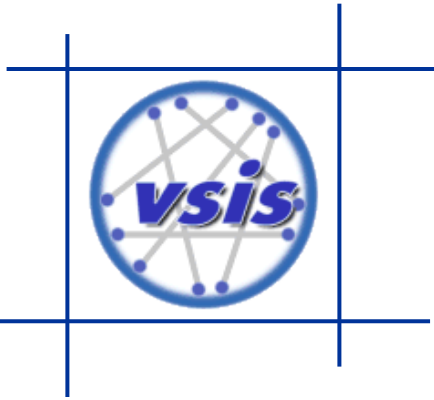
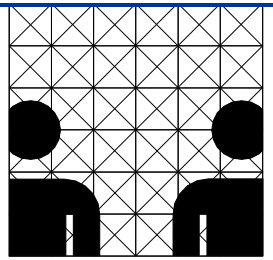


Praktikum

# Datenbanken und verteilte Systeme
# Jadex Einführung

SoSe 2013

Kai Jander

5. Aug. 2012

Verteilte Systeme und Informationssysteme (VSIS)
Fachbereich Informatik
Universität Hamburg


# Jadex Active Components Middleware: Überblick

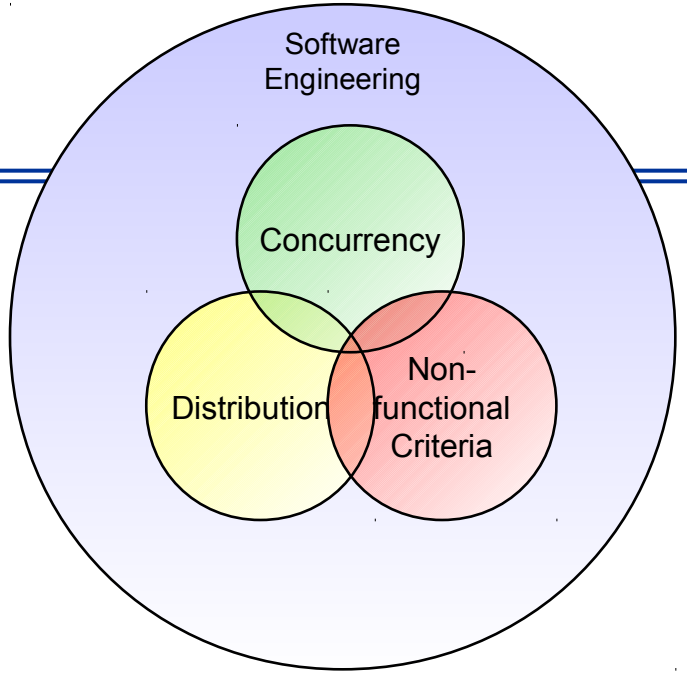
- Entwicklung als Open Source (http://www.activecomponents.org)
  - Uni Hamburg: Entwicklung und Einsatz in Forschung und Lehre
  - Zahlreiche Verwendungen durch dritte (Lehre, Forschung, Industrie)


- Programmiermodell
  - Kombination von SCA mit Agenten
  - Ausführungsinfrastruktur und Werkzeuge

- Wiederverwendbarkeit
  - Interne Architekturen: vorgefertigte Verhaltensmodelle (Agenten, Workflows, ...)
  - Verhandlungsprotokolle: vorgefertigte Interaktionsmuster (z.B. Auktionen)
  - Fließender Übergang zwischen Simulation und Realbetrieb

- Standards
  - Fokus auf Mainstream-Technologien
  - Flexibilität für Kommunikation, Implementation und Ausführungsumgebung

