Implementacja architektury usługowej - od specifikacji do kodu

Paweł Cesar Sanjuan Szklarz

Warszawa Java User Group

5 czerwca 2012

`DepencencyInjection implements InverseOfControl<new>`

(IoC) Odwrócenie sterowania dla `new`:

odbieramy sterowanie
W kodzie nie występuje słowo kluczowe `new`

przejmujemy sterowanie
Tworzenie instancji zgodnie z wydzieloną konfiguracją

Injection points

```
public class DependencyInjectionPoints implements
    FeatureSpecification {
@Inject
private InjectionPointField field;
@Inject
@Named("custom key")
private InjectionPointField fieldWithAnnotatedKey;
private final InjectionPointConstructor fromConstructor;

@Inject
public DependencyInjectionPoints(InjectionPointConstructor
     constArgument) {
super();
this.fromConstructor = constArgument;
}
}
```

Model "explicit" dla Injektowania zależności

Klucz

`Key<T>` = `Type<T>` × `Annotation`

Injection Point

Położenie o kodzie →`Key<T>`

Dostawca

```
public interface Provider<T> { T get(); }
```

BindingProvider (`Injector`)

```
public interface BindingProvider {
  <T> Provider<T> getProvider(Key<T> key);
}
```

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`
2. `Key<T>` $\longrightarrow$ `Provider<T>`

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`
2. `Key<T>` $\longrightarrow$ `Provider<T>`
3. `Provider<T>` $\longrightarrow$ Injection Point

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`
2. `Key<T>` $\longrightarrow$ `Provider<T>`
3. `Provider<T>` $\longrightarrow$ Injection Point
4. Injection Point $\longrightarrow$ `Key<T>`

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`
2. `Key<T>` $\longrightarrow$ `Provider<T>`
3. `Provider<T>` $\longrightarrow$ Injection Point
4. Injection Point $\longrightarrow$ `Key<T>`
5. Powtarzamy ....

Algorithm tworzenia instancji

1. `Injector.createInstance(Key<T>)`
2. `Key<T>` $\longrightarrow$ `Provider<T>`
3. `Provider<T>` $\longrightarrow$ Injection Point
4. Injection Point $\longrightarrow$ `Key<T>`
5. Powtarzamy ....
6. Powstaje graf zależności przy tworzeniu instancji danego `Key<T>`:
   `Key<T>` $\longrightarrow$ `Set<Provider<T>>`

Uporządkowanie konfiguracji w modułach

Moduł Guice to konfiguracja kilku kluczy w jednej klasie

Listing 1: Module

```
package doc.wjug;
import com.google.inject.AbstractModule;
public class ModuleExample extends AbstractModule {
    @Override
    protected void configure() {
        bind(FeatureSpecification.class).to(
            DependencyInjectionPoints.class);
        bind(InjectionPointField.class).to(
            InjectionPointClass.class);
    }
}
```

Zakres modułu

Jeden moduł określa "bindowanie" dla zestawu kluczy:

```
Modul = { Key<T> → Provider<T> }
```

Do tworzenia "Injector" potrzebujemy zestaw modułów tak, aby graf zależności był dobrze zdefiniowany:

```
Injector injector = Guice.createInjector(Module...
    modules);
```

Budowanie definicji architektury

Usługa to kontrakt zdefiniowany przez zbiór interfejsów.

Budowanie definicji architektury

Usługa to kontrakt zdefiniowany przez zbiór interfejsów.
Architektura jest to struktura uporządkowująca zbiór usług.

Specykacja usługi Zbiór intejfejsów

Kategoria Rozłączne zbiory usług

Architektura Zbiór wszystkich kategorii.

Budowanie definicji architektury

Usługa to kontrakt zdefiniowany przez zbiór interfejsów.
Architektura jest to struktura uporządkowująca zbiór usług.

