

# Geschäftskomponenten mit EJBs

FWP Aktuelle Technologien zur  
Entwicklung verteilter Java-  
Anwendungen

# **GRUNDWISSEN ENTERPRISE JAVA BEANS**

# Überblick

- Standard-Komponentenmodell für server-seitige Java-Komponenten
- Ziel: Schnelle Entwicklung von verteilten, transaktionalen, sicheren und portablen Anwendungen

# Eigenschaften

- Verteilbarkeit über Remote Interfaces / RMI
- Deklarative und programmatische Transaktionalität
- Deklarative und programmatische Security
- Deklarative und programmatische Persistenz
- Portabilität/Herstellerunabhängigkeit innerhalb der Java EE-Plattform

# EJB Typen (I)

- **Stateless Session Beans**

- ⊙ Halten keinen Zustand über Methodenaufrufe hinweg
- ⊙ Keine Bindung eines Beans an einen bestimmten Client
- ⊙ Container bestimmt die Lebensdauer
- ⊙ Mehrere Instanzen in einem Instanzenpool

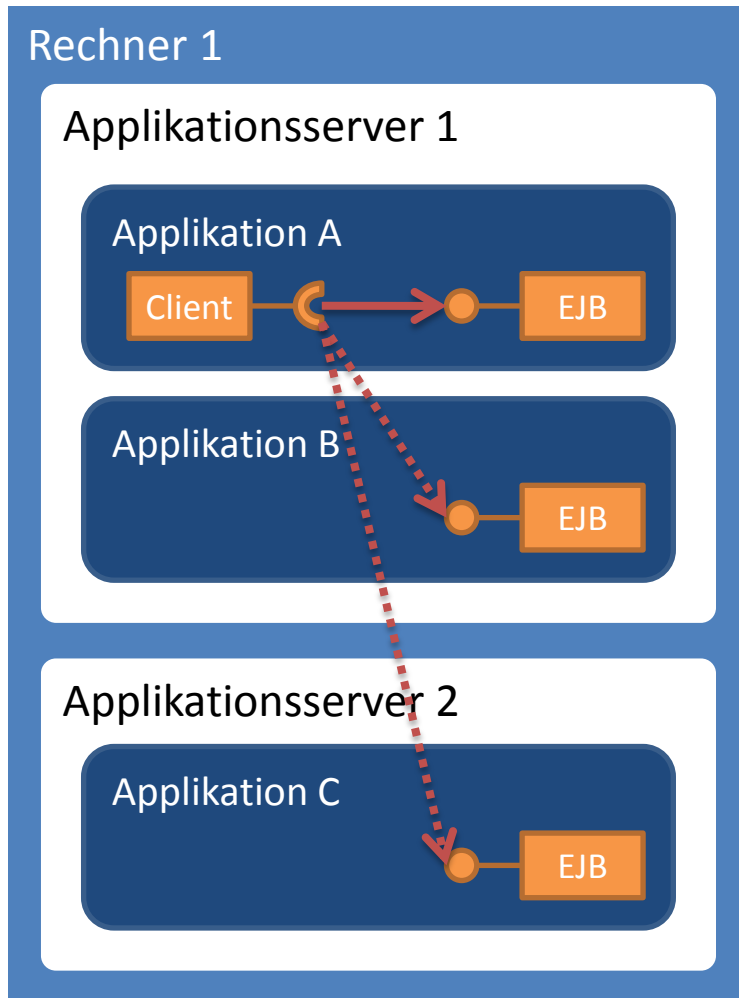
- **Stateful Session Beans**

- ⊙ Halten Zustand über Methodenaufrufe hinweg
- ⊙ Session Bean ist an bestimmten Client gebunden
- ⊙ Client bestimmt die Lebensdauer

# EJB Typen (II)

- **Singleton Session Bean**
  - ⊙ Sonderfall von Stateless Session Beans mit einer Instanz pro Applikation
- **Entities (Java Persistence Architecture)**
  - ⊙ Halten Zustand über Methodenaufrufe und Client-Sessions hinweg
  - ⊙ Objekt-orientierte Sicht auf (relationale) Daten
- **Message Driven Beans**
  - ⊙ Asynchrone Verarbeitung von Nachrichten

