



**adaptTo()**

EUROPE'S LEADING AEM DEVELOPER CONFERENCE  
28<sup>th</sup> – 30<sup>th</sup> SEPTEMBER 2020

**Don't use the repository structure as your  
primary abstraction!**

Jörg Hoh / Adobe Systems

# What is this talk about?

- In AEM projects very often low-level features are used, which hardcode repository structure and often are error-prone to use.
- I want to show how you can overcome this problem by introducing proper domain abstractions.



- Jörg Hoh, @joerghoh
- Architect, Adobe
- 10+ years with AEM/CQ5

# What's the problem?

- The only broadly accepted and used abstractions provided by AEM:
  - Page
  - Asset
- And the low-level "Resources", "Nodes" and "Properties"



- **Examples for missing abstractions**
  - **Site** (“What is the homepage for the German site?”)
  - **Product** (“what’s the name of product XY?”)
  - **Product Category** (“Give me all products of category A”)
  - ...

- We just focus on the UI elements of a page and create models for it (CarouselModel, NavigationModel, ...).
- For everything else we use Resources, Nodes, and a lot of utility functions.
- These hardcoded structures and implicit assumptions make evolution very hard.

# CRXDE driven development



```
String path = CONTENT_BRAND + country + "/" + language + "/homepage";  
Resource homepage = resourceResolver.getResource(path);
```

# The anti-pattern – an example

- Get the contact address which is stored on a site level.



# The anti pattern – an example

```
Page getSiteRoot(Resource r) {  
    // iterate up the tree up  
    // until the root page is found  
}
```

Potential NPE!

```
Page root = getSiteRoot(currentPage.getResource());  
Page settings = root.getChild("settings");  
String contactAddress = settings.getValueMap()  
    .get("contactAddress");
```

Hardcoded repository  
structure

# A better version of it

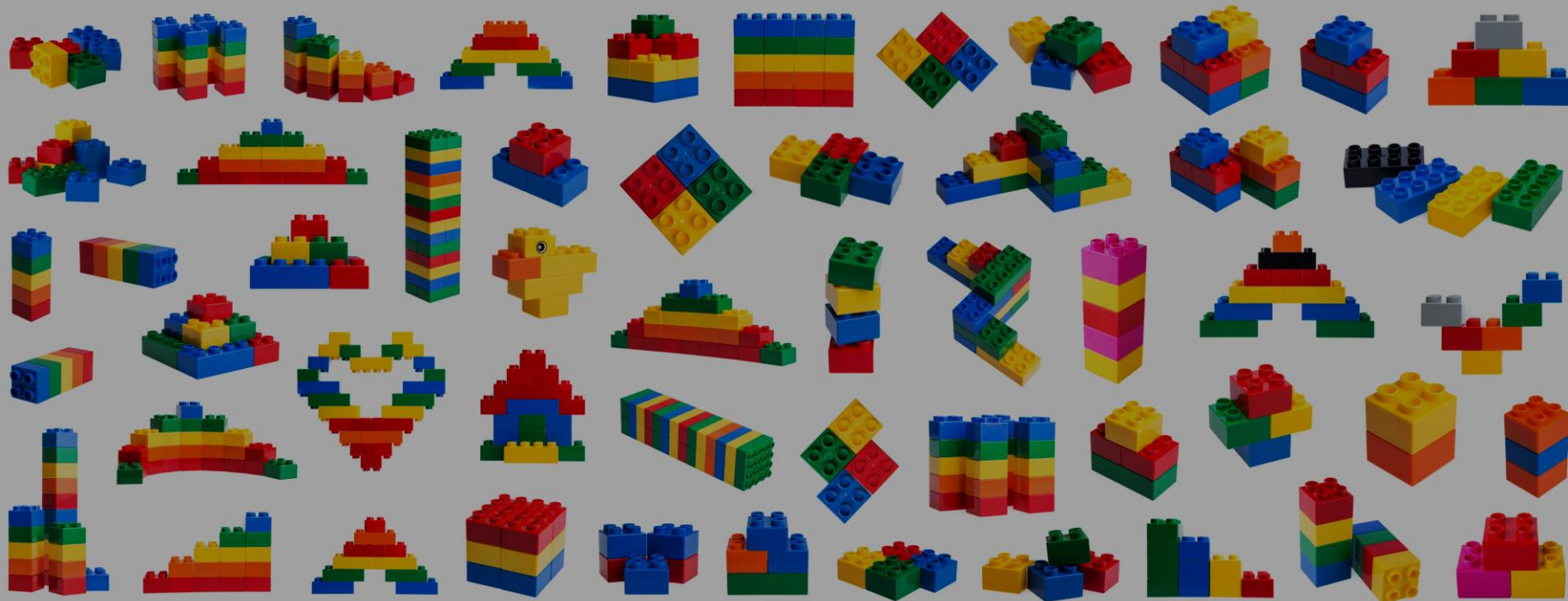
```
String contactAddress =  
Optional.ofNullable(currentPage.adaptTo(Site.class))  
    .map(site -> site.getSettings())  
    .map(settings -> settings.getProperty("contactAddress"))  
    .orElse(return "default");
```