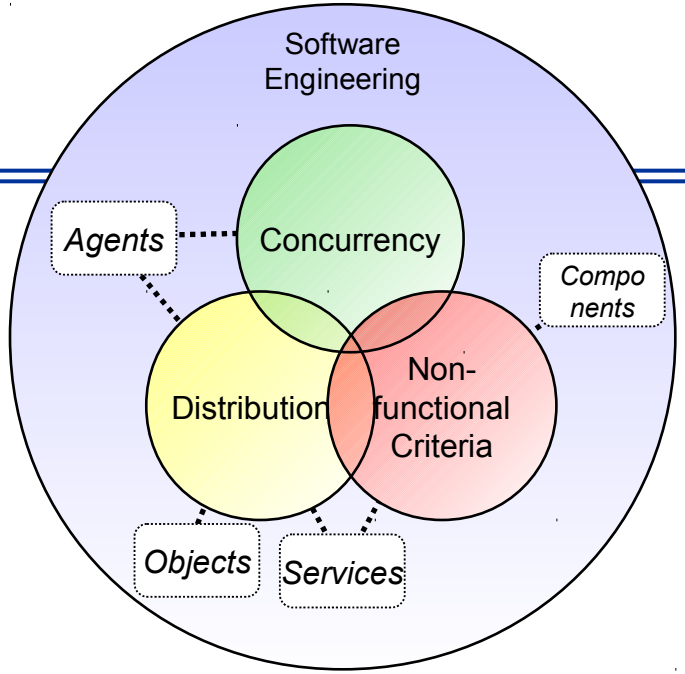
# Herausforderungen

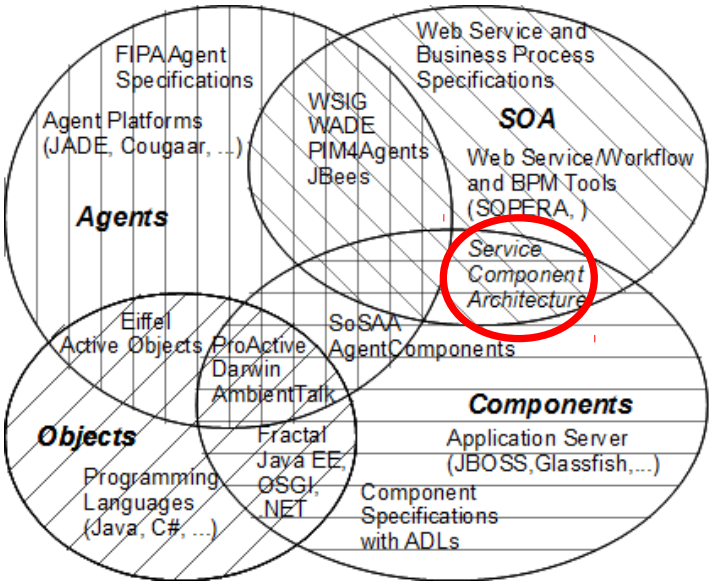



- Verteilung
  - Probleme: Unabhängige Knoten, Heterogenität
  - Ziele: Transparenz, Interoperabilität
- Nichtfunktionale Eigenschaften
  - Technisch: Fehlerbehandlung, Skalierbarkeit, Sicherheit, ...
  - Business-Sicht: SLAs bezüglich Sicherheit, Verfügbarkeit, ...
- Nebenläufigkeit
  - Gewünschte vs. inhärente Nebenläufigkeit
  - Konsistenz vs. Deadlocks
- Software Engineering
  - Modularisierung und Wartbarkeit
  - Portabilität, Erweiterbarkeit