Specykacja usługi Zbiór intejfejsów

Kategoria Rozłączne zbiory usług

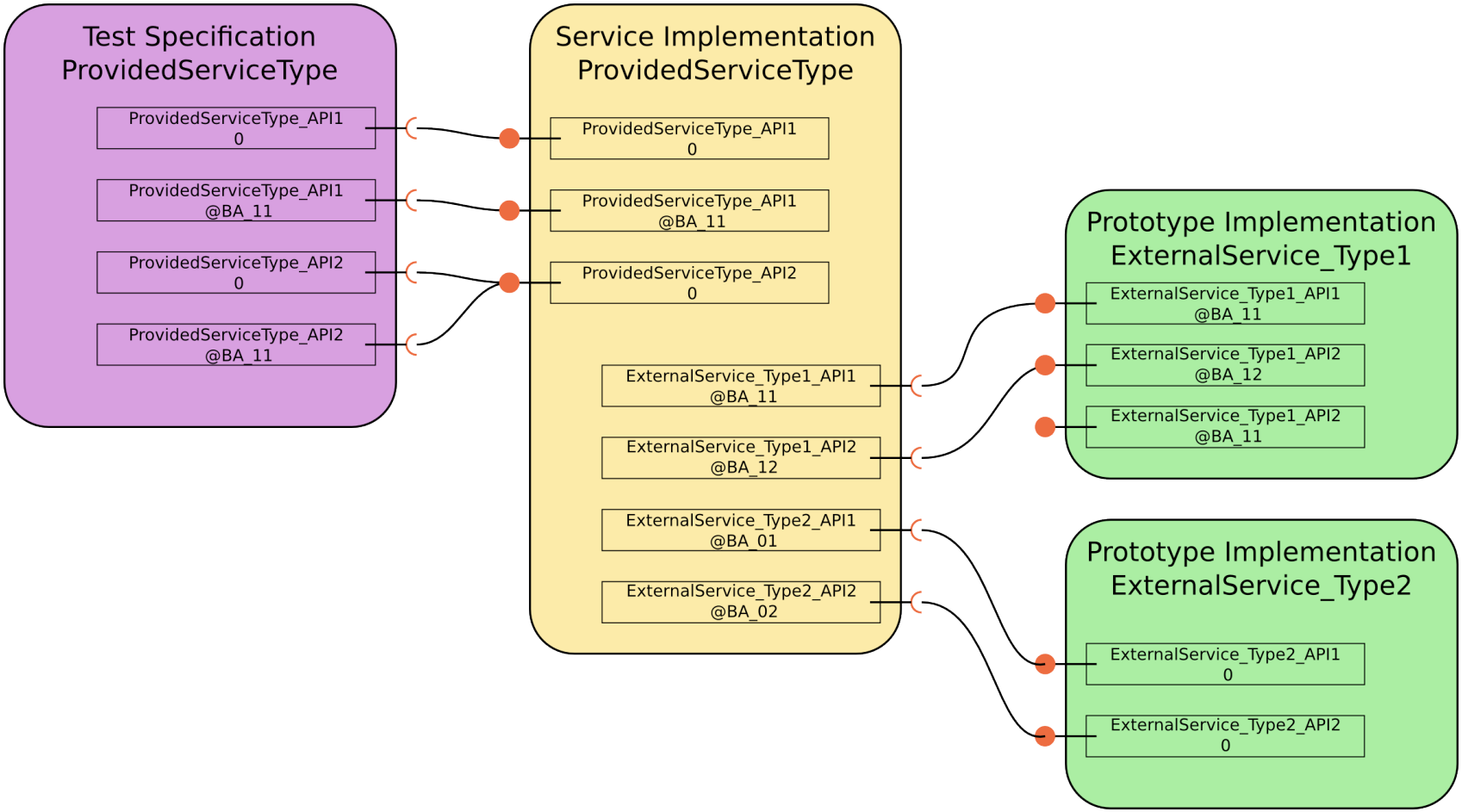
Architektura Zbiór wszystkich kategorii.

Definiujemy ograniczenia dla implementacji usług

Implementacja Bindowanie interfejsów usługi (moduł Guice)

Relacja dostępności kategorii Odpowiada na pytanie:
Jakie `Key<T>` mogę injektować w implementacji usługi?

Specyfikacja-Kontrakt



Implementacje usług oraz injektowanie instancji usług

Standardowe narzędzia DI nie odpowiadają architekturze usługowej.
Problemy:

Standardowe narzędzia DI nie odpowiadają architekturze usługowej.
Problemy:

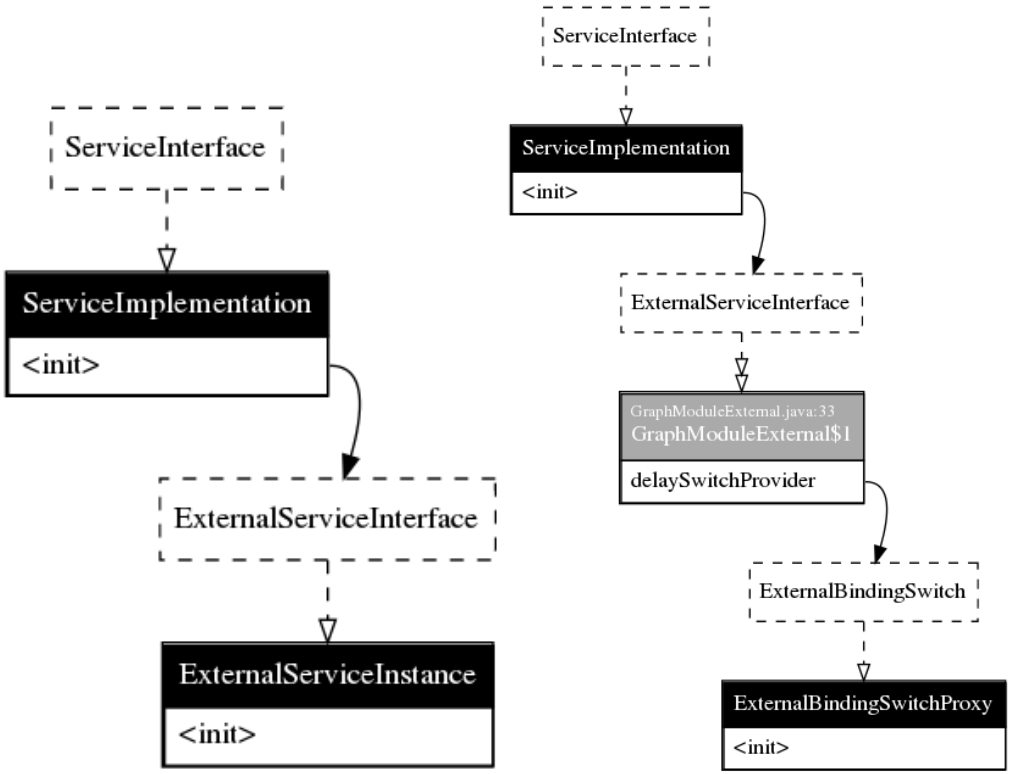
1. Moduł/Komponent to *NIE* instancji usługi:
    1. Sztywne połączenie
    2. Instancie to singletony

Standardowe narzędzia DI nie odpowiadają architekturze usługowej.
Problemy:

1. Moduł/Komponent to *NIE* instancji usługi:
   1. Sztywne połączenie
   2. Instancie to singletony
2. Kontekst uruchomienia
   1. Ograniczenie do jednej JVM
   2. ograniczenia dostępu
   3. "własność" usługi

