APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 26-28 SEPTEMBER 2012

Slice Framework

Łukasz Madrzak-Wecke

Jakub Przybytek

A little context

- Who is Cognifide?
  - " A marketing technology agency with a difference. "

  In a changing landscape, we don't do everything; we specialise.
  - Agile CQ5 projects - size range from 20 to 3000 mandays

adaptTo()

Reasons for creating a framework

- Avoid re-inventing the wheel
- Easily upgrade CQ version (API changes)
- Ease of reusing code
- Lower project entry cost
- Reduce training costs
- Reduce maintenance costs

What we wanted to achieve

- No scriptlets
- Strong typing
- Code and component reusability
- Low cost of introducing developers into projects
- Faster development
- Better code quality

Cognifide „classic” framework

- MVP based
- Achieved almost all features
- Successfully used for years
- A step in the right direction, but:
  - Cost – too many classes
  - Still a lot needs to be done manually
  - No easy implementation swapping
  - Unit testing difficult

Solution?

- Dependency Injection
  - Easy to reuse code
  - Easy to mock
  - Easy to implement complex solutions
- Why Guice?
  - Slightly (5-10%) faster than Spring
  - Lightweight while providing everything we need
- How we did it?
  - Peaberry (for services)
  - <slice:lookup>

Slice = Sling + Guice

- Main features:
  - Dependency injection
  - (Sling)Resource mapping

Slice = Sling + Guice

- Examples:
  - Simple mapped component

```
@SliceResource
public class TextModel {
   @Unescaped
   private String copy;
   private String align;
   public String getCopy() { return this.copy; }
   public String getAlign() { return this.align; }
}
<%@page %>
<slice:lookup var= ”model” type= ”<%= com.TextModel.class %>” />
<div class= ”${model.align}” > ${model.copy} </div>
```

Slice = Sling + Guice

- More features:
  - Annotation based
  - Default scope: request, other scopes possible
  - Complex composition using:
    - Child models
    - Custom properties
    - other…

Slice = Sling + Guice

- More features:
  - Allows to provide abstraction layers
  - Integrates with:
    - Sling: @RequestedPath, @Selectors, ResourceResolver.class
    - CQ: @RequestedPage, PageManager.class
    - OSGi: any service
  - Simple (injector) configuration in bundle activator:

```
injectorServiceRunner = new InjectorServiceRunner(bndlCtx, APP_NAME);
injectorServiceRunner.installModule(new YourModule());
injectorServiceRunner.start();
```

Slice = Sling + Guice

- Examples:
  - More complex injections

```
@SliceResource
public class TextModel {
   // model from child resource named 'socialLinks'
   private SocialLinksModel socialLinks;
   // general purpouse configuration object
   @Inject
   public GlobalConfiguration config;

   (...)
}
```

Slice = Sling + Guice

- Examples:
  - Injecting CQ5 services (Peaberry module)

```
@SliceResource
public class TextModel {
    @Inject
    private PageManager pageManager;
   @Inject
    private ResourceResolverFactory resourceResolverFactory ;
}
// in Guice module configuration
void configure() {
    bind(ResourceResolverFactory.class).toProvider(
        Peaberry.service(ResourceResolverFactory.class).single); }
```

Slice = Sling + Guice

- Testability
  - Improved application design
    - Reduced complexity, better coupling
    - Separation from CQ5 objects
  - Advanced mocking mechanisms:
    - Repository

```
@Repository(imports = {@Content(path="/", resource="test.xml") })
```

    - Request

```
@Request("/content/en/page.selector.html")
```

    - Slice

```
@SliceMockWith(LinkBuilderMock.class)
public LinkBuilder linkBuilder;
```

Slice = Sling + Guice

- Examples:
  - Unit test

```
@Repository @Slice(…)
public class TextModelTest extend SliceTest {
   @Inject private TextModel model;

   @Test @Request(”/content/page.html”);
    public void shouldCreateProperModel()  {
      assert(this.model.getCopy(), ”copy”);
    }
   @Test @Request(”/content/page.blue.html”);
    public void shouldCreateBlueModel()  {
      assert(this.model.getCopy(), ”blue copy”); }
}
```

adaptTo()