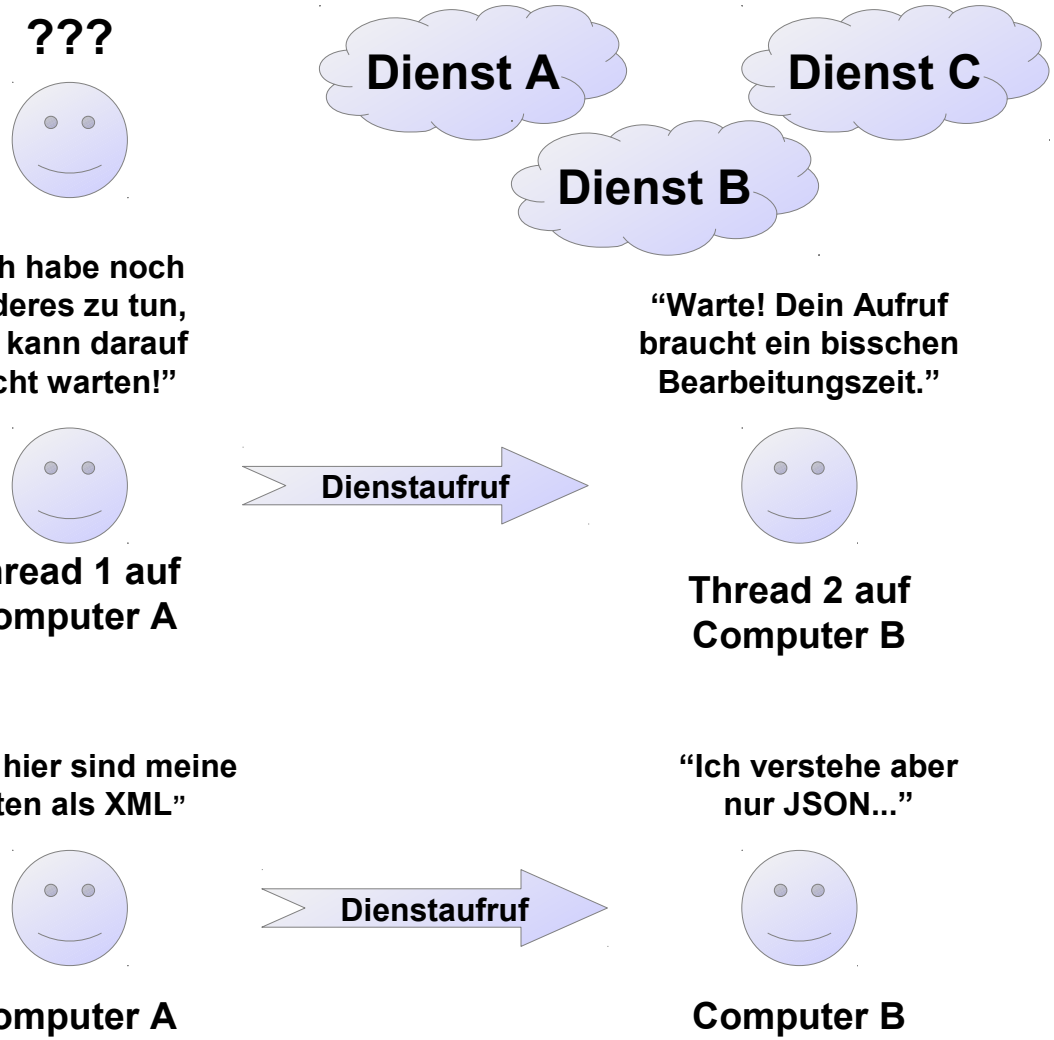
# Programmierparadigmen

- Objektorientierung
  - Intuitive Abstraktion für Realweltobjekte
- Komponenten
  - Wiederverwendbare Bausteine
  - Externe Konfiguration
  - Management-Infrastruktur
- Serviceorientierte Architektur (SOA)
  - Einheiten, die Geschäftsvorghänge umsetzen
  - Service-Registries, dynamisches Binden
  - SLAs, Standards (z.B. bzgl. Sicherheit)
- Multiagentensysteme (MAS)
  - Einheiten, die aufgrund lokaler Ziele agieren
  - Autonome Akteure, nachrichtenbasierte Koordination
  - Reagieren auf Ereignisse in einer dynamischen Umgebung