- Error handling included
- No resource, no valueMap, no assumptions
- Easier refactoring, easier reasoning, easier debugging

# Our goal

- Avoid using resources when better abstractions are available!
- If there are no abstractions, create them!
- Do not deal with paths!
- Error cases should be easy to spot and to deal with.

# Build domain objects



# How?

- You normally know all your domain models from the discovery phase of your project
- Model them explicitly
- If they are represented in the repository: make them available via `resource.adaptTo()`

# What's the benefit of it?

- You and your team can practice DRY
  - Avoid all the “get me the Site root” utility methods scattered across you codebase
- Centralize and harmonize the validation of your constraints
- Refactoring of code and content gets much easier

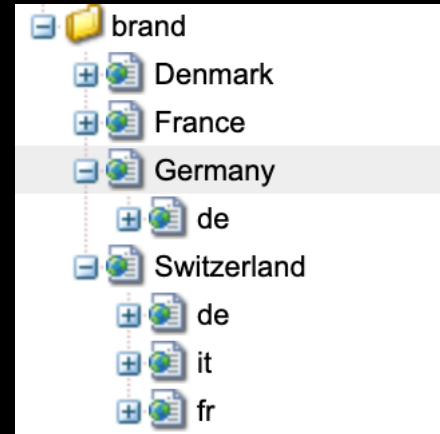
# Example

The missing concept in AEM ...

The “site” object

This 3-level structure is widely known and used

- Brand site
- Country site
- Language site





# Typical operations we need

- To what country belongs the current page?
- Is this page available in other languages?
- What's the contact address for the current site (language specific with country fallback)?
- ...

## To what country belongs currentPage?

```
String countryName =  
    currentPage.adaptTo(CountrySite.class)  
    .getLocalizedCountryName();
```

- We don't care how it is determined.
- And based on the available languages it could even return “Deutschland” when currentPage belongs to de/de and “Germany” for de/en.

## Is this page available in other languages?

```
String relativePath =
    currentPage.adaptTo(LanguageSite.class)
        .getRelativePath(currentpage);

Page[] siblingsInOtherLanguages =
    currentPage.adaptTo(CountrySite.class)
        .getLanguageSites().stream()
        .map(ls -> ls.getRelativePage(relativePath))
        .filter(Objects::NotNull)
        .toArray();
```

# The Implementation of the sites

- Checkout the code at <https://github.com/joerghoh/adapitto2020-domains>

# Remember our goals

- Avoid using resources when better abstractions are available!
- If there are no abstractions, create them!
- Do not deal with paths!
- Error cases should be easy to spot and to deal with.

# Call to action

- Identify the concepts you are constantly using in your daily discussions.
- Implement these concepts and facilitate the adapter pattern to make them universally available.

## Q&A

Questions? Contact me

@joerghoh

<https://cqdump.wordpress.com/about>