Saxonia Systems

A vision of technology.

Spring: Das neue J2EE?

Eberhard Wolff
Saxonia Systems AG
Eberhard.Wolff@saxsys.de

Saxonia Systems

A vision of technology.

Über mich

- Eberhard Wolff
  Chef Architekt
  Saxonia Systems
- Fokus: Enterprise Java
- Saxonia Systems:
  Breit aufgestelltes Systemhaus
- Zentrale Dresden
- Niederlassungen in
  Hamburg, Frankfurt, München

Einfachheit ist das Resultat der Reife.

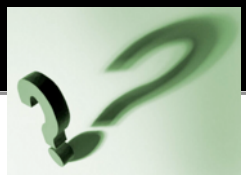
Friedrich von Schiller

Saxonia Systems
A vision of technology.

Die Leichtigkeit kann nicht ohne Anstrengung erreicht werden.

Wassily Kandinsky

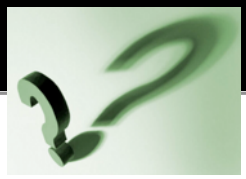
Saxonia Systems
A vision of technology.



Warum Spring?

- Einfachheit
- Fokus auf POJOs (Plain Old Java Objects)
- Vereinfachte Benutzung verschiedener APIs
  - Hibernate, JDO, Top Link, iBATIS, OJB, JMS, JDBC, JTA, Java Mail, EJB, Quartz, JMX ...
- Dependency Injection
  - Objekte bekommen Ressourcen zugewiesen
  - Objekte sind unabhängig von Infrastruktur

Saxonia Systems
A vision of technology.



Warum Spring?

- Keine Zwänge
  - Man muss Spring nicht komplett nutzen
  - Viele Integrationsmöglichkeiten
  - Kann aber One Stop Shop sein
- Code weitgehend unabhängig von Spring
- Spring - Das neue J2EE?

Übersicht

- Dependency Injection
- Aspekt-orientierte Programmierung mit Spring
- Meta-Framework Ansatz am Beispiel JDBC
- Verteilte Objekte mit Spring
- Ausblick

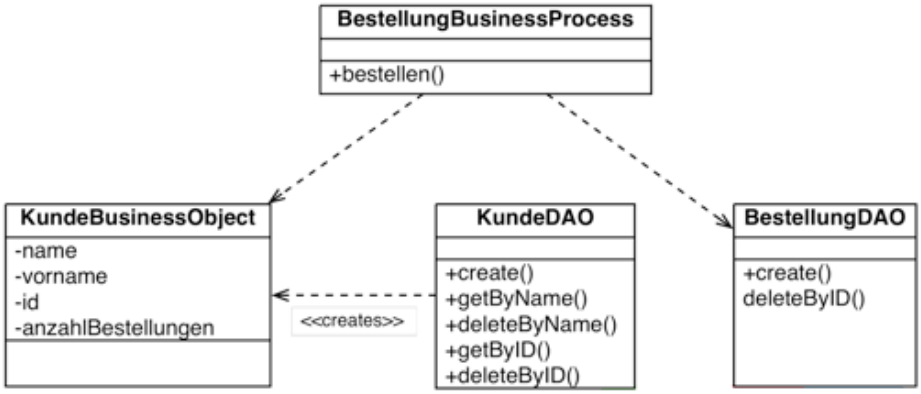
Saxonia Systems
A vision of technology.

Dependency Injection

Saxonia Systems
A vision of technology.

Das Beispiel

- Kunden
- Bestellungen
- Alles, was ein Unternehmen braucht!



Saxonia Systems
A vision of technology.

Umsetzung der Klassen

- „Normale“ Java Klassen
- Keine Abhängigkeiten zu Spring
- Referenzen auf andere Objekte durch set-Methoden zuweisbar
- Konfiguration in Spring Konfigurationsdatei

Saxonia Systems

A vision of technology.

Spring Konfigurationsdatei

```
<beans>

<bean id="kundeDAO" class="dao.KundeDAOJDBC">
  <property name="datasource"><ref bean="datasource"/></property>
</bean>

<bean id="bestellungDAO" class="dao.BestellungDAOJDBC">
  <property name="datasource"><ref bean="datasource"/></property>
</bean>

<bean id="bestellung" class="businessprocess.BestellungBusinessProcess">
  <property name="bestellungDAO"><ref bean="bestellungDAO"/></property>
  <property name="kundeDAO"><ref bean="kundeDAO"/></property>
</bean>

</beans>
```

Saxonia Systems

A vision of technology.

Benutzung...