# Warum aktive Komponenten im Praktikum?
# 3 Probleme von verteilten Systemen



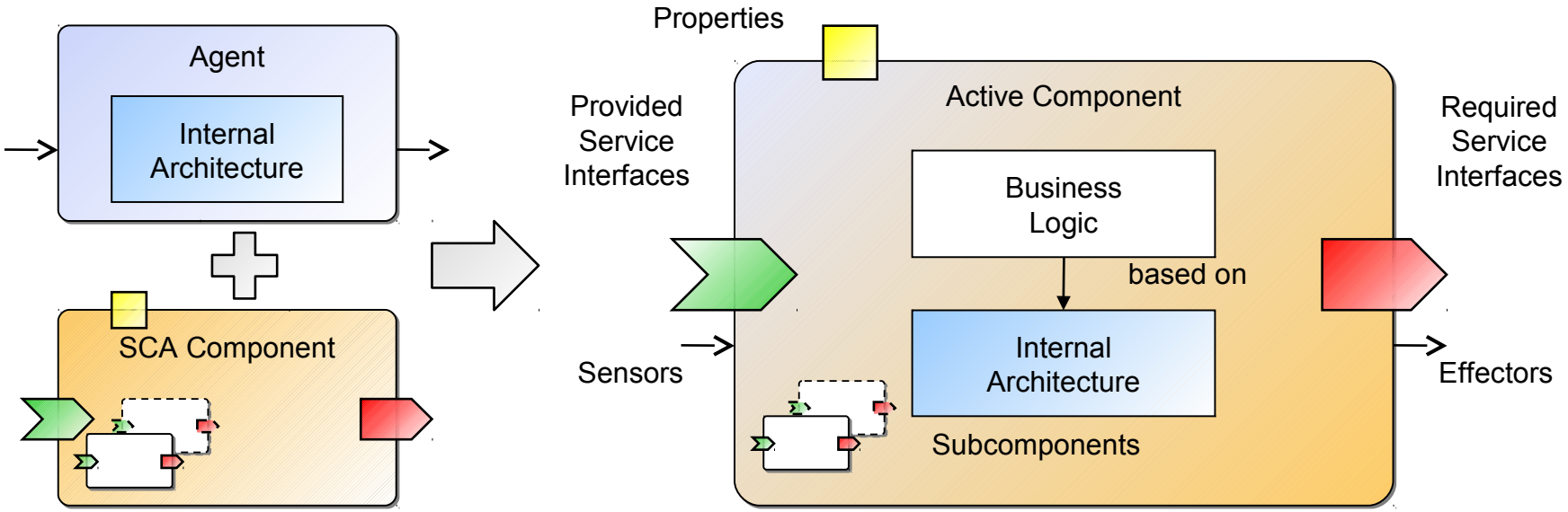

**Wie finde ich entfernt aufrufbare Dienste?**

**Wie synchronisiere ich aufrufe ohne Threads anzuhalten?**

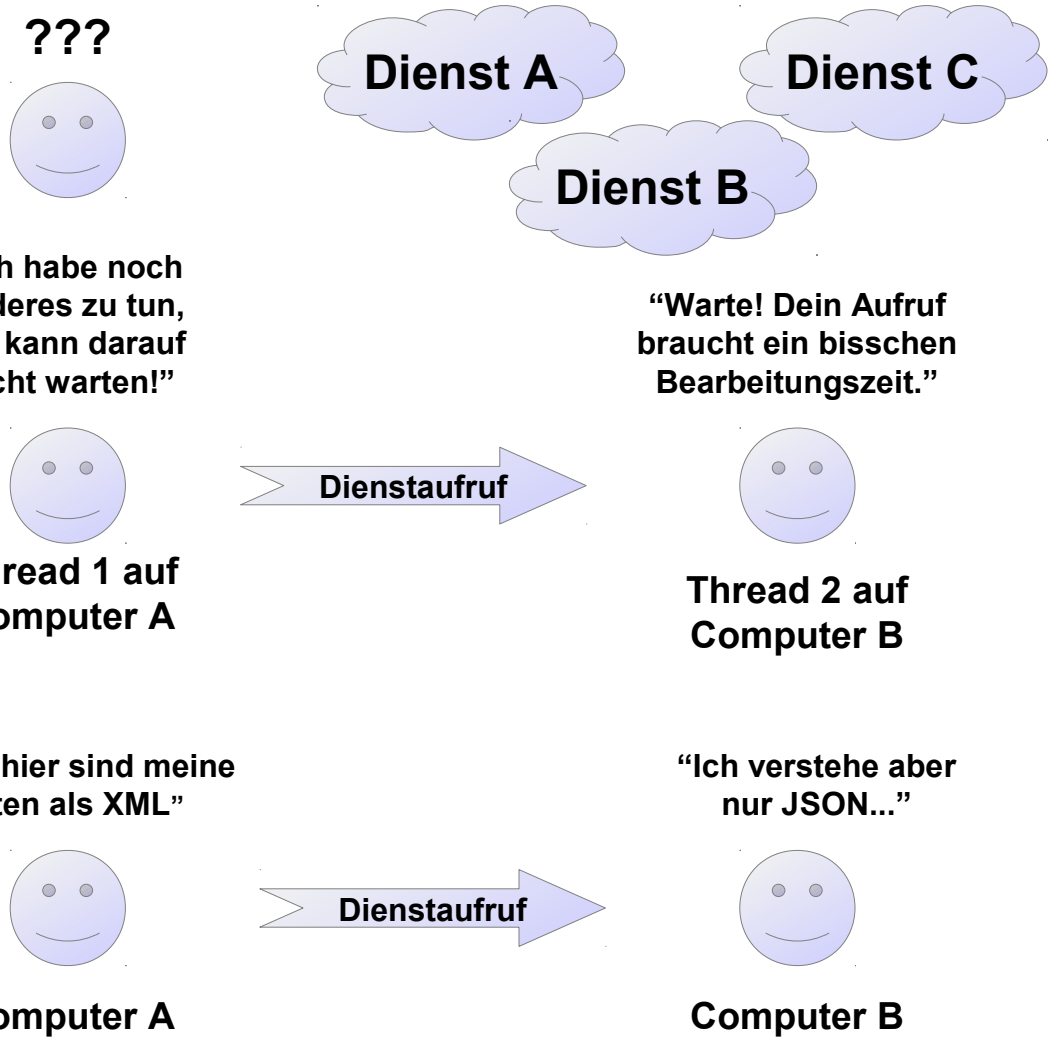
**Wie codiere ich meine Daten für den Aufruf?**

# Jadex Programmiermodell: Komponentenstruktur

- Inkrementeller Ansatz
  - Vorteile bestehender Ansätze beibehalten