# Interfaces von Session Beans



- **Remote Interface**
  - ⊙ Methodenaufrufe erfolgen entfernt (d.h. von anderen Applikationen)
  - ⊙ Alle Parameter werden serialisiert und als Kopie übergeben
  - ⊙ Aufrufer und Session Bean können überall laufen
- **Local Interface**
  - ⊙ Methodenaufrufe erfolgen lokal (d.h. innerhalb der gleichen Anwendung)
  - ⊙ Alle Parameter werden als Referenz übergeben (sofern es sich um Referenztypen handelt)
  - ⊙ Aufrufer und Session Bean müssen sich innerhalb der gleichen Anwendung befinden
- **No-interface view**
  - ⊙ Session-Bean-Klasse ohne Interface

# Registrierung und Lookup

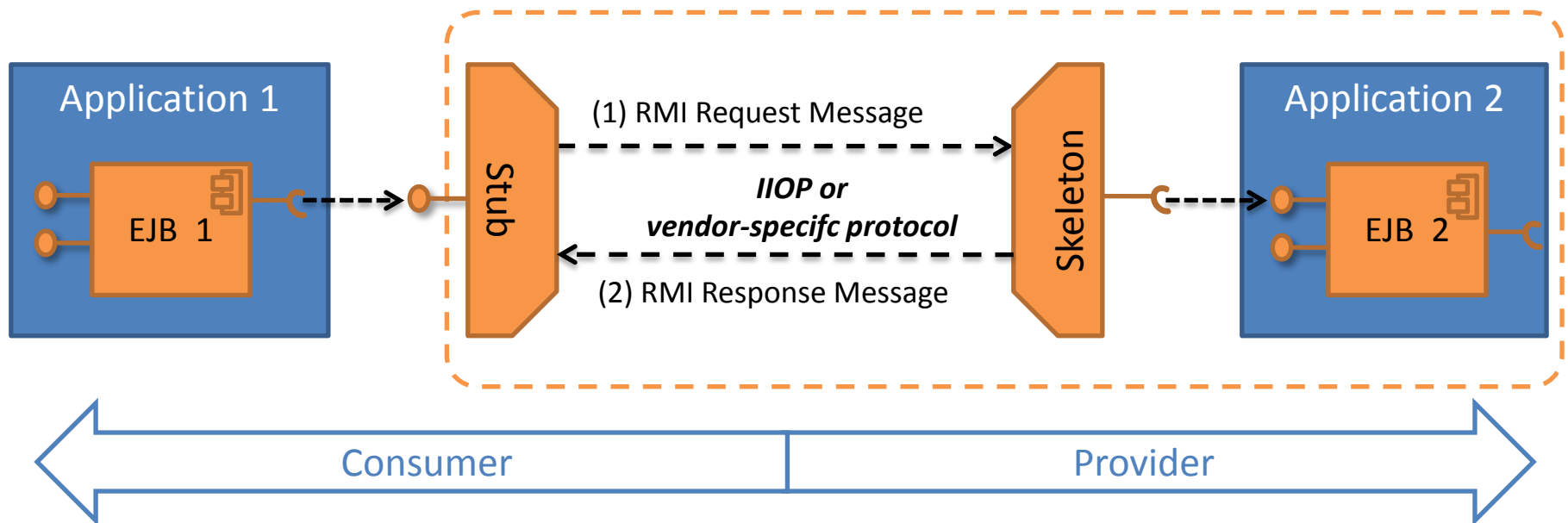
- Session Beans werden beim Deployment unter einem standardisierten Namen registriert
- Über den Namen kann eine Referenz auf ein Session Bean ermittelt werden (Lookup)
- Im Normalfall automatischer Lookup über Dependency Injection (EJB oder CDI)
- Manueller Lookup eines entfernten Session Beans ist der harte Teil

# Entfernte Aufrufe über RMI (I)

- Entfernte Aufrufe erfolgen über RMI (Remote Method Invocation)
- Alle Remote Interfaces der aufgerufenen Applikation müssen in ein EJB Client Modul gepackt werden
- Aufrufende Applikation muss dieses EJB Client Modul als Hilfsmodul hinzufügen

