andrena OBJECTS

Experts in agile software engineering

Modularisierung und Inversion of Control mit Spring done right

Fabian Knittel, David Burkhart

andrena objects ag

Entwicklertag Karlsruhe, 21. Mai 2014

andrena OBJECTS

Modularisierung

https://www.flickr.com/photos/juhansonin/4734829999/

andrena OBJECTS

Modularisierung

Modularisierung - Vorteile

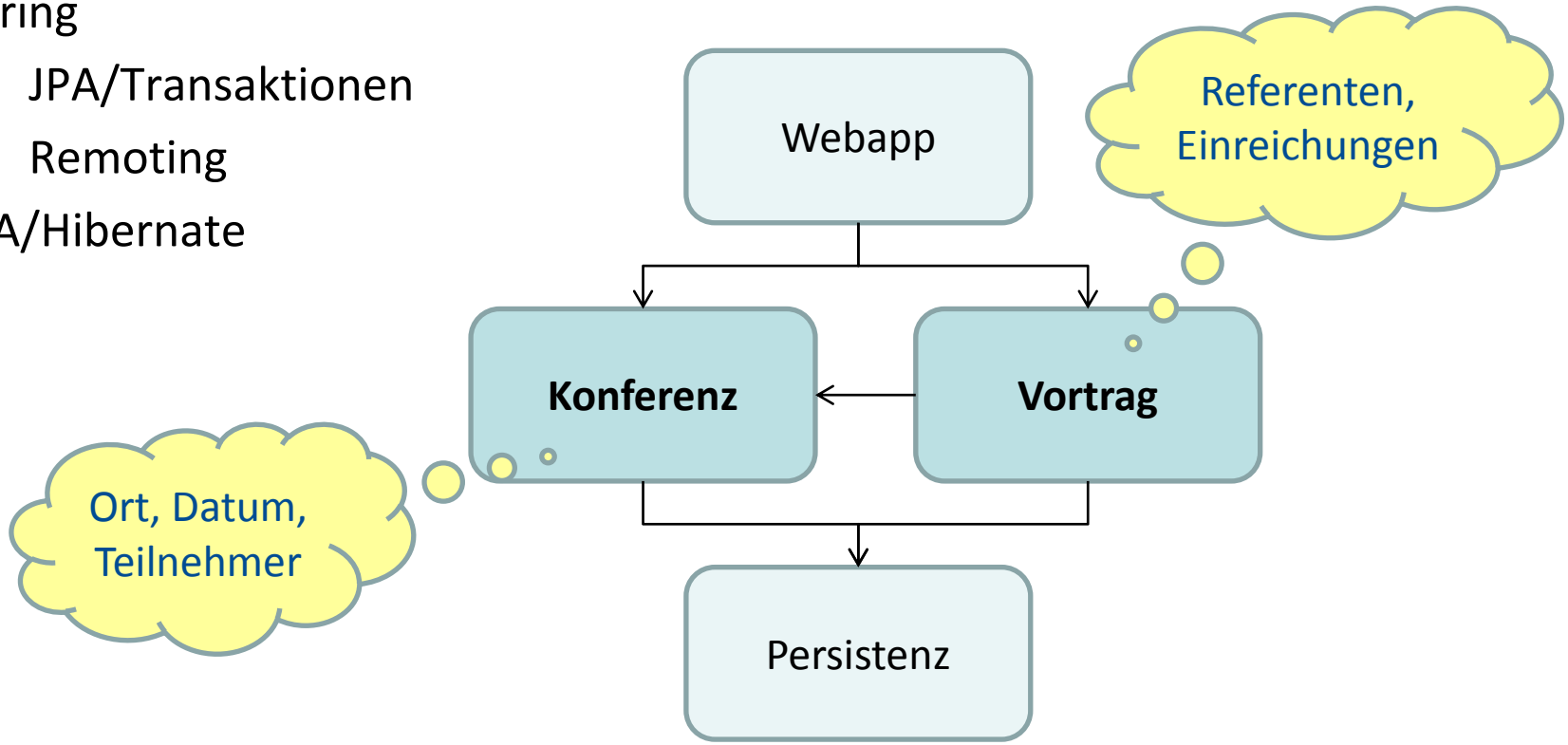
- ✓ Wiederverwendung von Teilen
- ✓ Änderungen bleiben lokal
- ✓ Unabhängiges Arbeiten an verschiedenen Stellen möglich
- ✓ Kleinerer Workspace
- ✓ Kleinere Runtimes möglich
- ✓ Frühere Releases wichtiger Teile möglich
- ✓ Keine/weniger Architekturchecks notwendig