  - Zusammenbringen von SCA mit Agenten, d.h. Komponenten + Services + Agenten



# Warum aktive Komponenten im Praktikum?
# 3 Probleme von verteilten Systemen




**Wie finde ich entfernt aufrufbare Dienste?**

Jadex Awareness und Servicesuche

**Wie synchronisiere ich aufrufe ohne Threads anzuhalten?**

Asynchrone Aufrufe und Futures

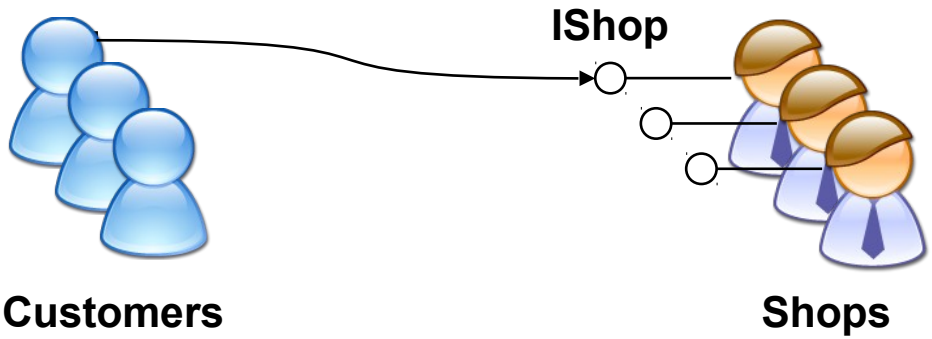
**Wie codiere ich meine Daten für den Aufruf?**

