Postmoderne Softwareentwicklung
Methoden und Rezepte

Dr. Thomas Wolff

Übersicht / Themen

- Was sind Softwareentwicklungstechniken?
- Patterns
- ein spezialisiertes Produkt: hochverfügbare Services
- Aspekte eines Framework für hochverfügbare Systeme
- Programmierbeispiel

Softwareentwicklungstechniken

Entwicklungsmethoden (OOA, OOD)

Notationen
Tools
Schemen / Guidelines, die den kreativen Prozess unterstützen

Patterns


generische Lösungen wiederkehrender Design-Probleme durch gesammelte Darstellung kollektiver Erfahrung in schematisierter, leicht zugänglicher Form

Erweiterte Programmierumgebungen

Standardbibliotheken (Java)
ergänzende Bibliotheken für bestimmte Anwendungen

Frameworks für spezielle Zwecke
Komponentenmodelle

Konstruktionsstrategien

Stepwise Refinement
Programming by Contract

Entwicklungs- und Projektstrategien

Meilenstein-Modell
Waterfall Model
eXtreme Programming (kooperativ, evolutionär)

Patterns

Klassifizierung nach „Level“ des Einsatzes

Architektur-Patterns

- beschreiben Strukturaspekte von Softwaresystemen
- zeigen günstige Aufgabenverteilung auf Komponenten
- *Bsp. Model-View-Controller, Blackboard*

Design-Patterns

- bieten Ansatz für Realisierung einer Komponente
- Verfeinerung, Klassenstruktur
- *Bsp. Dispatcher, Datenmodellierungspatterns*

Implementierungs-Patterns, Idiome

- empfehlen Programmierungsmöglichkeiten
- auch sprachspezifisch
- *Bsp. Umgang mit Threads, I/O, Datenstrukturen, Iterator*

Klassifizierung nach Anwendungsumfeld

allgemein verwendbare Patterns
anwendungsspezifische Patterns
Framework-spezifische Patterns

Beschreibung

- griffiger Name
- Kurzbeschreibung des Zwecks und Einführungsbeispiel
- Problembeschreibung und Kontext
- Lösungsbeschreibung
- Implementierungshinweise, Beispiele und Varianten

Ein Architektur-Pattern

Model-View-Controller

Funktionsaufteilung einer interaktiven Anwendung

Problem

User Interface unterliegt häufigen Änderungswünschen, auch Anpassungsbedarf an einzelne Nutzer(gruppen).