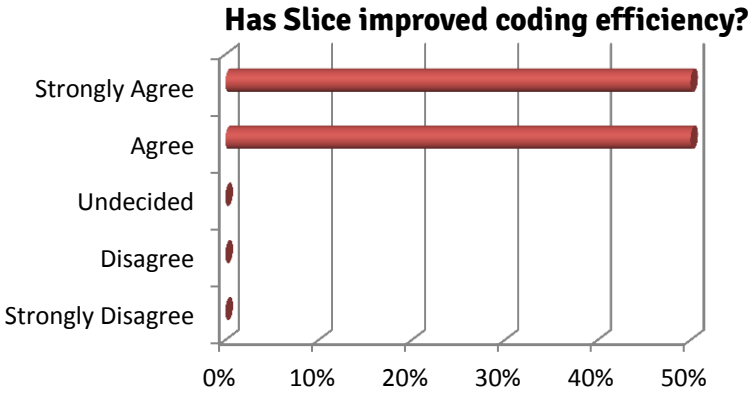
Slice = Sling + Guice

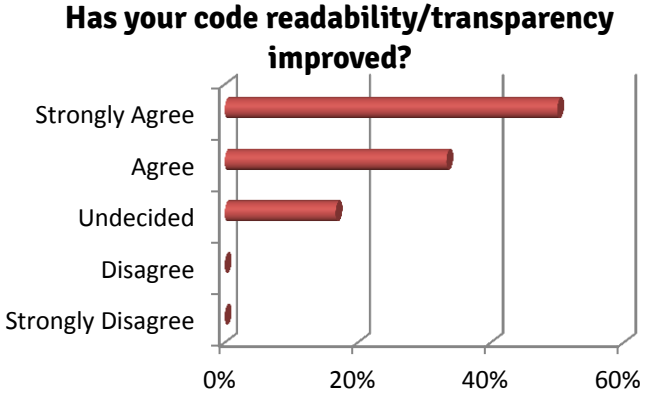
- Advantages
  - Extremely simple mapping from JCR to models
  - Easy injection of models to jsp - <slice:lookup>
  - Significantly less code
  - Easy to reuse code
  - Easy to mock
  - No complex configuration
  - Request scope provide basic caching

Survey

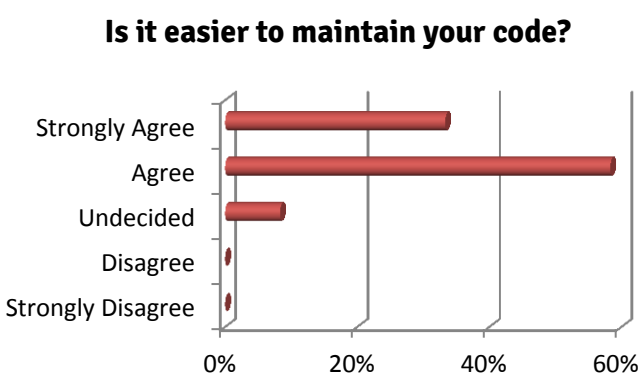
- We made a survey in our company, asking our developers about their impressions after switching to Slice.

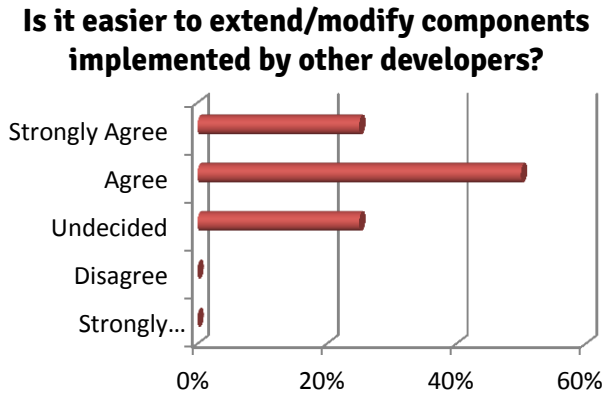
Here are the results:





Survey





Other comments

„Modern, declarative approach. Less amount of classes that are necessary. Good way of splitting classes contract."

„It is difficult to grasp the concept at the beginning (a lot of magical stuff is happening at once and it's slightly unclear) but once one gets used to it, it greatly increases work efficiency."

Slice = Sling + Guice

- Q: How much does it cost?

  A: It's FREE!

- Shared with the community

- Apache License

Slice = Sling + Guice

- Find Slice on github:

  https://github.com/**Cognifide/Slice**

- And few other:

  https://github.com/**Cognifide**

Maven-CRX-Plugin

cq5-open-calais

Sling Dynamic Include

Sling-Caching-Filter

Q&A

- Questions & Answers