- Vorsicht! Nur Tests oder Main Methode!
- ...denn nur die „Top Level Objekte“ müssen so zugegriffen werden
- Abhängige Objekte sind ja konfiguriert

```
ClassPathResource res =
  new ClassPathResource("beans.xml");
XmlBeanFactory beanFactory = new XmlBeanFactory(res);
BestellungBusinessProcess bestellung;
bestellung = (BestellungBusinessProcess)
  beanFactory.getBean("bestellung");
bestellung....;
```

Saxonia Systems
A vision of technology.

My Name is...

- Dependency Injection
  - Abhängige Objekte werden „injiziert“
- Auch: Inversion of Control
  - Meine Meinung: Weniger passend
  - Jedes Framework kehrt die Kontrolle um
- Spring ist nicht die einzige DI Implementierung!
  - Avalon, Pico Container ...
- Spring ist nicht nur DI!

Saxonia Systems
A vision of technology.

DI: Vorteile

- Klassen automatisch unabhängig von Umgebung
  - Oft Abhängigkeiten zu JNDI, Factory, Service Locator, ...
- Flexibilität
  - *DataSource* aus J2EE JNDI
  - ...oder bei J2SE aus Jakarta Commons Projekt
- Testbarkeit
  - Mocks oder besondere Konfiguration für Tests
- Objekte sind per default Singletons
  - Singletons != globale Variablen
  - Alternative zu statischen Methoden

„Man könnte sich ohrfeigen, dass man es nicht immer so gemacht hat!“

Weitere DI Themen

- Autowiring
- Andere Erzeugungsmöglichkeiten
  - Konstruktor Parameter
  - Statische Factory Methode
  - Instanz Factory Methode
- Factory Beans: Factories mit Spring Unterstützung
- Method Injection: Eine Methode wird durch Bean Erzeugung überschrieben
- Integration von .property Dateien in XML Dateien

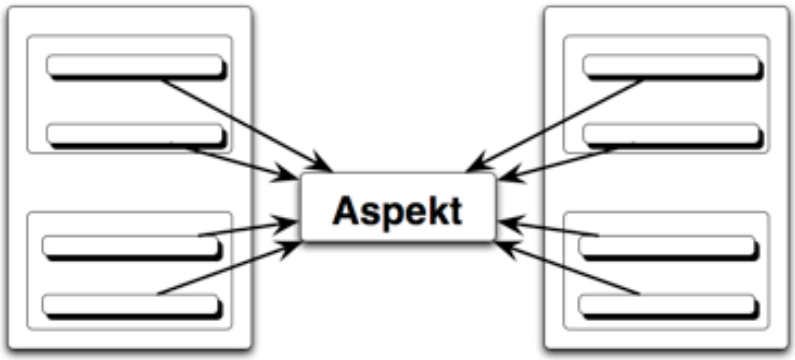
Saxonia Systems
A vision of technology.

Fazit Dependency Injection

Σ

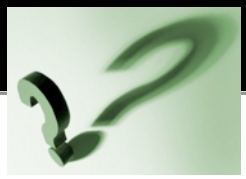
- DI definiert Referenzen zwischen Objekten
  - ...damit neben Vererbung
- Eingebaute Konfiguration
- Leichtere Testbarkeit
  - Statt echter Klassen Mocks zuweisen
- Sehr einfach
- Flexibel: Ressourcen aus dem Application Server, J2SE, ...

Aspekt-orientierte Programmierung



Saxonia Systems
A vision of technology.

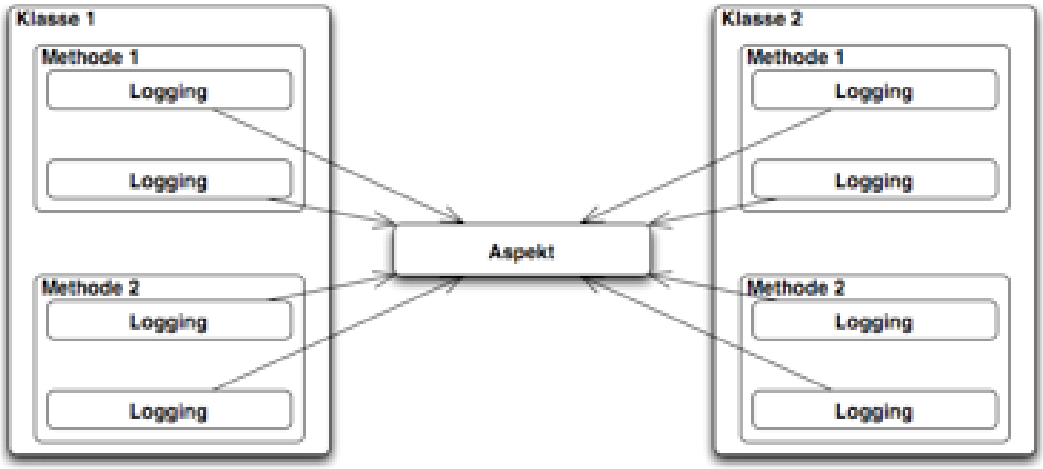
Was ist AOP?