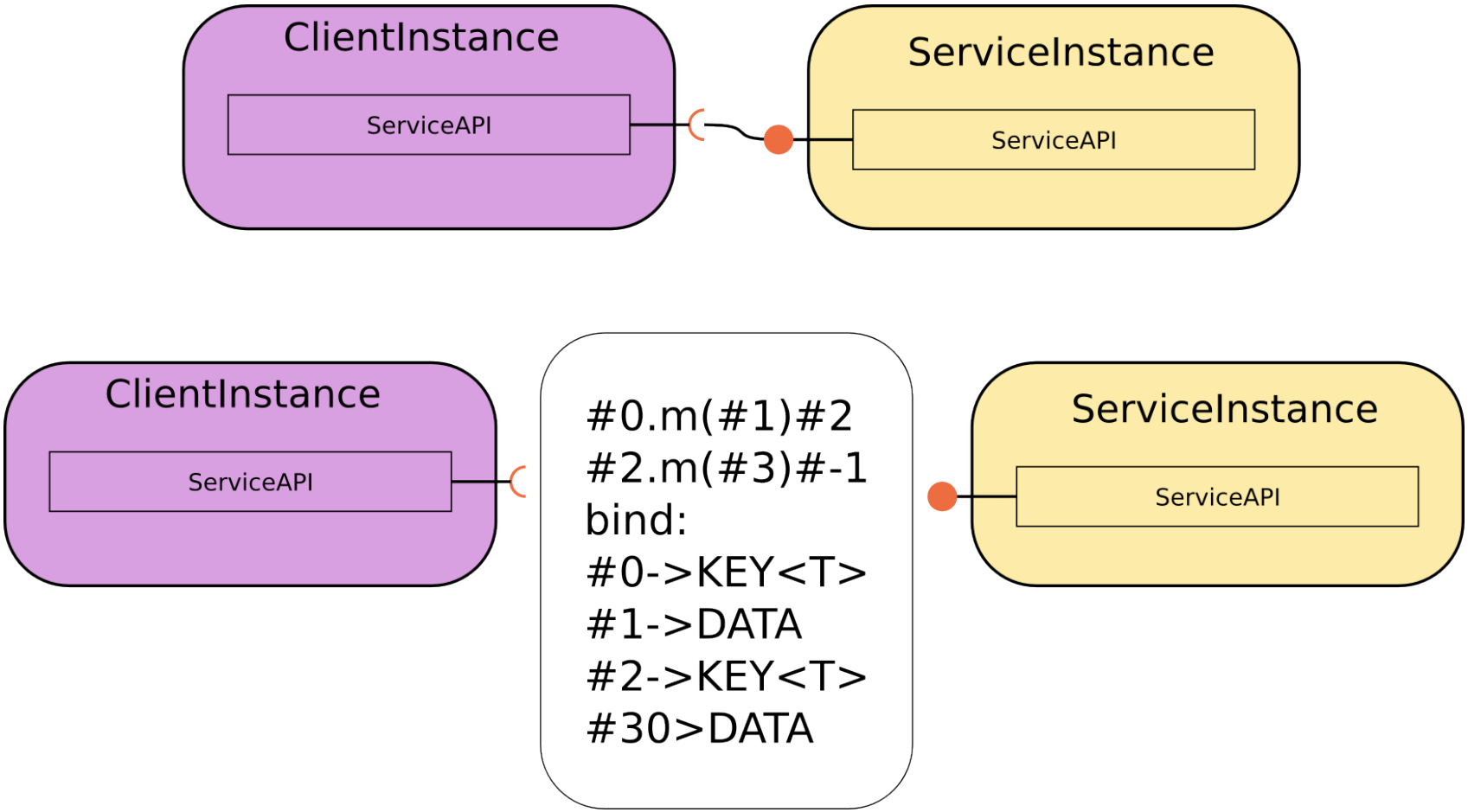
ExternalBindingSwitch

Instancja usługi to moduł uruchomiony z "przełącznikiem" na potrzebne usługi:



(a) Direct (b) Switch

Protokoł kanoniczny



DEMO

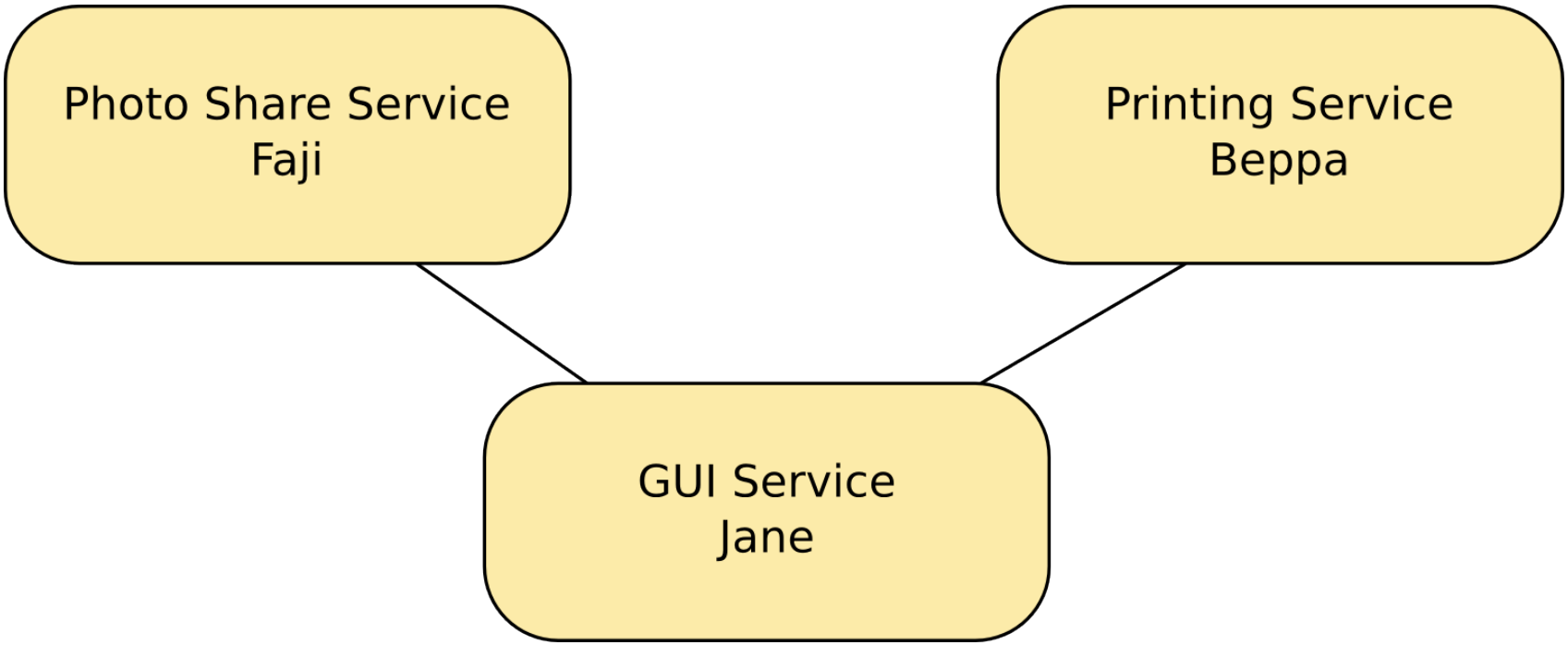


Photo Share Service
Faji
Printing Service
Beppa
GUI Service
Jane