Erledigt Jadex automatisch für Java-Beans

# Komponententypen: Micro Agenten

- Einfache Architektur
  - Lebenszyklus: init, run, terminate
  - Reaktivität: message arrived, wait for
- Implementation als einfache Java-Klassen (Plain Old Java Object - POJO) mit Annotationen
  - Für Agentenverhalten einfacher bis mittlerer Komplexität
  - Annotationen für spezielle Funktionalität: *@AgentMessageArrived*, *@AgentCreated*, *@AgentBody*, *@AgentKilled*
  - API für die Ausführung von Schritten: *scheduleStep(ICommand com)*, *waitFor(long time, ICommand com*

# Services: Programmierungsbeispiel: Shop




- Scenario
  - Shops have an inventory and offer items for certain prices
  - Customers can search stores and buy items in them

- System design
  - Shops define an interface IShop that allows customers to get the catalog of offered items and buy them
  - Customers search for IShop providers and use the interface to issue buy orders
  - The call is decoupled at interface level and executed asychronously in the callee

# Shop Service Interface Example

```
public interface IShop
{
    public String getName();

    public IFuture<Order> buyItem(String item, int quantity);

    public IFuture<Catalog> getCatalog();
}
```

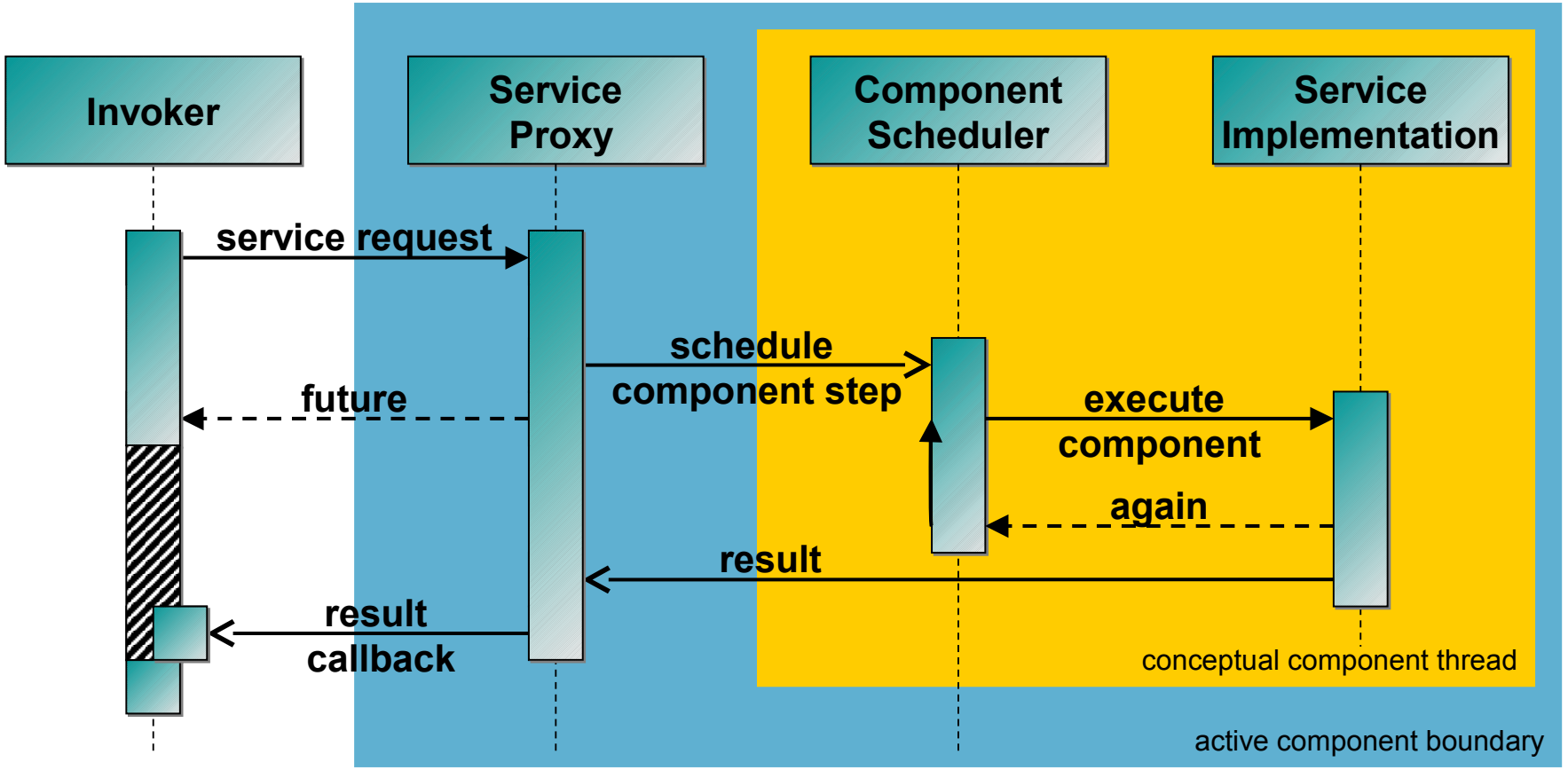
- Simple shop interface offers methods for getting the shop name, buying an item and getting the catalog
- getName() is allowed as it is considered as constant, i.e. the value will be cached
- IFuture represents a value that is immediately returned but may provide the result in future