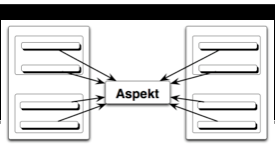
- Aspekt-orientierte Programmierung
- Cross Cutting Concerns (Querschnittsbelange) an einer Stelle zentralisieren
  - Transaktionen
  - Sicherheit
- „Hallo Welt“: Logging
  - Normalerweise im Code verstreut
  - An einer Stelle zentralisieren

Saxonia Systems
A vision of technology.

Logging: AOP „Hallo Welt“

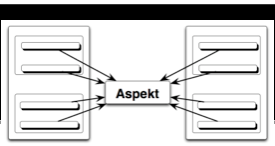


Saxonia Systems
A vision of technology.



Advice

- Was soll ausgeführt werden?
- Bei Spring 4 Typen
- MethodInterceptor
  - Bekommt alle Daten des Methodenaufrufs
  - Beliebige Funktionalität
  - muss nicht „echte“ Methode aufrufen (Cache)
- BeforeAdvices
  - vor Abarbeiten der Methode
- AfterReturningAdvices
  - nach der Ausführung der Methode
  - Können Rückgabewerte nicht modifizieren
- ThrowsAdvices
  - falls Exception fliegen

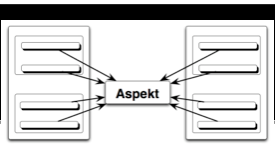


Beispiel für einen Advice

```
public class DebugInterceptor
  implements MethodInterceptor {

    public Object invoke(MethodInvocation invocation)
        throws Throwable {
          ... // log
          try {
                Object rval = invocation.proceed();
                 ... // log
                return rval;
          } catch (Throwable ex) {
                 ... // log
                throw ex;
          }
    }

}
```

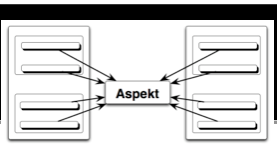



Wie ist AOP in Spring implementiert?

- ➜ Dynamic Proxies
- ➜ Erzeugen eine Implementierung eines Interfaces...
- ➜ ...bei der die Methodenaufrufe an einen InvocationHandler weitergeleitet werden

```
interface InvocationHandler {
  Object invoke(Object proxy,
                Method method,
                Object[] args);
}
```

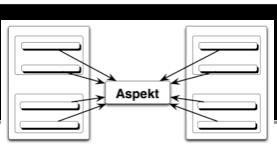
- ➜ Also: Dynamic Proxy führt Advices und dann „echte“ Methode aus



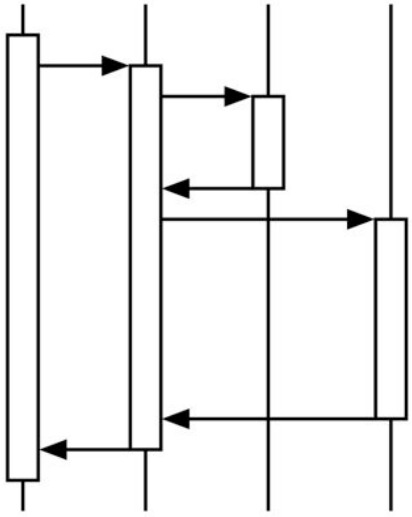
AOP bei Beans ohne Interface

- CGLIB
- Gleiche Schnittstelle wie Dynamic Proxies, aber Lösung durch Subklassen
- Subklasse überschreiben Methoden so, dass *InvocationHandler* ausgeführt werden
- Also *final* Klassen ausgenommen

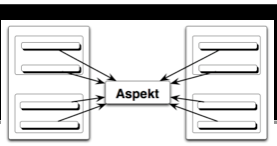
Saxonia Systems
A vision of technology.



Spring AOP zur Laufzeit



Saxonia Systems
A vision of technology.