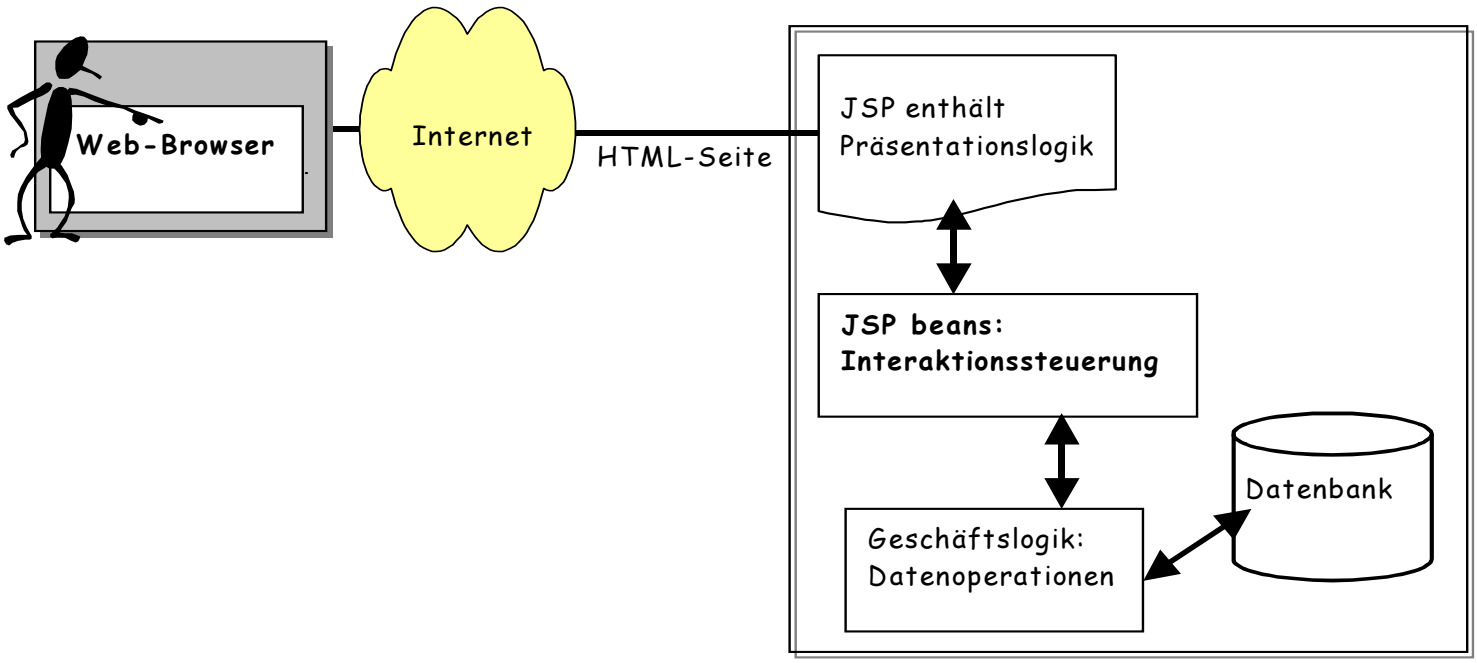
Bei „programmiertem GUI“ hoher Änderungsaufwand, fehlerträchtig wegen Nähe zur „Geschäftslogik“.

Lösung

Mehr-Säulen-Architektur („multi-tier“),

Präsentationslogik entkoppelt von Datenoperationen.

Dazwischen separat die Interaktionssteuerung.



Ein Programmierkultur-Pattern

(Programmierstil)

Das funktionale Idiom

```
public class Binärer_Suchbaum
{
    Knoten wurzel = null;

    public
    void einfügen (Schlüssel s) {
        wurzel = einfügen (wurzel, s);
    }

    private
    Knoten einfügen (Knoten k, Schlüssel s) {
        if (k == null) {
            return new Knoten (s);
        } else if (s < k.schlüssel) {
            k.links = einfügen (k.links, s);
            return k;
        } else if (s > k.schlüssel) {
            k.rechts = einfügen (k.rechts, s);
            return k;
        } else {
            // throw new SchlüsselSchonDrin ();
        }
    }
}
```

oder streng funktional mit Neuaufbau des Baumes:

```
            return new Knoten (k.schlüssel,
                               einfügen (k.links, s),
                               k.rechts);
```

Beispiel Industrieprojekt

Server für Telekommunikationsdienste

Bereich „Intelligente Netze“, Infrastruktur und Dienstlogik für besondere Funktionen im öffentlichen Telefonnetz

Besondere Eigenschaften

Ausfallsicherheit, *Hochverfügbarkeit*

Architektur

Cluster-Architektur

Software-Schichten:

- Hochverfügbarkeits-Plattform (Kommunikation, Prozessüberwachung, redundante Datenhaltung für Prozesse)
- Plattform-Abstraktionsschicht (Framework)
- Zentrale Plattform-Funktionen (Logging, Webserver etc.)
- Applikationen mit Dienstlogik (Telefonie, Internet)

Programmierumgebung

Sprachen: überwiegend Java, außerdem SDL, C++

Probleme:

- Unberechenbare Garbage Collection behindert definierte Antwortzeiten.
- Kleinteilige Objekterzeugung fragmentiert Speicher und erfordert häufigere Garbage-Collection – Strategie zur Vermeidung von Objekterzeugungen (recycelnde Factories, Programmierdisziplin). Konflikt mit OO.
- Sessions bestehen aus aufeinanderfolgenden Request-Bearbeitungen; viel mehr Sessions als Prozesse, daher und wegen Verfügbarkeit eigenes Kontextsicherungskonzept.