# Entfernte Aufrufe über RMI (II)

- Stubs (Consumer) und Skeletons (Provider) werden beim Deployment generiert



# Dependency Injection

- Abhängigkeiten zu anderen Ressourcen müssen nicht mehr von den EJBs selber aufgelöst werden
- Seit EJB 3.0 muss
  - nur noch ein Feld vom erwarteten Typ definiert
  - und dieses Feld annotiert werden
    - @EJB, @Resource, @PersistenceContext, @Inject
- Der EJB Container löst die Anhängigkeiten auf und injiziert die benötigten Referenzen in die markierten Felder

# Dependency Injection über @EJB

- UserRegistrationBean benötigt UserRepository:

```
@Stateless
public class UserRegistrationBean
implements UserRegistration {
    @EJB
    private UserRepository userRepository;
    public void registerUser(User newUser) {
        this.userRepository.addUser(newUser);
    }
}
```

```
@Stateless
@Local(UserRepository.class)
public class UserRespositoryBean
implements UserRepository { ... }
```



# Interzeptoren (I)

- **Interzeptoren** bringen AOP in die EJB-Welt
  - ⊙ Fangen Methodenaufrufe ab bevor sie das Ziel-EJB erreichen
  - ⊙ Ermöglichen das deklarative Hinzufügen von Querschnittsfunktionalität
- Mit **@Interceptors** wird eine Liste von Interzeptoren an ein EJB gebunden:

```
@Stateless
@Interceptors({ TraceInterceptor.class })
public class CourseManagementBean implements CourseManagement {
    ...
}
```

# Interzeptoren (II)

- Die Interzeptor-Klasse muss eine **Invocation-Handler-Methode** implementieren
- Diese muss mit **@AroundInvoke** annotiert werden

```
public class TraceInterceptor {  
    @AroundInvoke  
    public Object handleInvocation(InvocationContext invocation)  
        throws Exception {  
        // alle Aktionen vorm weiterleiten an das Ziel-EJB  
        invocation.proceed(); // weiterleiten an das Ziel-EJB  
        // alle Aktionen nach der Rückkehr vom Ziel-EJB  
    }  
}
```

# Deklarative Transaktionalität

- Transaktionales Verhalten eines EJBs wird bestimmt durch `@TransactionAttribute`

TransactionAttributeType	Beschreibung
MANDATORY	Erfordert eine aktive Transaktion beim Aufruf
REQUIRED (Default)	Erfordert eine aktive Transaktion beim Aufruf; startet eine neue Transaktion falls notwendig
REQUIRES_NEW	Legt beim Aufruf immer eine Transaktion an; erzwingt die Verwendung einer lokalen Transaktion
SUPPORTS	Es ist egal, ob eine aktive Transaktion vorliegt oder nicht
NOT_SUPPORTED	Unterstützt keine Transaktionen; aktive Transaktionen werden vorübergehend unterbrochen
NEVER	Es darf keine aktive Transaktion vorliegen

# Deklarative Security

- Zugriffsschutz beim Aufruf von EJBs wird über Annotations definiert

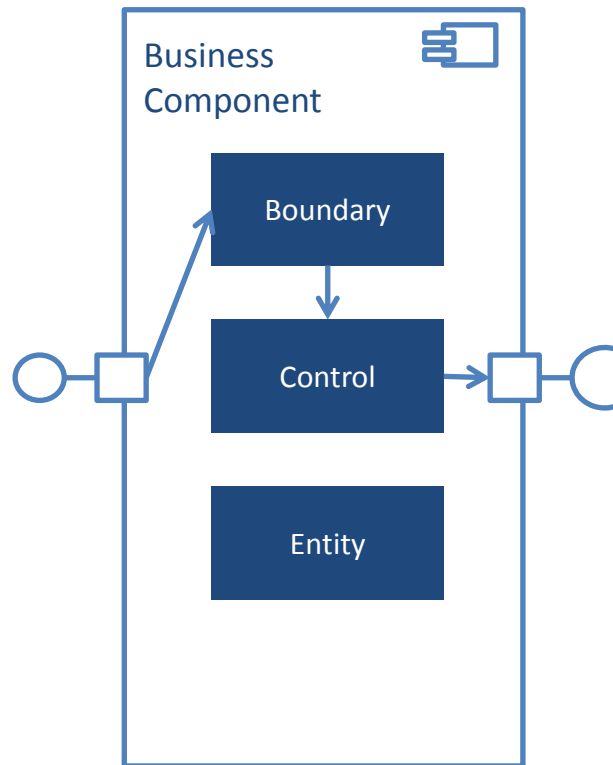
Wert	Beschreibung
@DeclareRoles	Definiert Rollen, gegen die beim Aufruf geprüft werden kann. Allerdings ist die Definition von Rollen im Deployment Deskriptor sinnvoller, da hier alle Rollen an einer Stelle definiert werden können.
@RolesAllowed	Definiert eine Liste von Rollen, die ein angemeldeter Benutzer haben muss, um Methoden eines EJBs aufzurufen
@DenyAll	Weist den Aufruf von Methoden eines EJBs für alle angemeldeten Benutzer ab
@PermitAll	Lässt den Aufruf für alle Benutzer zu, es wird keine Prüfung auf bestimmte Rollen vorgenommen
@RunAs	Führt eine Methode im Kontext einer bestimmten Rolle aus.

