
```
<?xml version="1.0"?>
<layout>
    <hlayoutmanager/>
    <input   id="1"/>
    <hlayoutmanager/>
    <label   value=" + "/>
    <hlayoutmanager/>
    <input   id="2"/>
    <hlayoutmanager/>
    <button  id="3" value="="/>
    <label   id="4"/>
</layout>
```

Taskimplementierung

3. Implementierung des Tasks

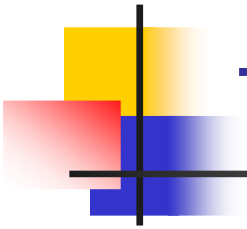
```
import org.omg.CORBA.StringHolder;
public class CalcTask extends SETaskImpl {
  int INPUT1 = 1; int INPUT2 = 2; int EXEC = 3; int RESULT = 4;
  CalcTask() {
    super(); name = "Add two numbers";
  }
  public void init() {
    super.init();
    model.createValue(INPUT1, "0");
    model.createValue(INPUT2, "0");
    model.createValue(RESULT, "0");
    gui.loadFromXML("CalcTaskView.xml");
  }
  public int performAction(int id) {
    if(id==EXEC){
      StringHolder sh = new StringHolder();
      model.getValue(INPUT1, sh); int a = Integer.parseInt(sh.value);
      model.getValue(INPUT2, sh); int b = Integer.parseInt(sh.value);
      model.setValue(RESULT, ""+(a+b));
    }
    return 0;
  }
}
```





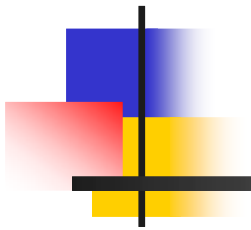
Benutzte Fremdsoftware

- XML Parser (kxml.enhydra.org, Uni Dortmund)
- ORBs:
  - Java ORB
  - OmniORB
  - ORBit2
- SOAP-Zugriff für Kursermittlung:
  - JavaMail 1.3
  - JAF 1.02
  - Apache SOAP 2.3

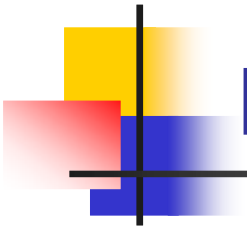


Testkonzept

- Tracing mit Debug-Ausgaben auf Konsole
- Regressionstest mit zufälligen und kritischen Daten für alle IDL Schnittstellen und Modelimplementationen
- Regressionstest der CORBA-Kommunikation nach Use Cases der Anforderungsanalyse



Abschließende Worte

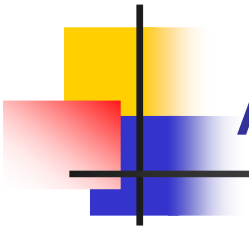


Probleme bei der Implementierung

- JavaClient: Dynamische Änderung des Widgetbaums nicht möglich mit Swing, nur als Workaround mit AWT.
- ORBit2: Unterstützt kein NamingService, Abhilfe durch Implementierung eines NamingService-Proxys.
- GUI-Beschreibung: Übertragung als rekursive Struktur zwischen verwendeten ORBs fehlerhaft, Übertragung als Liste des einheitlich traversierten Baums.

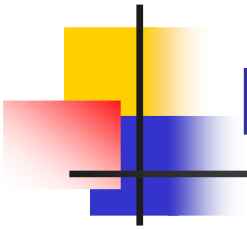
Was würden wir besser machen?

- Client nie wieder mit MFC
- Komponenten / Model nur mit Java + CORBA
- CORBA Debugging Tools einsetzen
- Versionsverwaltung verwenden



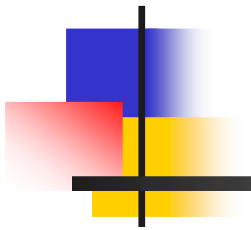
Ausblick

- Eigene Subkontexte im NamingService
- Weitere Widgets unterstützen, z.B. Listboxen
- Attribute für Widgets unterstützen, z.B. Layout
- Model mit Datenbankanbindung
- CORBA-Callback in Client, um z.B. Taskwechsel durch Tasks auszulösen
- GUI-Verschachtelung, z.B. Task nutzt GUI eines anderen CORBA-Objektes
- Verschlüsselte Datenübertragung



Fazit

- CORBA war gut geeignet für Projekt
- XP gut geeignet für Programmierer mit unterschiedlichem Wissen



Stock Exchange

Nico Danneberg,
Paul Führing,
Martin Hammitzsch,
Lars Lindner