Spring Bean um Advice erweitern

```
<bean id="debugInterceptor"
  class=" ….DebugInterceptor"/>

<bean id="debugProxyTemplate" abstract="true"
  class=" ….ProxyFactoryBean">
  <property name="proxyTargetClass">
    <value>true</value></property>
  <property name="interceptorNames">
    <list><value>debugInterceptor</value></list>
  </property>
</bean>

<bean id="kundeDAOTarget" class="dao.KundeDAOJDBC">…
<bean id="kundeDAO" parent="debugProxyTemplate">
 <property name="target"><ref local="kundeDAOTarget"/>
</property></bean>
```

Advice

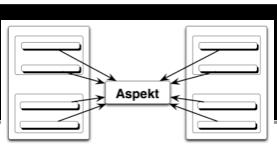
Schablone zum Einbauen der Advices

CGLIB

Eigentlicher Bean

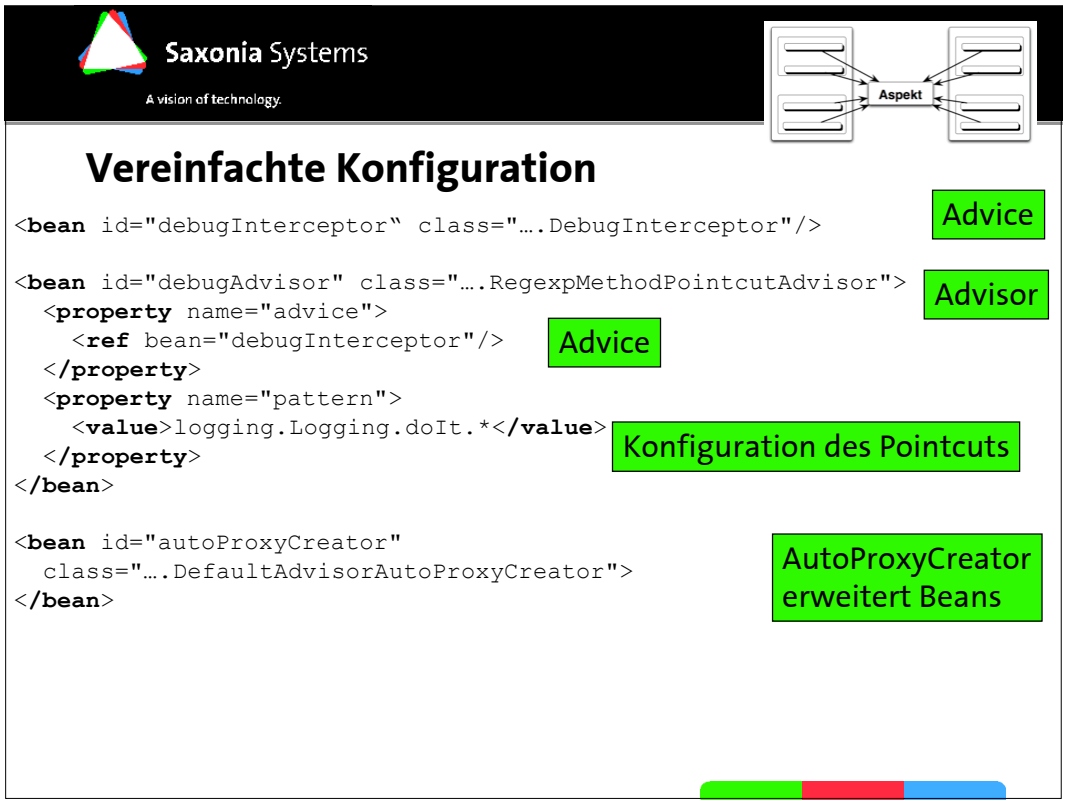
Einbauen der Advices

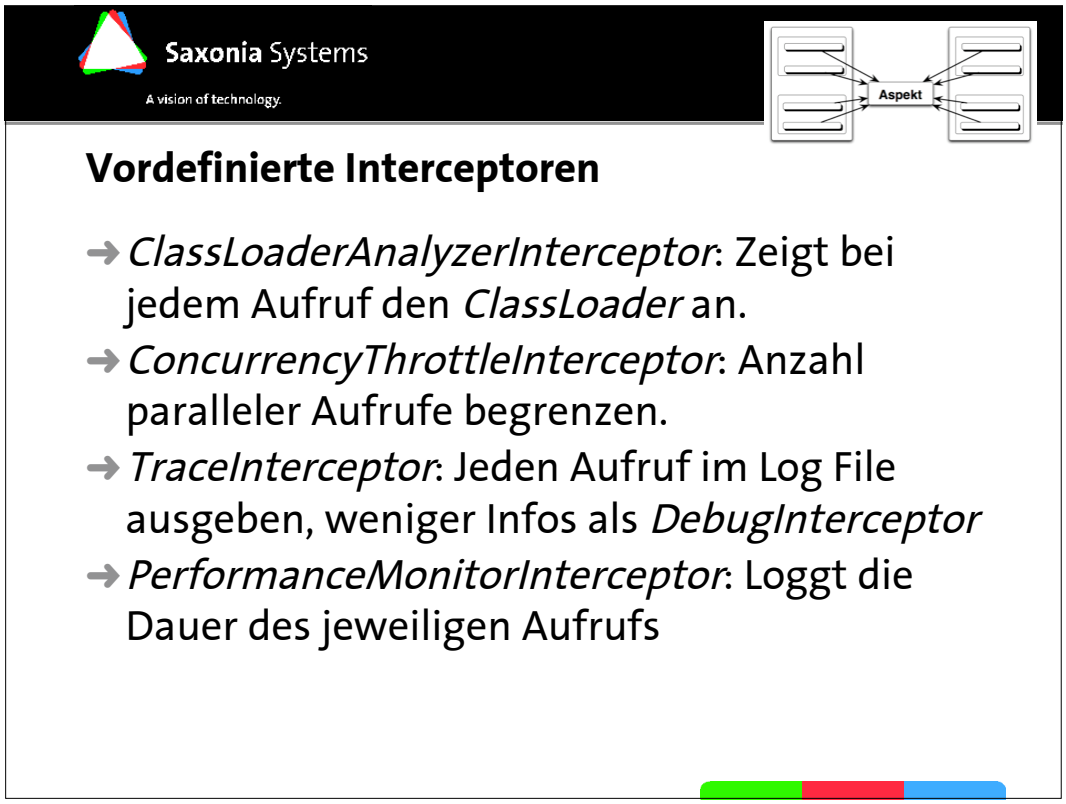
Saxonia Systems
A vision of technology.

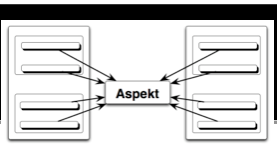


Einfachere Konfiguration: Pointcuts

- Lösung: Pointcuts
  - Wann soll ein Advice aktiv werden?
- In Spring eine Klasse mit *MethodMatcher* und *ClassFilter*
- Vordefiniert: Nach Methodenname, Union…
- Advisor=Advice+Pointcut
- *ApplicationContext* statt *BeanFactory* + *DefaultAutoProxyCreator* wenden Advisor automatisch an


Saxonia Systems
A vision of technology.
Aspekt
Vereinfachte Konfiguration