spezielles Framework und Programmiermodell

Einsatz von Entwicklungstechniken im Industrieprojekt

Designmethode UML

Tool und Notation: Dokumentation von Interface und Design

Use Cases

- Ausdruck von Requirements an eine Komponente
- Beschreibung der Features einer Komponente
- Verhandlung des Interfaces

Sequence Diagrams

- Dokumentation des dynamischen Verhaltens eines Interfaces
- Dokumentation und Konformitätsanalyse der externen Interaktion einer Komponente

Class Diagram

- Komponentendesign

Framework

spezifische Patterns für

- Ablaufsteuerung und Komponenteninteraktion
- Zugriff auf hochverfügbare Anwendungsvariablen

Konstruktion des Softwaresystems

Spezifikation und Review von Funktionalität und Interface

- Programming by Contract auf Komponenteninterface-Ebene

Entwicklungsprozess

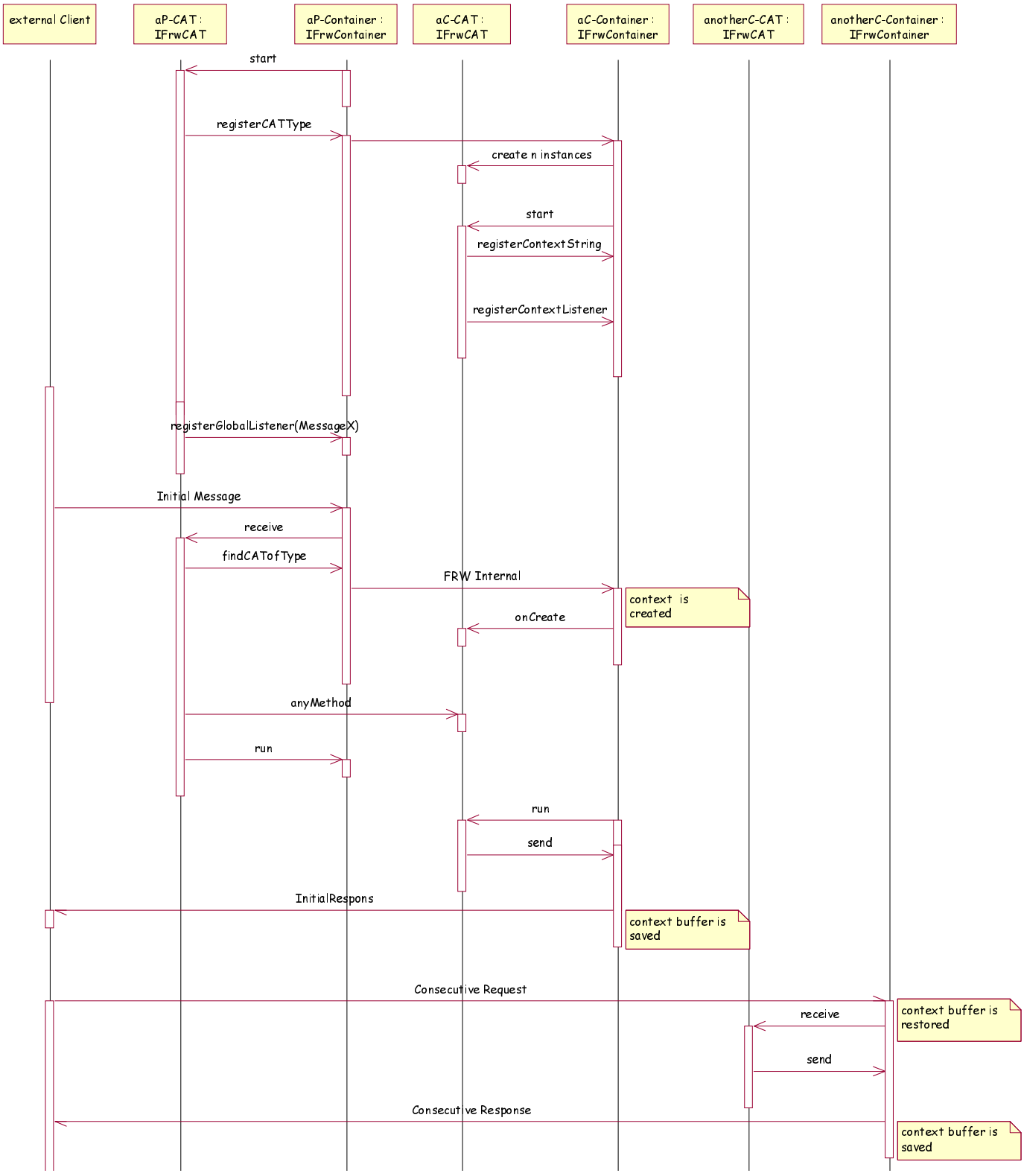
Meilensteinmodell, ISO 9000

- Spezifikation und Dokumentation
- Erreichen definierter Qualitätsziele
- strenges mehrstufiges Testverfahren

Beispiel 0 (sequence diagram)

Prozesse und Sessions im Framework

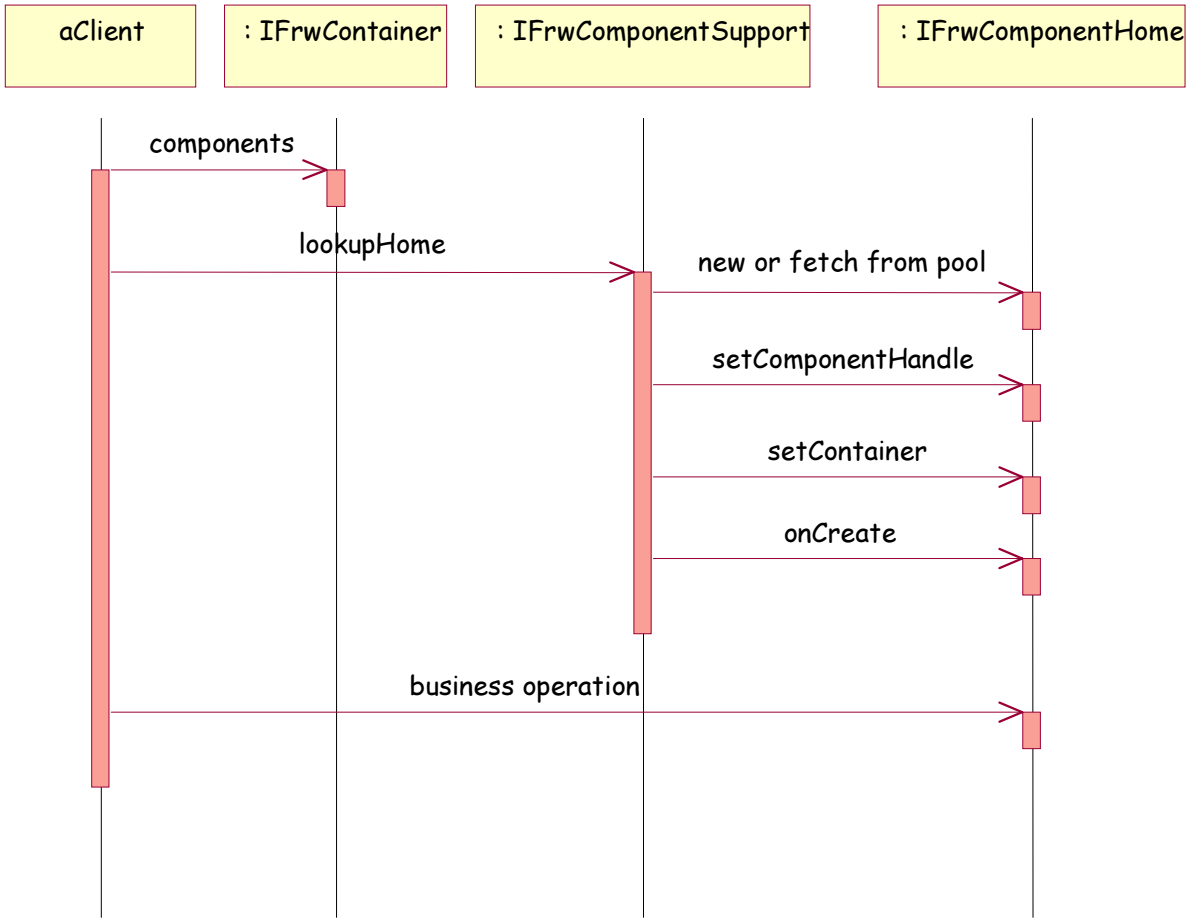
Service distribution:



Beispiel 1 (sequence diagram)

Komponentenmodell im Framework

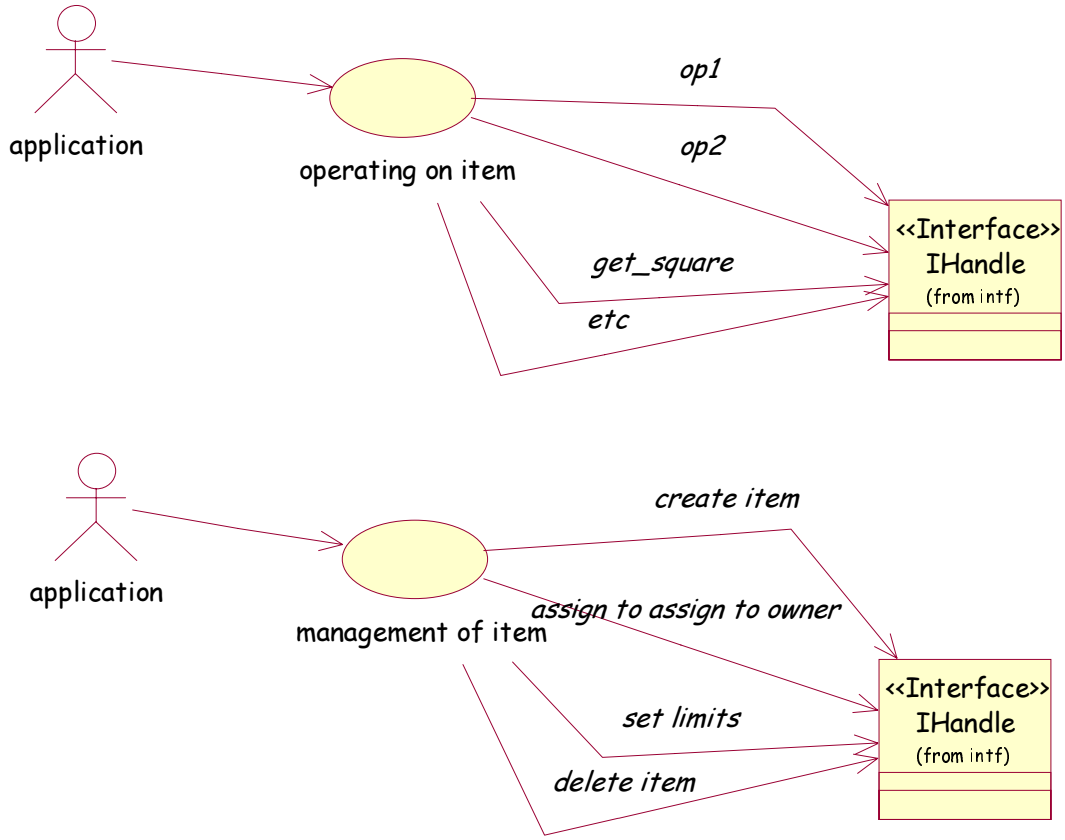
Component instantiation:



Beispiel 2 (use case diagram)

Funktionalität einer Komponente XY

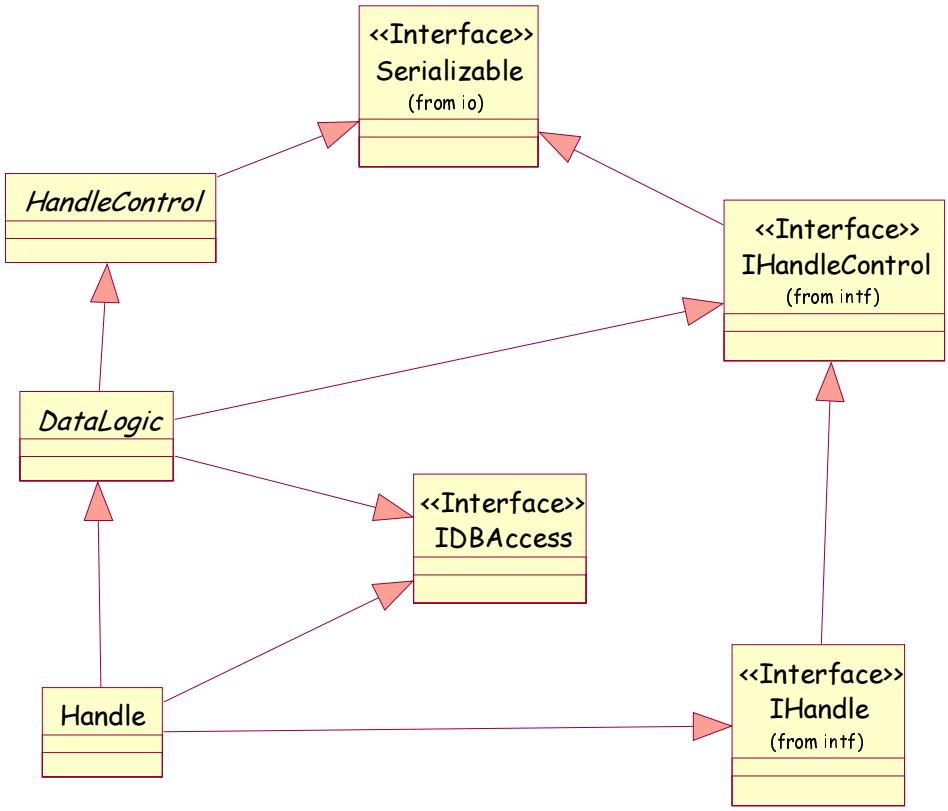
Usage of XY data handles:



Beispiel 3 (class diagram)

