

**adaptTo()**

**APACHE SLING & FRIENDS TECH MEETUP**  
2-4 SEPTEMBER 2019

# From monolith to modules

Dominik Süß, Robert Munteanu, Adobe



adaptTo()

# Welcome

# About us



Dominik Süß  
Software Engineer



Robert Munteanu  
Senior Computer Scientist



# Outline

- Benefits vs cost of modularization
- Decoupling patterns
- Tooling support
- Keeping QA effort under control



adaptTo()

# Modularisation

# Benefits

- Slimmer
- Simpler
- Reduced impact of bugs
- Clear dependencies
- Decoupled module lifecycles

# Costs

- Complexity of API Governance
- Planning & Testing for absence of modules
- Complexity & Fragmentation of deployment & maintenance procedures
- Additional aspect to development flow & tooling

# We need a plan...

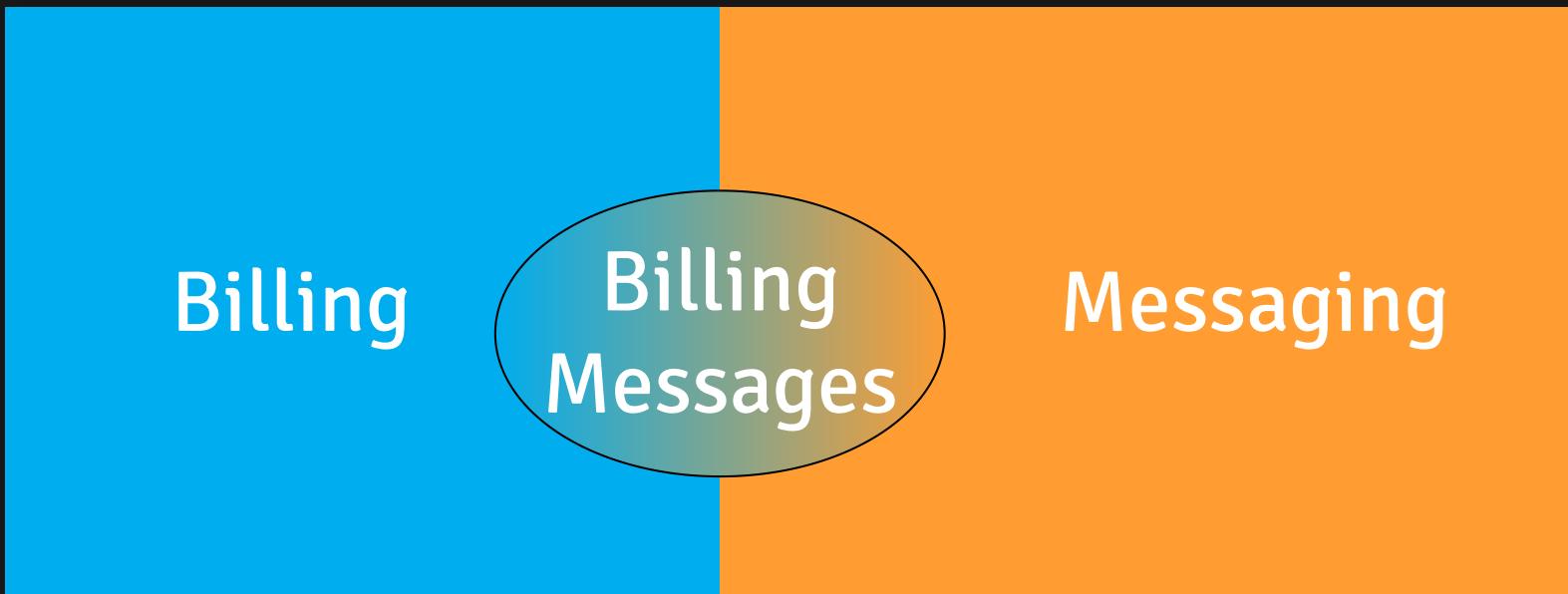
- Global API to modularized API
- Extend & integrate tooling
- Start cutting



# Decoupling patterns - OSGi



# Glue bundles





# Glue bundles

- + Encapsulates the dependencies
- Refactoring to a glue bundle can be difficult
- Glue bundles need to be explicitly deployed when needed



# Optional dependencies

```
<Import-Package>
    com.example.billing.api;resolution:=optional
    ,*
<Import-Package>
```



# Optional dependencies

```
@Component(enabled=false)
public class MyComponent { /* ... */ }

@Component
public class MyComponentStarter {
    protected void activate(ComponentContext ctx) {
        try {
            log.info("Found class {}", BarImpl.class);
            ctx.enableComponent(MyComponent.class.getName());
        } catch ( Throwable t) {
            log.info("Foo component unavailable due to
                    missing Bar module", t);
        }
    }
}
```



# Optional dependencies

- + Minimal impact on existing code bases
- Unnatural pattern to prevent ERROR log entries
- Indicator of low cohesion