<bean id="debugInterceptor" class="….DebugInterceptor"/>
<bean id="debugAdvisor" class="….RegexpMethodPointcutAdvisor">
<property name="advice">
<ref bean="debugInterceptor"/>
</property>
<property name="pattern">
<value>logging.Logging.doIt.*</value>
</property>
</bean>
<bean id="autoProxyCreator"
class="….DefaultAdvisorAutoProxyCreator">
</bean>
Advice
Advisor
Advice
Konfiguration des Pointcuts
AutoProxyCreator
erweitert Beans

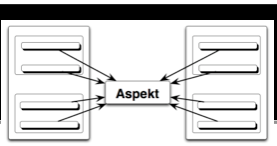

Saxonia Systems
A vision of technology.
Aspekt
Vordefinierte Interceptoren
➜ ClassLoaderAnalyzerInterceptor: Zeigt bei jedem Aufruf den ClassLoader an.
➜ ConcurrencyThrottleInterceptor: Anzahl paralleler Aufrufe begrenzen.
➜ TraceInterceptor: Jeden Aufruf im Log File ausgeben, weniger Infos als DebugInterceptor
➜ PerformanceMonitorInterceptor: Loggt die Dauer des jeweiligen Aufrufs



Synergie AOP und Dependency Injection

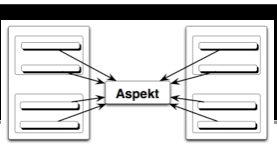
- Durch DI gibt es eine Stelle, an der Advice „eingemischt“ werden können
  - *BeanFactory* bzw. *ApplicationContext*
  - Allerdings: Aspekte können nur an Beans ansetzen
  - Aber: Aspekte wären sonst zu feingranular (?)
- Außerdem: DI eignet sich zum Konfigurieren von Aspekten

Saxonia Systems
A vision of technology.