Demo-Projekt

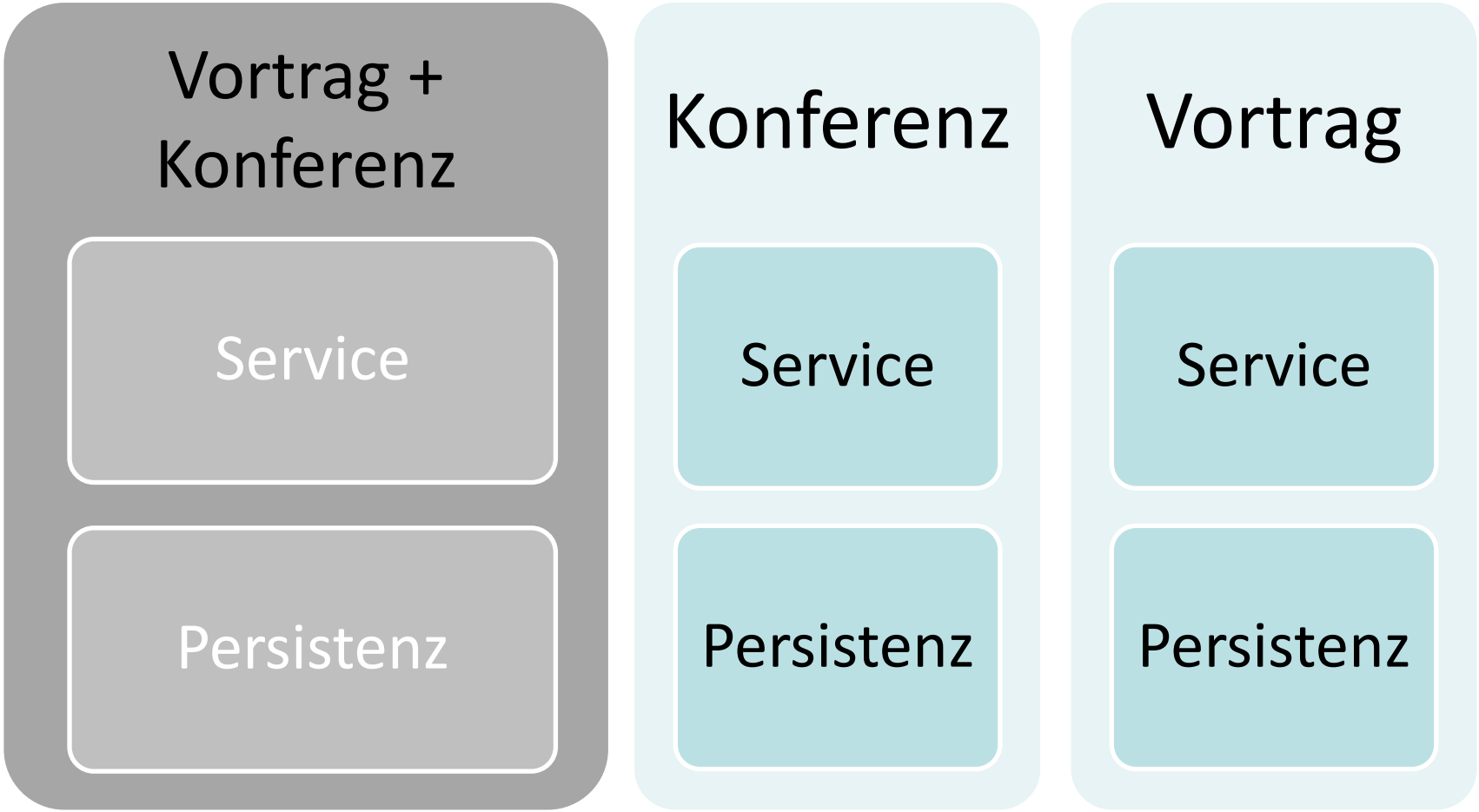
Technologien

- Spring
  - JPA/Transaktionen
  - Remoting
- JPA/Hibernate

Fachliche Domäne + techn. Module



Technische vs. Fachliche Modularisierung



IoC mit JPA

Problem

- `persistence.xml` enthält Klassennamen aller Module
  - Liegt im Webapp- oder Persistenz-Projekt