# Weitere Features von EJB 3.1

- Session-Bean-Methoden können asynchron aufgerufen werden (**@Asynchronous**)
- Session-Bean-Methoden können zeitgesteuert aufgerufen werden (**@Schedule**)
- Vereinfachte Verpackung in Web-Applikationen (**EJB Lite**)
- **Embedded EJB Container** ermöglicht Unit-Tests ohne Serverstart

# **GESCHÄFTSKOMPONENTEN MIT EJB**

# Einheitliches Komponentenmodell (Entity-Control-Boundary Pattern)



- **Boundary (Facade)**
  - ⊙ Bietet grob-granulare Geschäftslogik über Schnittstellen an
  - ⊙ Verbirgt die Komplexität der Geschäftslogik vor dem Aufrufer
  - ⊙ Benutzt Services zur Abwicklung der eigenen Funktionalität
- **Control (Service)**
  - ⊙ Bietet fein-granulare, wiederverwendbare Geschäftslogik über Schnittstellen an
  - ⊙ Benutzt Resource Access Components für den Zugriff auf externe Ressourcen
- **Entity (Domain Object)**
  - ⊙ Repräsentiert das für Aufrufer sichtbare Domänenmodell
  - ⊙ Simple Datenträger (SOA) oder „fette“ Objekte (DDD)

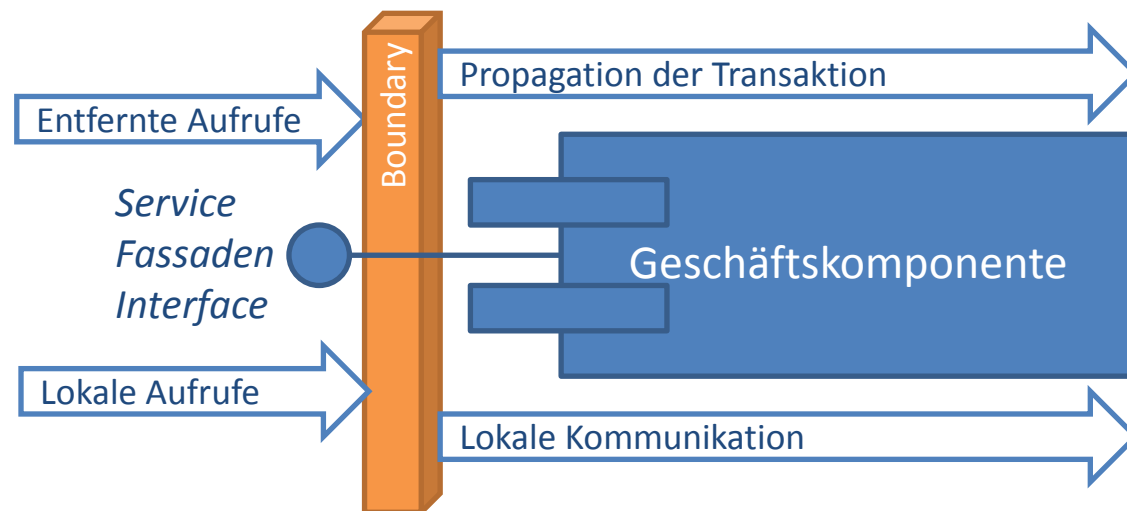
# Boundaries (Service-Fassade)