AOP .NET Style

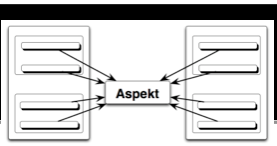
- .NET kennt Attributes zum Markieren von Methoden oder Klassen
- In JDK 1.5 enthalten (JSR 175)
- XDoclet geht auch in die Richtung
- Spring: Jakarta Commons Attributes oder JSR 175
- Attribute sind sinnvoll, um Methoden für Aspekte zu markieren
  - Z.B. Transaktionen
  - Siehe auch EJB



Alternative: AspectJ

- Der Klassiker unter den AOP Implementierungen
- Mit Spring nutzbar
- Spring zwingt eigene Lösung nicht auf
- Hat Syntax Erweiterungen zu Java
- Sehr mächtig
- Lässt Spring AOP trivial aussehen
- ...aber für einfache Aspekte überdimensioniert

Saxonia Systems
A vision of technology.



AOP Zusammenfassung

Σ

- AOP ist in Spring durch DI möglich
- AOP löst z.B. Logging
- ...aber auch Transaktionen
- ...weitere Möglichkeit: Exception Behandlung (z.B. Logging)
- Mit Attributen interessantes, .NET ähnliches Programmiermodell
- Man kann auch AspectJ in Spring integrieren

Meta-Framework Ansatz am Beispiel JDBC

Hilfsklassen: JDBC

- ➜ Problem bei JDBC: Aufräumen der Connection vor allem bei Exceptions
- ➜ *SQLException*
  - Gibt proprietäre Codes zurück
  - Checked Exception (keine *RuntimeException*)
  - Aber: kaum sinnvolle Reaktionsmöglichkeiten
  - Kann in Enterprise Anwendung fast überall auftauchen
    - Wie *NullPointerException*

Saxonia Systems

A vision of technology.

Lösung Allgemein: Template

- Auszuführenden Code einem Template übergeben
- Template führt Code aus
  - wandelt Exceptions in *RuntimeExceptions* um
  - Gemeinsame Exception Hierachie für alle Datenbanken
  - Stellt Ressourcen bereit und räumt sie auf
- Templates sind in Spring für viele Frameworks implementiert
  - Hibernate, iBATIS, JDO, OJB, Top Link, JMS...
  - „Meta-Framework“

Saxonia Systems

A vision of technology.

JDBC Lösung: JDBCTemplate

- Für Queries: *queryForInt()* ...für primitive Daten
- *queryForList()* für komplexe *ResultSets*: Gibt *List* von *Maps* mit Spaltenname als Schlüssel zurück

```
JdbcTemplate insertTemplate = new JdbcTemplate(
  datasource);
insertTemplate.update(
  "INSERT INTO KUNDE(VORNAME,NAME) VALUES(?,?)",
  new Object[] {vorname, name });
```

Saxonia Systems

A vision of technology.

JDBC Komplexere Queries

- Man kann einer *query()* Methoden auch übergeben:
  - *RowMapper*: wandelt Zeilen in Objekte
  - *ResultSetExtractor*: gesamtes Ergebnis in Objekt wandeln
  - Ebenfalls Möglichkeiten für komplexe Ergebnisse

Saxonia Systems

A vision of technology.

Hilfsklassen; Eigene Query Klassen

- Können z.B. zum Ablegen von Queries in Instanzvariablen genutzt werden
- Dadurch weniger Overhead

```
class BestellungInsert extends SqlUpdate {

 public BestellungInsert(DataSource ds) {
  super(ds,
   "INSERT INTO BESTELLUNG(ID_KUNDE,BETRAG) VALUES(?,?)",
    new int[] { Types.INTEGER, Types.INTEGER });
  compile();
 }
}
```

Meta-Framework Ansatz am Beispiel JDBC

- ➜ Macht Benutzung von JDBC deutlich einfacherer und sicherer
- ➜ Isoliert vom Rest von Spring nutzbar
- ➜ Beispiel für allgemeinen Template Mechanismus
- ➜ Nicht gezeigt:
  - Integration iBATIS, Hibernate, JDO, Toplink, JMS
  - Ähnliche Prinzipien (Templates)
  - Wesentliche Vereinfachungen (z.B. Hibernate Session)

Saxonia Systems
A vision of technology.

Verteilte Objekte mit Spring

Saxonia Systems
A vision of technology.

Verteilte Objekte mit Spring

- Spring fokusiert auf POJOs (Plain Old Java Objects)
- Wie mache ich einen POJO über‘s Netz zugreifbar?

Saxonia Systems
A vision of technology.