# Overlays

```
public interface Foo { }

@Component
public class MyConsumer {
    @Reference(policyOption = ReferencePolicyOption.GREEDY)
    private Foo foo;
}

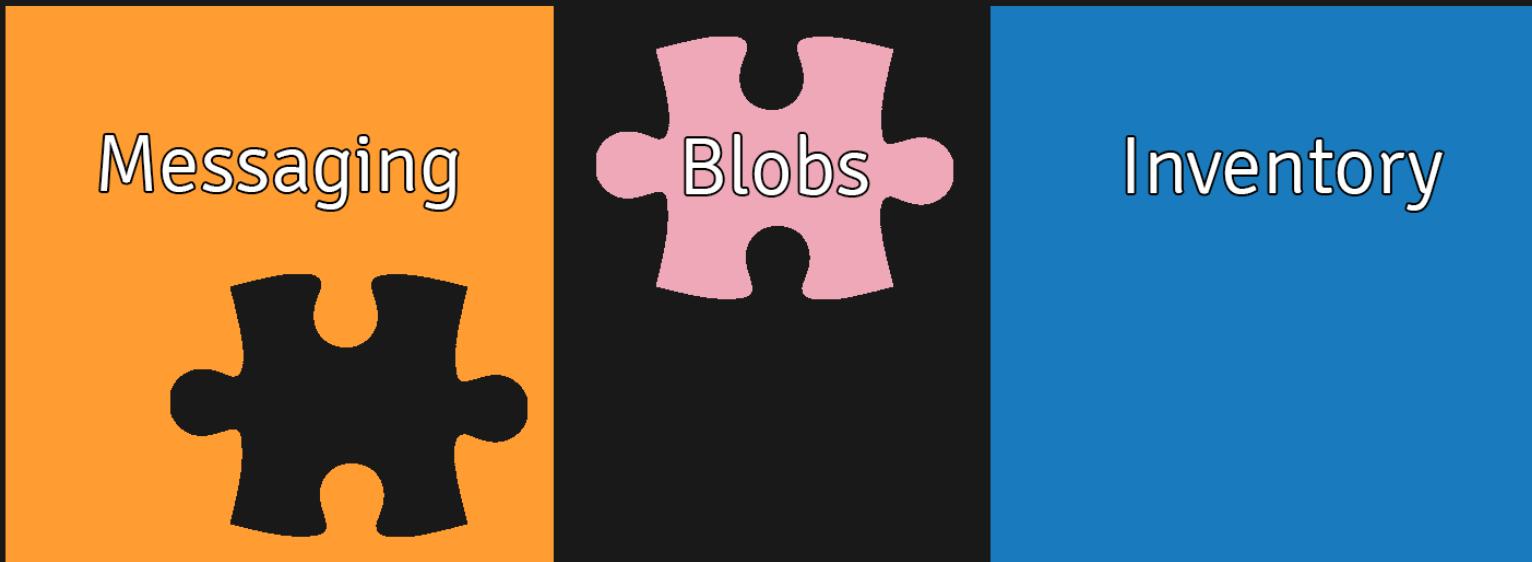
@Component // default service.ranking of 0
public class DefaultFooImpl implements Foo { }

@Component(property = {
    Constants.SERVICE_RANKING +":Integer=100"
})
public class MyBetterFooImpl implements Foo { }
```

# Overlays

- + Simple and well-understood mechanism of overriding service references
- Requires consumers to mark references as greedy
- Service ranking are complicated to manage for large numbers of implementations

# Feature extraction





# Feature extraction

- + Usually cleans up incorrect layering
- Dependencies can spiral out of control



# Decoupling patterns - content



# Glue components

```
private static final RenderCondition INSTANCE =
        new RenderCondition() {
    @Override
    public boolean check() {
        try {
            new Handlebars();
            return true;
        } catch (NoClassDefFoundError e) {
            return false;
        }
    }
};
```



# Glue components

- + Minimal impact on existing code bases
- + Established AEM pattern
- Pushes modularity at the class, instead of the bundle level



# Overlays

```
<execute
    jcr:primaryType="nt:unstructured"
    sling:resourceType="some/workflow/step">
    <executor
        jcr:primaryType="nt:unstructured"
        className="com.adobe.foreign.module.ExecutorImpl"/>
</execute>
```



# Overlays

- + Minimal impact on existing code bases
- Can lead to surprising behaviour at runtime



# Decoupling patterns - generic



# Inlining

- Rule of three → write, duplicate, refactor
- Code duplication is a smell, not a crime
- Refactoring and abstractions are costly



# Decoupling patterns - generic



# Moving checks at build time

Build time >> Test Time >> Run Time

```
[ERROR] Bundle org.apache.sling.caconfig.spi:1.3.4 is importing  
package(s) org.apache.sling.api.resource in start level 20 but  
no bundle is exporting these for that start level.
```



# Potential areas for extension

- Validate content-package Java imports are satisfied
- Generate constraints for discovered workflow steps



# Modularised testing

# Modularised testing

- Combinatorial nightmare
- Optimize CI/CD pipeline to act on module level





adaptTo()

# Summary

# Impact of AEM modularisation

- + Reduced API surface
- + Faster install and startup times
- + Lower memory usage
- + Reduced disk footprint
- More complex development and testing scenarios