- Stellen grob-granulare Interfaces für spezifische Geschäftslogik zur Verfügung
- Verbergen deren Komplexität vor dem Aufrufer
- Orchestrieren meist Services zur Umsetzung der Geschäftslogik, können diese aber auch selber implementieren
- Bereitstellung als Webservices möglich

***Facade:** Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. [GOF1995]*

# Wächter am Tor zur Logik

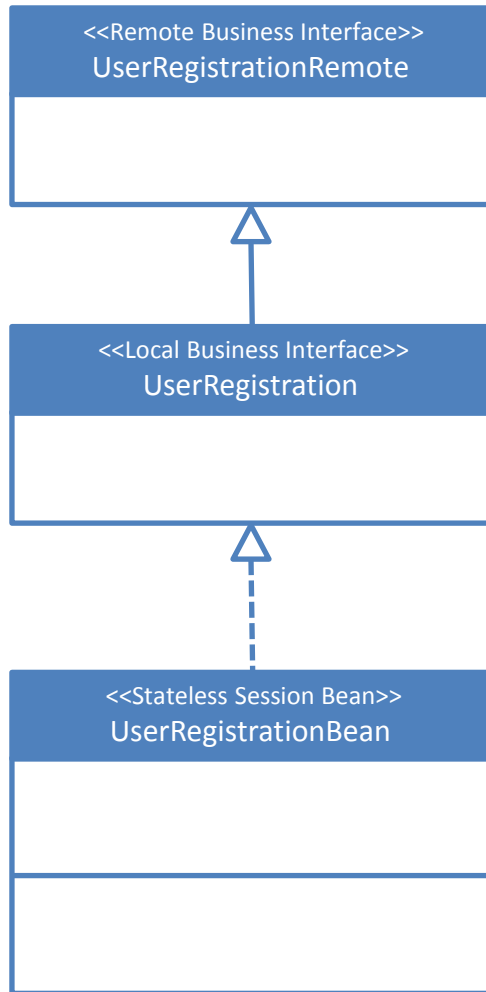
- Boundaries agieren als Torwächter, die sicherstellen
  - ⊙ dass Geschäftslogik immer konsistent ausgeführt wird
  - ⊙ dass nur Benutzer mit den erforderlichen Zugriffsrechten Zugang erhalten



# Eigenschaften von Boundaries

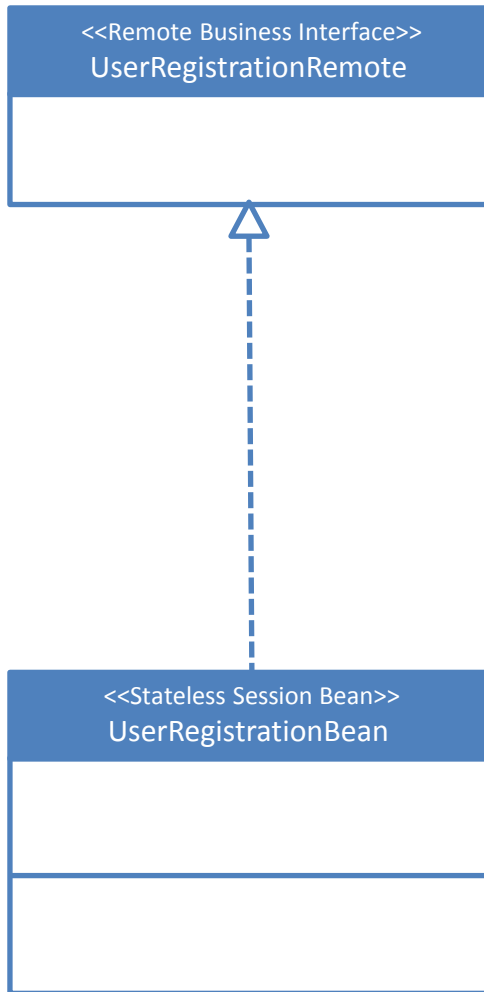
- Einziger lokaler oder entfernter Zugang zur Geschäftslogik
- Alle Methoden einer Boundary
  - ⊙ sollten eine aktive Transaktion voraussetzen oder besser eine neue erzeugen (**REQUIRES\_NEW**)
  - ⊙ sollten für entfernte Aufrufe designt werden auch wenn man sie nur lokal nutzt (**dual view**)
  - ⊙ sollten immer einen authentisierten Benutzer verlangen (**@RolesAllowed**)