RMI

Erste Technologie: RMI

- RMI erwartet...
  - ...eigenes Remote Interface
  - Jede Methode darf eine *RemoteException* werfen
  - Implementierung des Remote Interface
    - Delegiert an POJO
- Lösung: Adapter Pattern
  - GoF Pattern
  - Interface einer Klasse in Interface der Clients umwandeln
  - In Spring automatisiert

Saxonia Systems

A vision of technology.

RMI

RMI Server Konfiguration

```
<bean class="....RmiServiceExporter">
  <property name="serviceName">
    <value>kundeDAO</value>
  </property>
  <property name="service">
    <ref bean="kundeDAO"/>
  </property>
  <property name="serviceInterface">
    <value>dao.IKundeDAO</value>
  </property>
  <property name="registryHost">
    <value>localhost</value>
  </property>
</bean>
```

Saxonia Systems

A vision of technology.

RMI

Hinter den Kulissen...

- ➜ *RmiInvocationHandler* wird verwendet
- ➜ Leitet Aufrufe dynamisch an POJO weiter
- ➜ Server muss nur *ApplicationContext* instanziieren
- ➜ Dadurch wird *ServiceExporter* instanziiert
- ➜ ...und der Service läuft!

Client: Benutzung des Servers

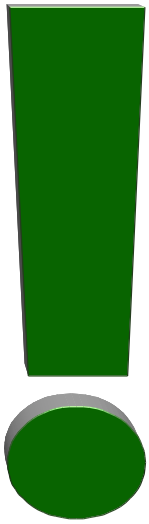
- ➜ Wie geht man mit der *RemoteException* um?
  - Technisches Problem
  - Umwandlung in *RuntimeException* scheint sinnvoll
  - Interface kompatibel zu POJO Interface
- ➜ Mit Spring transparent durch *RmiProxyFactoryBean*
  - *FactoryBeans* erzeugen Beans

Saxonia Systems
A vision of technology.
RMI

RMI Client Konfiguration

```
<bean id="kundeDAO" ....rmi.RmiProxyFactoryBean">
  <property name="serviceUrl">
    <value>rmi://localhost:1099/kundeDAO</value>
  </property>
  <property name="serviceInterface">
    <value>dao.IKundeDAO</value>
  </property>
</bean>
```

Nur, um es nochmal zu sagen:



- Verteilte Kommunikation ist *vollständig* transparent!

Hessian
Burlap

Hessian und Burlap

- HTTP basierte Protokolle
- Open Source Implementierung
- Einfacher als SOAP
- Einfache Serialisierung von Objekten
- Keine Interface Sprache wie WSDL
- Hessian: Binäres Protokoll
- Burlap: XML basiertes Protokoll

Hessian Burlap

Web Konfiguration für Spring

- *ContextLoaderServlet* bzw. *ContextLoaderListener*
  - Lädt Beans aus *WEB-INF/applicationContext.xml*
- *DispatcherServlet*
  - Verteilt Aufrufe an Beans
  - Eigene Konfiguration unter *WEB-INF/<servlet-name>-servlet.xml*
- Ziel: Modularisierung der Konfiguration

Hessian Burlap

Hessian Exporter Konfiguration

```
<bean name="/kundeDAOHessian"
  class="....HessianServiceExporter">

  <property name="service">
    <ref bean="kundeDAO"/>
  </property>
  <property name="serviceInterface">
    <value>dao.IKundeDAO</value>
  </property>

</bean>
```

Das „/“ zeigt an, dass der Bean Name die URL angibt.

Saxonia Systems

A vision of technology.

Hessian
Burlap

HTTP Protokolle

- Für Burlap in der Konfiguration „Hessian“ durch „Burlap“ ersetzen
- Auf Client Seite *FactoryBean* wie bei RMI
- Spring bietet außerdem ein proprietäres, auf Java Serialisierung basierendes Protkoll

Saxonia Systems

A vision of technology.

SOAP

SOAP

- Mit JAX-RPC / Axis
- Axis Servlet konfigurieren
  - Spring Konfiguration durch *ContextLoaderServlet*
  - Eigene Konfiguration (server-config.wsdd)
- Konvertierung zwischen SOAP und Java Typen
  - Auf Client Seite: Implementierung des *JaxRpcServicePostProcessor*
  - Auf Server Seite: Deklaration im server-config.wsdd
- Manuelle Implementierung des Adapters
  - Erweitert *ServletEndpointSupport*
  - Delegiert jede Methode an POJO