Nachteile

- Keine echte Modularisierung
- Anpassungen im Modul oder neue Module erfordern Anpassungen in zentraler Datei
- Persistenz-JPA-Tests liegen evt. im Webapp-Projekt

IoC mit JPA Lösung a)

`persistence.xml` durch Classpathscan weglassen:

- `ClasspathScanningPersistenceUnitPostProcessor` (aus spring-data-jpa)
- oder ab Spring 3.1: `packagesToScan`-Property an `LocalContainerEntityManagerFactoryBean`

IoC mit JPA Lösung b)

`persistence.xml` modularisieren und durch Java-Code ablösen

- Eigener `PersistenceUnitPostProcessor` scannt Classpath nach Implementierungen von `PersistentClassesProvider`
- Auflistung der Klassen in jedem Modul durch Implementierung von `PersistentClassesProvider`

IoC mit JPA Fazit

Vorteile beider Lösungen

- Keine zentrale Datei mit Klassennamen aus allen Modulen
- JPA-Tests verwenden nur die Klassen, die im Abhängigkeitsgraph des entsprechenden Moduls liegen

Vorteile a gegen b

- Lösung b: Verwendung mehrerer PersistenceUnits möglich
- Referenzensuche

Remoting (HTTPInvoker, …)

Problem

- Zentrale Servlet-Kontext-Konfiguration referenziert andere Konfigurationen oder Beans daraus

Einfache Lösung

- Andere Konfigurationen per ‚*‘ importieren

Remoting – elegante Lösung

- HTTPInvoker-Exporter selbst registrieren
- Services registrieren sich selbst oder werden per Marker (Interface oder Annotation) gefunden und exportiert

Remoting – Vorteile

- Keine Referenzen von Webapp auf Module
- Kein XML
- Keine Wiederholung von Konfigurationen
  - Aspekte, wie Logging / Security zentral einbindbar
- Einfaches Exportieren von Remoting-Schnittstellen per Interface oder Annotation
  - Exportierte Schnittstellen per Java-Referenzen auffindbar
- Neues Modul erfordert keine Änderung an zentralen Stellen

Zwischenfazit: Spring = Modularisierung (?)

Problematisch: Zentrale XML-Datei(en) mit Abhängigkeiten auf Module

- `applicationContext.xml`
  - Import anderer xml-Konfigurationen
  - Referenzen auf Klassen aus Modulen
  - Referenzen auf Beans von Modulen
  - Endpoints aus Modulen (WebServices, HTTPInvoker, …)
  - Quartz-Trigger aus Modulen
- `persistence.xml`

Java-basierte Konfiguration

Probleme

- Zur Compile-Zeit unauffällige Abhängigkeiten in XML-Files
- Refactorings im Code benötigen spezielle Spring-Plugins

Lösung

- Konfigurieren via Java-Code
- Seit Spring 3.1 komplett ohne XML

Einfacher Plugin-Mechanismus

- Statt direkter Verdrahtung und damit womöglich einhergehenden ungewollten Abhängigkeiten

- Möglichkeit 1: Autowiring
  - `@Autowire` / `@Inject` funktioniert wunderbar für Listen
  - Liste von Beans die ein Interface implementieren wird injected

- Möglichkeit 2: `InitializingBean` + `applicationContext.getBeansOfType()`

- Möglichkeit 3: Bean „Plugins“ wird injected und man registriert sich explizit

Zwischenfazit 2: Spring = Modularisierung (?)

Problematisch: Module referenzieren sich gegenseitig

- Compile-Abhängigkeit
- Entgegengesetzte Laufzeit-Abhängigkeit durch zu Compile-Zeit unauffälliger XML-Referenzen

Bonus:

- IDE-Unterstützung für Refactoring ohne Spring-Plugins

Fazit / Zusammenfassung

- (Fachliche) Modularisierung ist wichtig
- Spring kann Modularisierung unterstützen
  - man muss aber selbst darauf achten, um bestimmte Fehler zu vermeiden
- Spring ist sehr leicht erweiterbar und anpassbar
- Code-Beispiele:

  https://github.com/andrena/spring-modules-done-right