# Duale Sicht auf Boundaries (I)



- Boundaries bieten lokale und entfernte Interfaces an (**dual-view**)
- Im Sinne der Einfachheit
  - ⊙ erweitert das lokale Interface das entfernte Interface
  - ⊙ implementiert die Session Bean-Klasse das lokale Interface
- Im Sinne der Reinheit der Interfaces erfolgt die Definition der Interfaces in der Implementierungsklasse

# Duale Sicht auf Boundaries (II)



- Boundary bietet entferntes Interface an
- Lokales Interface ist die Implementierungsklasse selber (*no-view*)
- Optimierte Version der dualen Sicht
- Falls kein entfernter Zugriff erforderlich, kann auch das Remote Interface entfallen

# Control (Services)

- Implementieren fein-granulare Geschäftslogik
  - ⦿ die nur innerhalb der Geschäftslogikschicht sichtbar ist
  - ⦿ die leicht zu komplexeren Abläufen von Geschäftslogik orchestriert werden kann
- Realisiert als zustandslose POJOs oder Stateless Session Beans
- Services arbeiten auf dem applikationsspezifischen Domänenmodell

# Entity (Domänen-Objekte)

- Repräsentieren das Domänen-Modell
  - ⊙ gegenüber der Benutzerschnittstelle der Anwendung
  - ⊙ gegenüber entfernten Aufrufern der Anwendung
- Meist simple Daten-Transfer-Objekte (DTO)
  - ⊙ Eingabeparameter und Rückgabewerte in Methoden der Service-Fassaden-Interfaces
  - ⊙ Reine Datenträger (fast) ohne Logik  
(nur einfache Validierungsregeln und Zugriffskontrolle auf Feldebene)

# Typen von Domänen-Objekten

- **Entitäten (Entities)**: persistente Geschäftsdaten mit einer spezifische Identität
- **Wertobjekte (Value Objects)**: transiente Geschäftsdaten, nicht an eine Identität gebunden
- **Aggregate (Aggregate)**: Cluster aus Entitäten und Wertobjekten umgeben von einer transaktionalen Grenze

# Design von Geschäftskomponenten

- Boundaries und Entities bilden gemeinsam die öffentliche API der Geschäftslogikschicht
- Sie tragen daher wesentlich zum Erfolg (oder Misserfolg) einer Applikation bei
- Es lohnt sich daher, ein wenig über deren Design nachzudenken!

# Kriterien für gute Geschäftskomponenten

- Keine äußerlichen Abhängigkeiten zum Domänenmodell aus anderen Anwendungen
- Kompatibilität gegenüber älteren Versionen muss eingehalten werden
- Leicht verständlich und einfach zu benutzen durch
  - ⊙ vollständige Dokumentation
  - ⊙ sprechende Namen (*intention-revealing names*)
  - ⊙ klare Verträge (*design by contract*)
  - ⊙ eindeutige Informationen im Fehlerfall

# Fragen?



# ANHANG

# Quellen

- Eric Jendrock et. al.: ***The Java EE 7 Tutorial Part VII Enterprise Beans***  
<http://docs.oracle.com/javaee/7/tutorial/doc/partentbeans.htm>  
Oracle September 2013
- Adam Bien: ***Real World Java EE Patterns: Rethinking Best Practices***  
press.adam-bien.com September 2012; ISBN 978-0-300-14931-6
- Eric Evans: ***Domain Driven Design:  
Tackling Complexity in the Heart of Software***  
Addison Wesley 2004; ISBN 0-321-12521-5



# Kontakt



## **Michael Theis**

Lehrbeauftragter Hochschule München

email        michael.theis@hm.edu

mobile      + 49 170 5403805

web         <http://www.tschutschu.de/Lehrauftrag.html>