Implementierung einer Komponente XY

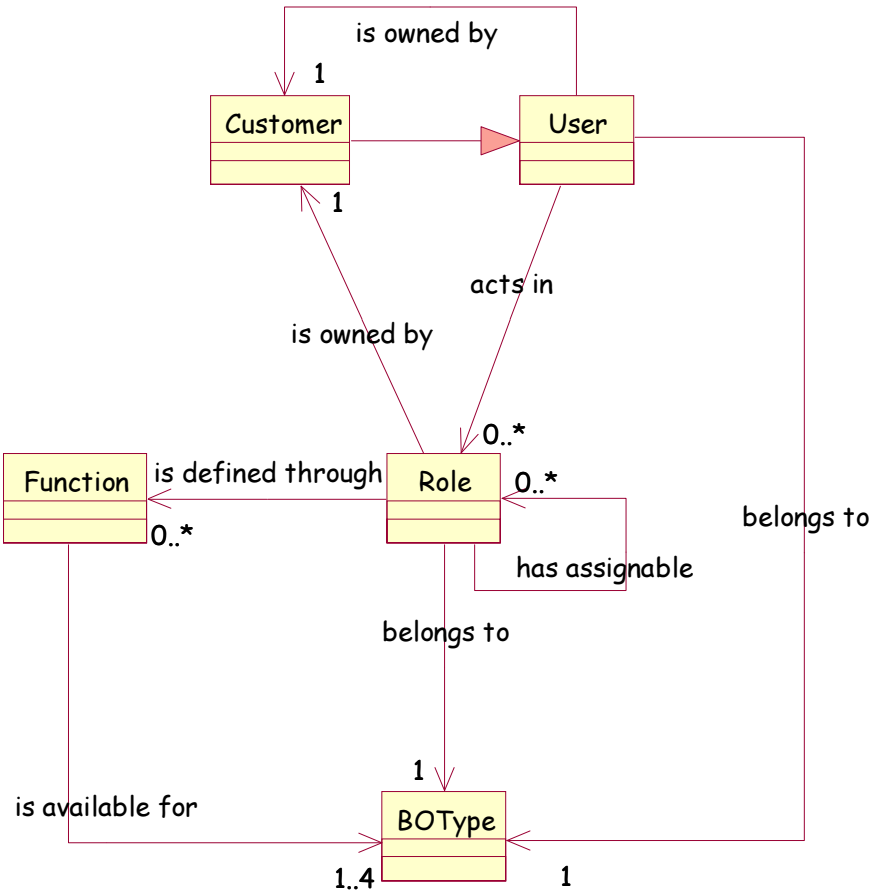
XY class structure design:



Beispiel 4

Klassendiagram als Datenmodell-Beschreibung

User Management component entities:



Programmierbeispiel 1

Factory für Komponente XY

```
public class XYHandleFactory
implements IComponentHome, IFactory
{
    private
    IContainer _container;

    public
    void set_container (IContainer container)
    {
        _container = container;
    }

    private
    Handle [] _handles;

    /**
      Setup of n handles. Called by component users.
     */
    public
    void start (int n)
    {
        _handles = new Handle [number];
        for (int i = 0; i < _handles.length; i ++) {
            _handles [i] = new Handle (_container, ...);
        }
    }

    /**
      Invoked by Framework when the container terminates.
      Resources must be released.
     */
    public
    void on_container_termination ()
    {
        for (int i = 0; i < _handles.length; i ++) {
            if (_handles [i].is_in_use ()) {
                _handles [i].cleanup ();
            }
        }
    }
```

Programmierbeispiel 1

noch Factory für Komponente XY

```
        /**
          Acquire one of the configured handles.
          @param tag the index of the registered handle to use
          @param id the data ID to use by the handle
         */
        public
        IHandle get_handle (int tag, long id)
        throws FrameworkException
        {
            if (i < 0 || i >= _handles.length) {
                throw new HandleException (FAC_no_handle);
            }

            Handle handle = _handles [i];
            if (handle.is_in_use () && (handle.getID () == id)) {
                handle.continue_use (_container);
                return handle;
            } else if (! handle.is_in_use ()) {
                handle.new_use (_container, id);
                return handle;
            } else {
                throw new HandleException (FAC_handle_in_use);
            }
        }

        /**
          Release handle. Free resources (e.g. to their pools).
         */
        public
        void release (IHandle handle)
        {
            ((Handle) handle).cleanup ();
        }
    }
```

Programmierbeispiel 2

Framework-Einbettung eines XY-Objekts

```
abstract class HandleControl
implements ITimerListener
{
    /**
      Setup context for initial or continuing handle use.
     */
    private
    void set_context (IContainer container) {
        _container = container;
        _container.setup_timer (_timer_id, this);
    }

    /**
      Invoked on timer time-out via registered ITimerListener.
     */
    abstract
    void timeout ();

    synchronized
    void new_use (IContainer container, long id) {
        set_context (container);
        _container.set_context_var („handle id“, id);
    }

    synchronized
    void continue_use (IContainer container) {
        set_context (container);
    }
}

public class Handle
extends HandleControl
{
    public
    long get_square () {
        long id = _container.get_context_var („handle id“);
        long value = DB.lookup_XY (id);
        return value * value;
    }
}
```