# Future

- A future is a special means for encapsulating a return value of a method invocation that might be available in future
- Typically offers a *get()* method, which blocks the calling thread until the result is provided (*wait by necessity*)
- Adding a callback mechanism *addResultListener()* allows the caller being invoked when the result is available
- Using chains of futures and result listeners allows to avoid blocking threads

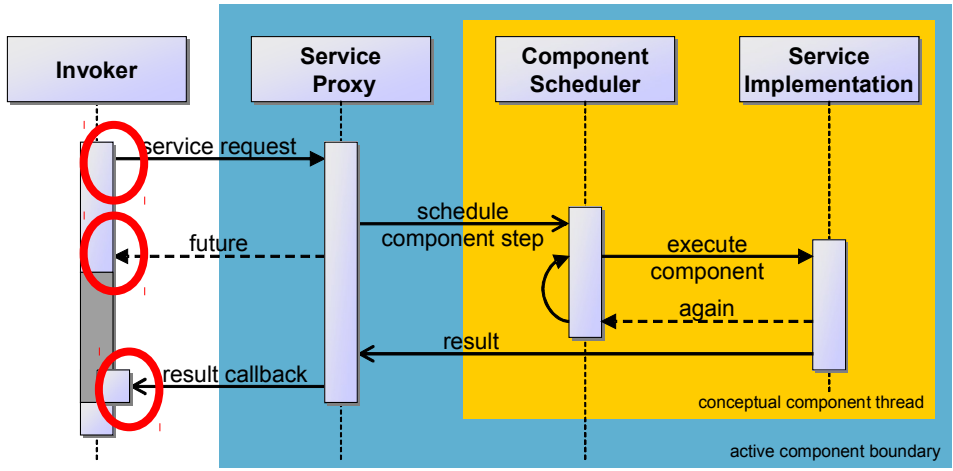
```
public interface IFuture<T> {
  public boolean isDone();
  public T get(ISuspendable caller);
  public T get(ISuspendable caller, long timeout);
  public void addResultListener(IResultListener<T> listener);
}

public interface IResultListener<T> {
  public void resultAvailable(T result);
  public void exceptionOccurred(Exception exception);
}
```