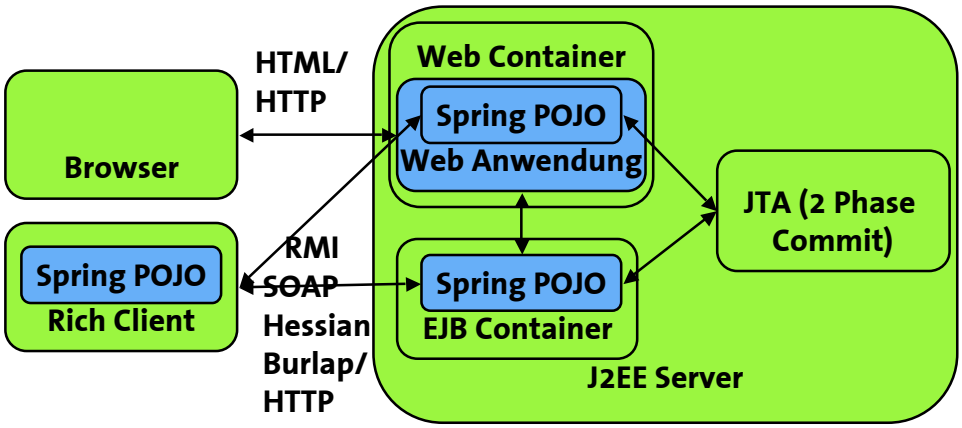
Verwendung eines SOAP Objekts

- Wieder eine *ProxyFactoryBean*: *JaxRpcPortProxyFactoryBean*
- Konfiguration anhand von WSDL
  - URL des WSDL Dokuments
  - *serviceName* im WSDL Dokument
  - *portName* des Service
  - servicePostProcessor für den *JaxRpcServicePostProcessor*
  - ...und das Java Interface

EJB mit Spring

- Adapter manuell schreiben (wie bei SOAP)
- Mit XDoclet Konfiguration erzeugen lassen
  - Spezielle Templates notwendig: http://www.jochnet.de/html/spring_ejb.shtml
- Adapter für
  - Stateless Session Beans
  - Stateful Session Beans
  - Message Driven Beans

Also...



Saxonia Systems
A vision of technology.

Was bedeutet das alles?

- Infrastruktur ist völlig transparent!
  - Entweder Konfiguration oder zusätzliche Adapter
  - Infrastruktur Entscheidung kann spät getroffen werden
  - Infrastruktur kann gewechselt werden
  - Tests ohne Infrastruktur
  - Fokus auf Logik statt Infrastruktur
- Szenario: Web Anwendung benötigt Rich Clients
  - Web Services? EJB? RMI?
- Oder: Anwendung soll auf Laptops und Servern lauffähig sein

Saxonia Systems

A vision of technology.

Ausblick

Saxonia Systems

A vision of technology.

Spring MVC

- ➜ Eigenes Web Framework
- ➜ Vorteil: Formulardaten in POJOs speichern
- ➜ Flexible View Technologien
  - JSP/JSTL
  - Tiles
  - Velocity/Freemaker
  - XSLT
  - PDF/Excel
  - JasperReports
- ➜ Einfach und komplett konfigurierbar

Integration anderer Web Frameworks

- Struts
- Java Server Faces (JSF)
- Tapestry
- WebWork
- Logik in Spring, Oberfläche flexibel

Saxonia Systems
A vision of technology.

Acegi
SECURITY
SYSTEM
FOR SPRING

Sicherheit mit Acegi

- Basiert auf Spring AOP
- Damit fachlicher Code frei von Sicherheitsbelangen
- Single Sign On mit Central Authentication Service (Yale)
- Instant-basierte Access Control Lists

Saxonia Systems

A vision of technology.

Spring Rich Client

- Unterstützung für GUI Entwicklung
- Einige Hilfen
- Validierung
- Unterstützung für Commands
- Keine Möglichkeit für dynamische Updates von Teilen

Saxonia Systems

A vision of technology.

Spring Webflow

- Im Mittelpunkt steht der Ablauf der Seiten
- Flows können wieder verwendet werden
- Integriert sich in Struts oder Spring MVC
- Gut für komplexe Abläufe

Fazit

- DI kann Objektreferenzen ausdrücken
- DI Vorteile: Flexibilität, Konfigurierbarkeit, Vereinfachung, OO Fokus
- AOP als Basis Enterprise-Technologie integriert
- Transaktionen und Sicherheit (Acegi) als Aspekte implementiert
- Infrastruktur Flexibilität und Unabhängigkeit
- Eigenes, elegantes Web Framework
- Zahlreiche Integrationsmöglichkeiten (Meta-Framework)
- Weitergehende Frameworks / Ergänzungen entstehen