# Asynchrone Aufrufe: Prinzip



# Asynchrone Aufrufe: Aufrufer

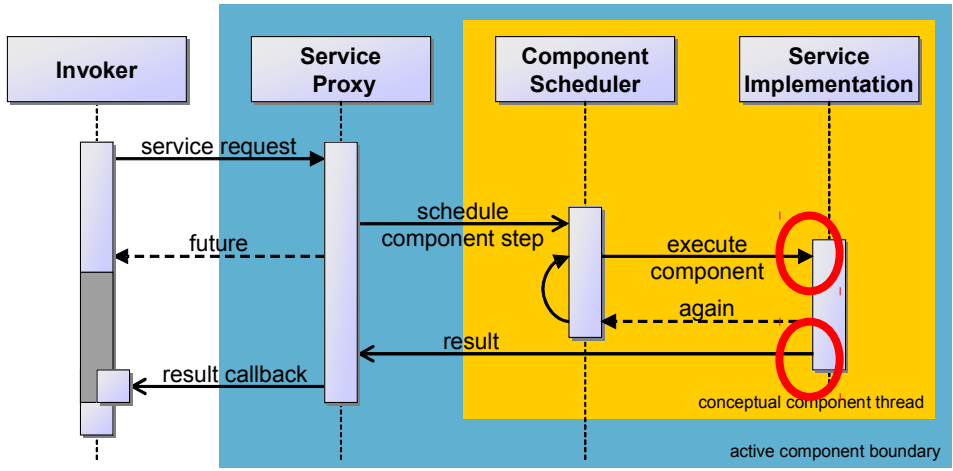
- Automatische Entkopplung
- Verarbeitung des Futures in einem IResultListener




```
IShopService shop = ...
IFuture<Catalog> fut = shop.getCatalog();
fut.addResultListener(new IResultListener()
{
  public void resultAvailable(Catalog result)
  {
    // Handle result
    display(result);
  }

  public void exceptionOccurred(Exception exception)
  {
    // Handle exception
    ...
  }
});
```

# Asynchrone Aufrufe: Implementation (1)

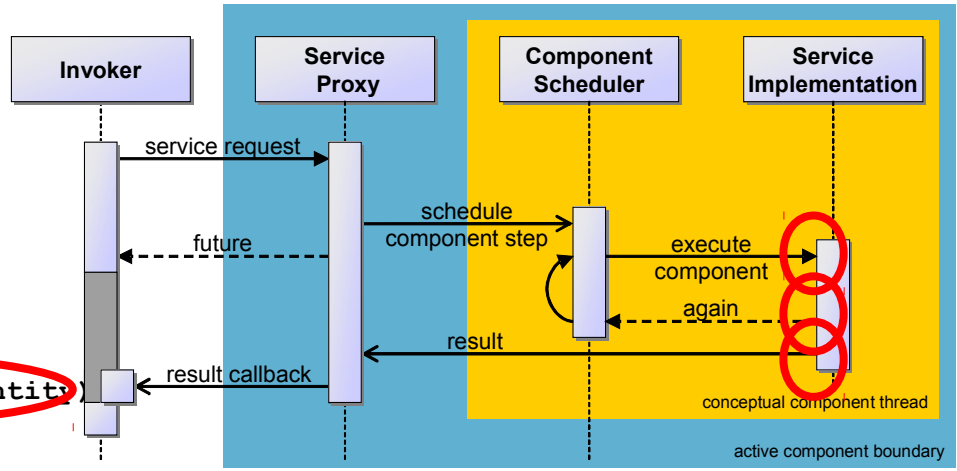
- Automatisch auf Komponenten-Thread ausgeführt
- Im einfachen Fall kann Ergebnis-Future direkt erzeugt werden.




```
public class MyShopService implements IShopService
{
  protected Catalog catalog;

  ...

  public IFuture<Catalog> getCatalog()
  {
    return new Future<Catalog>(this.catalog);
  }
}
```

# Asynchrone Aufrufe: Implementation (2)

- Verschachtelte Aufrufe über DelegationResultListener
- Automatisches Propagieren von Fehlern (exceptionOccurred)




```
public IFuture<Order> buyItem(String item, int quantity)
{
  final Future ret = new Future();
  stock.checkItem(item, quantity).addResultListener(new
    ExceptionDelegationResultListener<Boolean, Order>(ret)
  {
    public void customResultAvailable(Boolean result)
    {
      if(result.booleanValue())
      {
        Order order = new Order(...);
        ret.setResult(order);
      }
      else
      {
        ret.setException(new RuntimeException("Item not in stock"));
      }
    }
  });
  return ret;
}
```


**Fragen?**